37th International Symposium on Theoretical Aspects of Computer Science

STACS 2020, March 10–13, 2020, Montpellier, France

Edited by

Christophe Paul

Markus Bläser

Editors

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

Markus Bläser
Universität des Saarlandes, Saarbrücken, Germany
mblaeser@cs.uni-saarland.de

ACM Classification 2012
Mathematics of computing → Combinatorics; Mathematics of computing → Graph theory; Theory of computation → Formal languages and automata theory; Theory of computation → Logic; Theory of computation → Design and analysis of algorithms; Theory of computation → Computational complexity and cryptography; Theory of computation → Models of computation

ISBN 978-3-95977-140-5

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-140-5.

Publication date
March, 2020

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

Digital Object Identifier: 10.4230/LIPIcs.STACS.2020.0

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020).
Editors: Christophe Paul and Markus Bläser
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

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

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

STACS is held alternately in France and in Germany. This year's conference (taking place March 10-13 in Montpellier) is the 37th in the series. Previous meetings took place in Paris (1984), Saarbrücken (1985), Orsay (1986), Passau (1987), Bordeaux (1988), Paderborn (1989), Rouen (1990), Hamburg (1991), Cachan (1992), Würzburg (1993), Caen (1994), München (1995), Grenoble (1996), Lübeck (1997), Paris (1998), Trier (1999), Lille (2000), Dresden (2001), Antibes (2002), Berlin (2003), Montpellier (2004), Stuttgart (2005), Marseille (2006), Aachen (2007), Bordeaux (2008), Freiburg (2009), Nancy (2010), Dortmund (2011), Paris (2012), Kiel (2013), Lyon (2014), München (2015), Orléans (2016), Hannover (2017), Caen (2018), Berlin (2019).

The interest in STACS has remained at a very high level over the past years. The STACS 2020 call for papers led to 242 submissions with authors from 43 countries. Each paper was assigned to three program committee members who, at their discretion, asked external reviewers for reports. For the sixth 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. The committee selected 56 papers during a three-week electronic meeting held in November/December 2019. This means an acceptance rate of only 23%. As co-chairs of the program committee, we would like to sincerely thank all its members and the 448 external reviewers for their valuable work. In particular, there were intense and interesting discussions inside the PC committee. The overall very high quality of the submissions made the selection an extremely difficult task.

We would like to express our thanks to the three invited speakers: Dana Randal (Georgia Technical Institute, Atlanta, USA), Olivier Bournez (LIX, École Polytechnique, Palaiseau, France), and Martin Grohe (RWTH Aachen University, Germany). Since 2011, the conference program includes tutorials. This year, we are pleased to invite Thomas Schiex (INRAE, Toulouse, France) and Stéphan Thomassé (LIP, ENS Lyon, France) to the tutorial session.

Special thanks go to the local organizing committee for continuous help throughout the conference organization. In particular, we wish to thank the colleagues and student from the ALGCO, ECO and ESCAPE resarch groups for their help as well as Mégane Miquel and Virginie Fèche from LIRMM laboratory staff for her permanent organisation support.

Moreover, we thank Michael Wagner from the Dagstuhl/LIPIcs team for assisting us in the publication process and the final production of the proceedings. These proceedings

37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020).
Editors: Christophe Paul and Markus Bläser
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

contain extended abstracts of the accepted contributions and abstracts of the invited talks and the tutorials. The authors retain their rights and make their work available under a Creative Commons license. The proceedings are published electronically by Schloss Dagstuhl – Leibniz-Center for Informatics within their LIPIcs series. Finally we would like to thank our sponsors for their financial supports: Occitanie Region District, Institut des Sciences de l'Information et leurs Interaction (INS2I) of CNRS; the University of Montpellier and the I-Site MUSE project; the LabEx NUMEV and the LIRMM Laboratory.

Montpellier and Saarbrücken, March 2020 Christophe Paul and Markus Bläser

Conference organization

Program committee

Spyros Angelopoulos	CNRS, Sorbonne Université, Paris, France
V. Arvind	The Institute of Mathematical Sciences (HBNI), Chennai, India
Sayan Bhattacharya	University of Warwick, United Kingdom
Laurent Bienvenu	CNRS, Université de Bordeaux, France
Markus Bläser	Saarland University, Saarbrücken, Germany, co-chair
Manuel Bodirsky	Technische Universität Dresden, Germany
Jop Briët	CWI, Amsterdam, Netherlands
Wojciech Czerwiński	University of Warsaw, Poland
Holger Dell	IT University of Copenhagen, Denmark
Faith Ellen	University of Toronto, Canada
Petr Golovach	Bergen University, Norway
John Hitchcock	University of Wyoming, USA
Christian Ikenmeyer	University of Liverpool, United Kingdom
Shunsuke Inenaga	Kyushu University, Fukuoka, Japan
Christian Konrad	University of Bristol, United Kingdom
Kasper Green Larsen	Aarhus University, Denmark
Ranko Lazic	University of Warwick, United Kingdom
Meena Mahajan	The Institute of Mathematical Sciences (HBNI), Chennai, India
Ulrich Meyer	Goethe Universität, Frankfurt am Main, Germany
Benjamin Monmege	Aix-Marseille Université, France
Christophe Paul	CNRS, Université de Montpellier, France, co-chair
Marcin Pilipczuk	University of Warsaw, Poland
Eva Rotenberg	Technical University of Denmark, Lingby, Denmark
Pierre Senellart	Ecole normale supérieure, Université PSL, France
Till Tantau	Universität zu Lübeck, Germany
Lidia Tendera	Uniwersytet Opolski, Poland
Corentin Travers	Université de Bordeaux, France
Alfredo Viola	Universidad de la República de Uruguay, Uruguay
Georg Zetzsche	The Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany
Thomas Zeume	Technische Universität Dortmund, Germany
Martin Ziegler	KAIST, Daejeon, Republic of Korea
Standa Zivny	University of Oxford, United Kingdom

Steering committee

Thomas Colcombet	CNRS, Université Paris Diderot, France
Martin Dietzfelbinger	Technische Universität Ilmenau, Germany
Arnaud Durand	Université Paris Diderot, France
Christoph Dürr	CNRS, Sorbonne Université, co-chair, France
Henning Fernau	Universität Trier, Germany
Dietrich Kuske	Technische Universität Ilmenau, Germany
Arne Meier	Leibniz Universität Hannover, Germany
Rolf Niedermeier	Technische Universität Berlin, Germany
Natacha Portier	ENS Lyon, France
Gilles Schaeffer	CNRS, Ecole Polytechnique, Palaiseau, France
Thomas Schwentick	Technische Universität Dortmund, co-chair, Germany
Ioan Todinca	Université d'Orléans, France

Local organizing committee (LIRMM, Université de Montpellier, CNRS)

Marin Bougeret	Fabien Jacques	Andrei Romashchenko
Virginie Fèche	Hoang La	Ignasi Sau Valls
Daniel Goncalves	Romain Lebreton	Alexander Shen
Bruno Grenet	Mégane Miquel	Ilaria Zappatore
Emirhan Gürpinar	Christophe Paul (Chair)	
Lucas Isenmann	Alexandre Pinlou	

Sponsors

Statistical Physics and Algorithms

Dana Randall
School of Computer Science, Georgia Institute of Technology, Atlanta, GA 30332-0765, USA
randall@cc.gatech.edu

Abstract

The field of randomized algorithms has benefitted greatly from insights from statistical physics. We give examples in two distinct settings. The first is in the context of Markov chain Monte Carlo algorithms, which have become ubiquitous across science and engineering as a means of exploring large configuration spaces. One of the most striking discoveries was the realization that many natural Markov chains undergo phase transitions, whereby they are efficient for some parameter settings and then suddenly become inefficient as a parameter of the system is slowly modified. The second is in the context of distributed algorithms for programmable matter. Self-organizing particle systems based on statistical models with phase changes have been used to achieve basic tasks involving coordination, movement, and conformation in a fully distributed, local setting. We briefly describe these two settings to demonstrate how computing and statistical physics together provide powerful insights that apply across multiple domains.

2012 ACM Subject Classification Theory of computation → Random walks and Markov chains; Mathematics of computing → Stochastic processes; Theory of computation → Self-organization

Keywords and phrases Markov chains, mixing times, phase transitions, programmable matter

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.1

Category Invited Talk

Funding Funded in part by NSF awards CCF-1526900, CCF-1637031, CCF-1733812 and ARO MURI award #W911NF-19-1-0233.

1 Introduction

Statistical physics employs probabilistic techniques to study systems consisting of large populations. The underlying principles explain numerous physical phenomena, such as magnetism, changes in states of matter, thermal radiation, noise in electronic devices, and more (see, e.g., [16]). In addition, these scientific insights help explain collective behavior across disciplines, including interacting biological systems [22], colloidal mixtures from chemistry [3, 21], segregation models from economics [5, 25], and random graph models in combinatorics [9].

Throughout theoretical computer science, we also find many examples where a statistical physics perspective has enriched the design and analysis of algorithms. A significant example concerns the role of *phase transitions*, showing how micro-scale behavior can induce global, macro-scale changes to a system (see, e.g., [4, 7, 26]). For example, phase transitions in random structures allow us to identify emergent characteristics of a configuration space, such as the birth of the giant component [19]. Moreover, Markov chains have been shown to undergo phase changes in their convergence times, transitioning from *disordered phases*, where they converge to (near) stationarity in polynomial time, to *ordered phases* that require exponential time [7, 10, 26]. More recently, algorithms exhibiting particular phase changes from disordered gaseous or liquid phases to ordered solid phases have proven effective for the design of distributed algorithms for robot swarms and active matter, where we seek collective organization achieving certain tasks [1, 11, 12].

Physical systems define probability measures favoring configurations that minimize energy. Each configuration σ has energy determined by a *Hamiltonian* $H(\sigma)$ and a corresponding weight $w(\sigma) = \mathrm{e}^{-B \cdot H(\sigma)}$, where $B = 1/T$ is inverse temperature. The *Gibbs* (or *Boltzmann*) distribution assigns probabilities proportional to their weight $w(\sigma)$, where configurations with the least energy $H(\sigma)$ have the highest weight and are most likely. However, if there are few of these higher weight configurations, the sample space may be dominated instead by those with small weight, simply because there are many more of them (i.e., there is higher *entropy*), giving these configurations much higher probability overall, even if they are individually less probable. The thermodynamic properties of a physical system, such as specific heat and free energy, are derived from statistical properties of these distributions, and discontinuities in any of these quantities indicate a phase transition between states of matter.

We will provide a small window into the rich marriage between statistical physics and algorithms in the context of Markov chains and programmable matter. Markov chains can become prohibitively slow once the energy from the Hamiltonian outweighs the effects of entropy and the system transitions to an ordered state. In contrast, for programmable matter, we purposefully design algorithms that achieve distinct collective behaviors in their disordered and ordered phases, leading to robust distributed algorithms for self-organizing particle systems.

2 Local Markov Chains

Markov chain Monte Carlo (MCMC) algorithms are ubiquitous throughout science and engineering, providing useful tools for approximate counting, combinatorial optimization and modeling. The main idea is to perform a random walk among a set of configurations so that samples drawn from the limiting distribution are meaningful. For this to be useful, these algorithms need to be efficient, and indeed bounding the converence time of a Markov chain is often the critical step in establishing the efficiency of approximation algorithms based on random sampling. For example, if $G = (V_1, V_2, E)$ is a bipartite graph with $E \subseteq V_1 \times V_2$, then sampling perfect matchings on G allows us to estimate the permanent of the adjacency matrix [20]. Calculating the permanent of a matrix was shown by Valiant to be $\#P$-complete [27], or as hard as counting solutions to any NP-complete problem, so solutions that efficiently produce estimates approximating the exact count are the best we can expect.

Markov chains based on local moves, known as *Glauber dynamics*, are common in practice, primarily because of ther simplicity. As an example, consider the following chain that can be used to sample from the set of independent sets in a given graph, known in statistical physics as the *hard-core lattice gas model*. Given a graph G, the state space Ω is the set of independent sets. We are also given an input parameter λ, known as the *fugacity* (or *activity*). Our goal is to sample from the Gibbs distribution

$$\pi(I) = \lambda^{|I|}/Z,$$

where $|I|$ is the size of independent set I and $Z = \sum_{J \in \Omega} \lambda^{|J|}$ is the normalizing constant known as the *partition function*. We define the Glauber dynamics so that we can move between pairs of configurations that differ by a single vertex, and the celebated Metropolis Algorithm tells us how to implement these moves so that we converge to the Gibbs distribution π, as follows. Starting at any configuration $\sigma \in \Omega$, say the empty independent set (with no vertices), we repeat the following: choose a vertex v at random; if v is in the current independent set, remove it with probability $\min(1, \lambda^{-1})/2$; if it is not in the independent

set, add it with probability $\min(1, \lambda)/2$, if possible; in all other cases, the independent set remains unchanged. It is simple to show that this chain is ergodic and converges to π, so our goal is to determine if it is efficient.

An interesting phenomenon occurs as λ is varied. For small values of λ, Glauber dynamics converge quickly to stationarity, while for large values it is prohibitively slow. To see why, imagine the underlying graph G is an $n \times n$ region of $\mathbb{Z}^2$. Large independent sets dominate the stationary distribution π when λ is sufficiently large and lie on one of the two sublattices (corresponding to each of the two colors of the checkerboard coloring on the dual lattice). When λ is large, it will take exponential time to move from an independent set that lies mostly on the odd sublattice to one that is mostly even. Currently, the best known rigorous bounds verify the the Markov chain converges in polynomial time whenever $\lambda < 2.538$ [26] and requires exponential time when $\lambda > 5.365$ [7, 8].

This type of dichotomy is well known in the statistical physics community, where many models have been shown to abruptly transition from a disordered state to a predominantly ordered one. Physicists observe phase transitions when extending Gibbs distributions to infinite lattices and studying whether there is a unique limiting Gibbs measure, known as a *Gibbs state* (see, e.g., [14]). For the hard-core model on $\mathbb{Z}^2$, is believed that there exists a critical value λ_c such that for $\lambda < \lambda_c$ there is a unique Gibbs state, while for $\lambda > \lambda_c$ there are multiple Gibbs states. This has been verified for small and large values of λ bounded away from the conjectured critical point $\lambda_c \approx 3.79$ in both the computational and physics settings [7, 26].

Fortunately, insights from statistical physics can also allow us to design alternative approaches to sampling in the slow regimes, in some cases. One approach that has proven fruitful far below the critical point (in the slow regime) is based on the *cluster expansion* [18]; at sufficiently low temperatures, configurations have long-range order, and can be precisely defined as small, randomized perturbations from some ground state, or highest probability state. Then configurations can be sampled by first randomly picking a ground state, and then inserting random defects with the appropriate conditional probabilities. A second approach uses *simulated tempering* or *parallel tempering* to sample at low temperatures by dynamically adjusting temperatures up and down during each simulation. These algorithms can be effective when we can (i) generate random samples from a family of temperatures so that low temperature configurations of interest arise often enough, and with the correct conditional probabilities, and (ii) the composite Markov chain on the larger state space (including configurations at all temperatures) converges quickly, even if it is prohibitively slow at low temperatures [6]. Finally, in some contexts it may be possible to rewrite the partition function as a sum over a different family of configurations, and this new representation may suggest alternative Markov chains that are quickly converging, even at low temperatures (see, e.g., [17, 28]).

3 Programmable Matter

Systems of programmable matter can be viewed as collections of simple interacting components with constant-size memory and limited computational capacity. We are interested in how these systems can be made to self-organize to produce emergent behaviors, such as coordination and collective movement.

Using a stochastic approach based on Markov chains, we can design rigorous and robust distributed algorithms for programmable matter exhibiting various desirable properties. For example, for the *compression* problem, our goal is to design an algorithm that allows an

interacting particle system to self-organize and gather together compactly. We say a connected particle system on a planar lattice is α-*compressed* if the perimeter of the ensemble is at most α times the minimum perimeter possible for the n particles, $p_{min} = \Theta(\sqrt{n})$. In [12], we gave a distributed, local Markov chain-based algorithm that solves the compression problem for connected particle systems under the *geometric amoebot model* [13], a formal distributed model in which particles move on the triangular lattice.

Our approach to this and other basic tasks proceeds as follows. We first choose a Hamiltonian $H(\sigma)$ over particle configurations that assigns lower values to preferable (compressed) configurations. The transitions of the Markov chain are then defined to favor configurations with small Hamiltonians. For compression, we let $H(\sigma) = -e(\sigma)$, where $e(\sigma)$ is the number of edges induced by configuration σ, i.e., the number of lattice edges with both endpoints occupied. Setting $\lambda = \mathrm{e}^B$, we get $w(\sigma) = \lambda^{e(\sigma)}$. It is easy to verify that the number of induced edges negatively correlated with the size of the perimeter, so the more induced edges, the more compressed a configuration will be.

Using a Metropolis filter, we design a Markov chain $\mathcal{M}$ that performs local moves and converges to a distribution that generates configurations proportional to their weight $w(\sigma)$. In particular, the probability of a configuration σ is $w(\sigma)/Z$, where $Z = \sum_{\sigma'} w(\sigma')$ is the normalizing constant known as the *partition function*. Using tools from both statistical physics and Markov chain analysis, we prove that, if we wait long enough, non-compressed configurations occupy an exponentially small fraction of probability distribution when λ is sufficiently large.

The Markov chain $\mathcal{M}$ for compression is defined as follows. Starting with an arbitrary configuration σ_0 of n simply connected particles, we define local rules that maintain connectivity throughout the algorithm. There is a *bias parameter* λ given as input, where $\lambda > 1$ corresponds to particles prefering more neighbors and $\lambda < 1$ corresponds to particles prefering fewer neighbors. The moves of the Markov chain $\mathcal{M}$ are carefully designed so that the particle system always remains simply connected, preventing the chain to disconnect or form holes, which still keeping the state space connected via allowable transitions (so $\mathcal{M}$ is ergodic). Moreover, the moves are defined locally so that they can be implemented in a fully distributed setting. Maintaining connectivity makes the analysis of the limiting distribution simpler, but showing ergodicity is more challenging.

Particles individually execute a distributed algorithm defined by $\mathcal{M}$, using Poisson clocks to define when to attempt local moves. We prove that for all $\lambda > 2 + \sqrt{2}$, there is a constant $\alpha = \alpha(\lambda) > 1$ such that at stationarity, with all but exponentially small probability, the particle system will be α-compressed. In fact, we show that for any $\alpha > 1$, there exists λ such that our algorithms achieve α-compression. Moreoever, when λ is small we achieve the inverse property of *expansion*. For all $0 < \lambda < 2.17$, there is a constant $\beta < 1$ such that at stationarity, with all but exponentially small probability, the perimeter will be β-expanded, i.e., the perimeter will be within a β fraction of the maximum perimeter $p_{max} = \Theta(n)$. This implies that for any $0 < \lambda < 2.17$, the probability that the particle system is α-compressed is exponentially small for any constant $\alpha > 1$.

The key ingredient used to establish compression and expansion is a careful *Peierls argument*, used in statistical physics to study non-uniqueness of limiting Gibbs measures and in computer science to establish slow mixing of Markov chains. Because we enforce connectivity throughout the Markov process, our Peierls arguments are significantly simpler than many standard arguments on configurations that are not required to be connected. In subsequent work, we extended these results to the disconnected setting where, in contrast, verifying ergodicity becomes trivial but analyzing the stationary distribution requires more sophisticated tools [15].

One appeal of such a stochastic, distributed algorithm is its robustness. The system can recover from deviations in Poisson clocks waking particles to perform moves, anomalies in our individual particle's movements, and even some particle failures. Moreoever, this stochastic approach provides a general framework that is applicable beyond compression – it has the potential to solve any problem where the objective can be described in terms of minimizing some energy function, provided changes in that energy function can be calculated using only local information. One example is an optimization problem inspired by ant behavior [23] known as *shortcut bridging* where particles maintain bridge structures that balance a efficiency-cost tradeoff [1, 2]. A second example is a self-organizing system achieving *separation*, where particles of different colors can be shown to either intermingle or segregate depending on the settings of parameters [11]. Distributed algorithms based on Markov chains also have provided a theoretical explanation of *phototaxing*, or directed collective motion towards or away from a light source, in an experimental system of swarm robots [24]. Finally, we have promising directions for *alignment* and *flocking*, where oriented particles coordinate to determine a preferred direction of movement. In many of these cases, the collective behavior can be controlled by adjusting whether a physical system is in a disordered (gaseous) or an ordered (solid) state by exploring the physical properties of these systems.

References

1 Marta Andrés Arroyo, Sarah Cannon, Joshua J. Daymude, Dana Randall, and Andréa W. Richa. A stochastic approach to shortcut bridging in programmable matter. In *DNA Computing and Molecular Programming*, DNA23, pages 122–138, 2017.

2 Marta Andrés Arroyo, Sarah Cannon, Joshua J. Daymude, Dana Randall, and Andréa W. Richa. A stochastic approach to shortcut bridging in programmable matter. *Natural Computing*, 17(4):723–741, 2018.

3 Akhilest K. Arora and Raj Rajagopalan. Applications of colloids in studies of phase transitions and patterning of surfaces. *Current Opinion in Colloid & Interface Science*, 2(4), 1997.

4 R. J. Baxter, I. G. Enting, and S. K. Tsang. Hard-square lattice gas. *Journal of Statistical Physics*, 22:465–489, 1980.

5 Prateek Bhakta, Sarah Miracle, and Dana Randall. Clustering and mixing times for segregation models on $\mathbb{Z}^2$. In *Proceedings of the 25th ACM/SIAM Symposium on Discrete Algorithms*, (SODA), 2014.

6 Nayantara Bhatnagar and Dana Randall. Simulated tempering and swapping on mean-field models. *Journal of Statistical Physics*, 164(3):495–530, 2016.

7 Antonio Blanca, Yuxuan Chen, David Galvin, Dana Randall, and Prasad Tetali. Phase coexistence for the hard-core model on $\mathbb{Z}^2$. *Combinatorics, Probability and Computing*, pages 1–22, 2018.

8 Antonio Blanca, David Galvin, Dana Randall, and Prasad Tetali. Coexistence and slow mixing for the hard-core model on $\mathbb{Z}^2$. In *Approximation, Randomization and Combinatorial Optimization (APPROX/RANDOM)*, volume 8096, pages 379–394, 2013.

9 Bela Bollobas. The evolution of random graphs. *Transactions of the American Mathematical Society*, 286(1):257–274, 1984.

10 Christian Borgs, Jennifer T. Chayes, Jeong Han Kim, Alan Frieze, Prasad Tetali, Eric Vigoda, and Van Ha Vu. Torpid mixing of some Monte Carlo Markov chain algorithms in statistical physics. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, pages 218–229, Washington, DC, USA, 1999. IEEE Computer Society.

11 Sarah Cannon, Joshua J. Daymude, Cem Gökmen, Dana Randall, and Andréa W. Richa. A local stochastic algorithm for separation in heterogeneous self-organizing particle systems. In *Approximation, Randomization and Combinatorial Optimization (APPROX/RANDOM)*, pages 54:1–54:22, 2019.

12 Sarah Cannon, Joshua J. Daymude, Dana Randall, and Andréa W. Richa. A Markov chain algorithm for compression in self-organizing particle systems. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, pages 279–288, New York, NY, USA, 2016. ACM.

13 Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. An algorithmic framework for shape formation problems in self-organizing particle systems. In *Proceedings of the Second Annual International Conference on Nanoscale Computing and Communication*, NANOCOM '15, pages 21:1–21:2, 2015.

14 Roland L. Dobrushin. The problem of uniqueness of a gibbsian random field and the problem of phase transitions. *Functional Analysis and Its Applications*, 2:302–312, 1968.

15 Bahnisikha Dutta, Shengkai Li, Sarah Cannon, Joshua J. Daymude, Enes Aydin, Andrea W. Richa, Daniel I. Goldman, and Dana Randall. Programming robot collecitves using mechanics-induced phase changes, 2020. (In preparation).

16 Sacha Friedli and Yvan Velenik. *Statistical Mechanics of Lattice Systems: A Concrete Mathematical Introduction.* Cambridge University Press, Cambridge, 2017.

17 Vivek K. Gore and Mark R. Jerrum. The Swendsen–Wang process does not always mix rapidly. *Journal of Statistical Physics*, 97(1):67–86, 1999.

18 Tyler Helmuth, Will Perkins, and Guus Regts. Algorithmic pirogov-sinai theory. In *Proceedings of the 51st Annual ACM Symposium on the Theory of Computing*, pages 1009–1020, 2019.

19 Svante Janson, Donald Knuth, Tomasz Luczak, and Boris Pittel. The birth of the giant component. *Random Structures and Algorithms*, 4(3):231–358, 1993.

20 Mark R. Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries. *Journal of the ACM*, 51:671–697, 2004.

21 Sarah Miracle, Dana Randall, and Amanda Pascoe Streib. Clustering in interfering binary mixtures. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, APPROX '11, RANDOM '11, pages 652–663, 2011.

22 Gerald H. Pollack and Wei-Chun Chin (Eds.), editors. *Phase Transitions in Cell Biology.* Springer International Publishing, 2008.

23 Chris R. Reid, Matthew J. Lutz, Scott Powell, Albert B. Kao, Iain D. Couzin, and Simon Garnier. Army ants dynamically adjust living bridges in response to a cost–benefit trade-off. *Proceedings of the National Academy of Sciences*, 112(49):15113–15118, 2015.

24 William Savoie, Sarah Cannon, Joshua J. Daymude, Ross Warkentin, Shengkai Li, Andréa W. Richa, Dana Randall, and Daniel I. Goldman. Phototactic supersmarticles. *Artificial Life and Robotics*, 23(4):459–468, 2018.

25 Thomas C. Schelling. Models of segregation. *American Economic Review*, 59(2):488–493, 1969.

26 Alistair Sinclair, Piyush Srivastava, Daniel Stefankovic, and Yintong Yin. Spatial mixing and the connective constant: Optimal bounds. *Probabability Theory Relatated Fields*, 168(1–2):153–197, 2017.

27 Leslie Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

28 Jian-Sheng Wang and Robert H. Swendsen. Nonuniversal critical dynamics in monte carlo simulations. *Physics Review Letters*, 58(2):86–88, 1987.

Weisfeiler and Leman's Unlikely Journey from Graph Isomorphism to Neural Networks

Martin Grohe
RWTH Aachen University, Germany
https://www.lics.rwth-aachen.de/~grohe
grohe@informatik.rwth-aachen.de

Abstract

The Weisfeiler-Leman algorithm is a well-known combinatorial graph isomorphism test going back to work of Weisfeiler and Leman in the late 1960s. The algorithm has a surprising number of seemingly unrelated characterisations in terms of logic, algebra, linear and semi-definite programming, and graph homomorphisms. Due to its simplicity and efficiency, it is an important subroutine of all modern graph isomorphism tools. In recent years, further applications in linear optimisation, probabilistic inference, and machine learning have surfaced. In the first part of my talk, I will give an introduction to the Weisfeiler-Leman algorithm and its various characterisations. In the second part I will speak about its applications, in particular about recent work relating the algorithm to graph neural networks.

2012 ACM Subject Classification Theory of computation → Discrete optimization; Mathematics of computing → Graph algorithms

Keywords and phrases Weisfeiler adn Leman algorithm, Graph isomorphism, Neural network, logic, algebra, linear and semi-definite programming

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.2

Category Invited Talk

37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020).
Editors: Christophe Paul and Markus Bläser; Article No. 2; pp. 2:1–2:1

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

Computability, Complexity and Programming with Ordinary Differential Equations

Olivier Bournez
Laboratoire d'Informatique de l'X (LIX), CNRS, Ecole Polytechnique, Institut Polytechnique de Paris, 91128 Palaiseau Cedex, France
bournez@lix.polytechnique.fr

Abstract

Ordinary Differential Equations (ODEs) appear to be a universally adopted and very natural way for expressing properties for continuous time dynamical systems. They are intensively used, in particular in applied sciences. There exists an abundant literature about the hardness of solving ODEs with numerical methods.

We adopt a dual view: we consider ODEs as a way to program or to describe our mathematical/computer science world. We survey several results considering ODEs under this computational perspective, with a computability and complexity theory point of view. In particular, we provide various reasons why polynomial ODEs should be considered as the continuous time analog of Turing machines for continuous-time computations, or should be used as a way to talk about mathematical logic.

This has already applications in various fields: determining whether analog models of computation can compute faster than classical digital models of computation; solving complexity issues for computations with biochemical reactions in bioinformatics; machine independent characterizations of various computability and complexity classes such as PTIME or NPTIME, or proof of the existence of a universal polynomial ordinary differential equation whose solutions can approximate any continuous function if provided with a suitable well-chosen initial condition.

2012 ACM Subject Classification Theory of computation; Theory of computation → Models of computation; Theory of computation → Computability; Theory of computation → Recursive functions; Computer systems organization → Analog computers; Theory of computation → Complexity classes; Theory of computation → Complexity theory and logic; Mathematics of computing → Differential equations; Mathematics of computing → Ordinary differential equations

Keywords and phrases Ordinary differential equations, Models of computation, Analog Computations, discrete ordinary differential equations, Implicit complexity, recursion scheme

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.3

Category Invited Talk

Funding *Olivier Bournez*: Supported by RACAF Project from Agence National de la Recherche.

Acknowledgements This material is based upon work supported by the RACAF Project from Agence National de la Recherche.

1 Back to the history of analog models of computation

In 1941, Claude Shannon introduced in [42] the *General Purpose Analog Computer* (GPAC) model as a model for the Differential Analyzer [13], on which he worked as an operator.

Differential Analysers were mechanical (and later on electronics) continuous time analog machines. First ever built machine was built under the supervision of V. Bush 1931 at MIT: Applications were from gunfire control up to aircraft design. First differential analyzers, such as the ones at the time of Shannon, were mechanical. Electronic versions were used from late 40s until 70s. Nowadays, company *Analog paradigm* is selling some modern differential analyzers based on operational amplifiers.

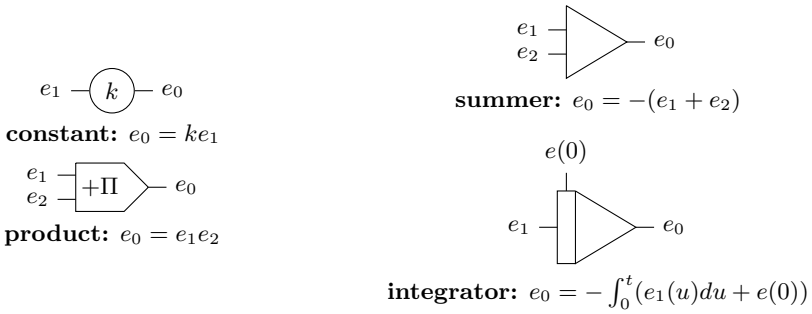


Figure 1 Presentation of the 4 types of units: constant, adder, multiplier, and integrator.

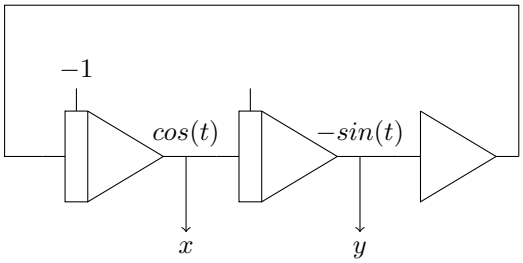


Figure 2 Example of GPAC circuit: computing sine and cosine with two variables.

Basically, a GPAC is any circuit that can be build from the 4 basic units of Figure 1, that is to say from basic units realizing constants, additions, multiplications and integrations, all of them working over analog real quantities (that were corresponding to angles in the mechanical Differential Analysers, and later on to voltage in the electronic versions).

Actually, not all kinds of interconnections must be allowed since this may lead to some undesirable behavior (*e.g.* non-unique outputs. For further details, refer to [29]).

Figures 2 illustrates for example how the sine function can generated using two integrators, with suitable initial state, as being the solution of ordinary differential equation

$$\begin{cases} y'(t) = z(t) \\ z'(t) = -y(t) \end{cases}$$

with suitable initial conditions.

Shannon, in his original paper, already mentioned that the GPAC generates polynomials, the exponential function, the usual trigonometric functions, their inverses, and their composition. More generally, Shannon claimed that all functions generated by a GPAC are differentially algebraic in the sense of the following definition.

▶ **Definition 1.** *A unary function y is differentially algebraic (d.a.) on the interval I if there exists an $n \in \mathbb{N}$ and a nonzero polynomial p with real coefficients such that*

$$p\left(t, y, y', ..., y^{(n)}\right) = 0, \qquad \text{on } I. \tag{1}$$

As a corollary, and noting that the Gamma function $\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt$ is not d.a. [38], we get that

▶ **Proposition 2.** *The Gamma function cannot be generated by a GPAC.*

Another famous example of not d.a. function is Riemann's Zeta function $\zeta(x) = \sum_{k=0}^{\infty} \frac{1}{k^x}$.

2 Polynomial Ordinary Differential Equations (pODEs)

Following corrections and refinements [37, 34, 29, 27] of some statements from Shannon, GPACs can be identified with polynomial Ordinary Differential Equations (pODE), i.e. dynamics of type $y' = p(y)$ where $y(t) = (y_1(t), \ldots, y_d(t)) \in \mathbb{R}^d$ is a (vectorial) function of time, and $p : \mathbb{R}^d \to \mathbb{R}^d$ is a (vector of) polynomials, where d is some integer.

Hence, one can consider that a GPAC is a polynomial ordinary differential equation of *fixed* dimension, or a polynomial initial value (pIVP) if some initial condition $y(t_0) = y_0$ is added.

Following Shannon,

▶ **Definition 3** ([42]). *A function $f : \mathbb{R} \to \mathbb{R}$ is* GPAC-generable *if it corresponds to some (coordinate) projection of a solution of an pIVP: that is there exists a pODE (i.e. some p, t_0, $i_0 \in \{1, 2, \ldots, d\}$ and y_0 over $\mathbb{R}^d$ for some d) such that $f(t) = y_{i_0}(t)$ for all $t \in \mathbb{R}$.*

The concept can naturally be extended to function $f : \mathbb{R}^m \to \mathbb{R}^m$ [42, 2]. From definition, a GPAC-generable function is necessarily (real) analytic.

In modern terms, this concept is strongly related to the so-called class of *Noetherian functions* [36] (and hence also to *Pfaffian* or *differentially algebraic functions*: see e.g. [24]).

The class of generable functions is particularly robust: Several variations on the GPAC circuits were explored and proven to be all equivalent. This essentially shows that the above notion is probably the right definition to be considered [27]. Furthermore, it has very strong closure properties: For example.

▶ **Proposition 4** ([28]). *The class of functions generated by GPACs is closed under the operations $+, -, \times, \div$, under composition, derivation, and compositional inverses (i.e. if f is generated by a GPAC, then so is f^{-1}).*

The following result states that the solution of an initial-value problem defined with functions generated by GPACs is also generated by a GPAC.

▶ **Proposition 5** ([28]). *Consider the initial value problem (IVP)*

$$\begin{cases} x' = f(t, x), \\ x(t_0) = x_0, \end{cases} \tag{2}$$

where $f : \mathbb{R}^{n+1} \to \mathbb{R}^n$ and each component of f is a composition of polynomials and functions generated by GPACs. Then there exist $m \geq n$, a polynomial $p : \mathbb{R}^{m+1} \to \mathbb{R}^m$ and a $y_0 \in \mathbb{R}^m$ such that the solution of (2) is given by the first n components of $y = (y_1, ..., y_m)$, where y is the solution of the polynomial IVP

$$\begin{cases} y' = p(t, y), \\ y(t_0) = y_0. \end{cases} \tag{3}$$

A digression in order to possibly help with intuition: Let us authorize ourselves in this single paragraph to leave the world of formal mathematical statements, to possibly help for intuition: It follows from all these closure properties that *"most" "common"* analytic functions fall in the class of generable functions and turn out to be GPAC-generable, i.e. generable with a differential analyzer. Of course, this last statement is not referring to a well-defined concept of *"most"* and *"common"*, but to some well-known fact to engineers at the time of analog machines: Repeating Shannon [42], *"When the Differential Analyzers was first built it was thought that all functional relationships between terms of the equation*

being solved would have to be introduced into the machine by means of input tables. Soon it was found that practically all the important simple functions could be 'generated' using only integrators and adders.". This phenomenon has similarities with the class of computable functions in classical computability which include all *"most" "common"* functions, even if we can build some non-computable functions (e.g. by diagonalization).

Back to the mathematical world: From Shannon's analysis, as we know some (analytic) Turing computable (in the sense of computable analysis) *non-differentially algebraic* functions (e.g. Euler's ζ and Riemann's Γ) it was deduced that the model is not Turing complete, and the model was mainly forgotten.

▶ Remark 6. Observe that the proofs of the non-differential algebraicity of ζ and Γ by respectively Hilbert and Hölder [31] can consequently be seen as a proof of uncomputability (ungenerability to be more precise) really different from usual methods of (un)computability such as diagonalization.

3 Revisiting computability with polynomial ordinary differential equations

We proved in 2007 [6] that this statement about the limitation of the computational power of the GPAC model comes mainly from a misconception of what was called *computable* at the time of Shannon with respect to what would be called such today: If the input is given as some initial data, and if we allow the machine not to produce its output in real-time, but after *some delay* (i.e. after some *computation time* (encoded by the delay required for y_j to be less than required precision in Theorem 9 which follows)), as in all modern models of computations, then the model can compute any function that is classically (Turing) computable.

Call this notion *GPAC-computability* by opposition to *GPAC-generability*: GPAC-computability does correspond to classical (i.e. Turing) computability [5]. Consequently, the class of GPAC-computable does inherit from all the closure and robustness properties of computable functions and does include all GPAC-generable functions, as well as functions such as Γ, ζ and many others, as soon as we know that there are (Turing) computable.

Formally (see Figure 3 for illustration).

▶ **Definition 7.** *A function $f : [a, b] \to \mathbb{R}$ is GPAC-computable iff there exist some polynomials with polynomial coefficients $p : \mathbb{R}^{n+1} \to \mathbb{R}^n$, $p_0 : \mathbb{R} \to \mathbb{R}$, and $n - 1$ computable real values $\alpha_1, ..., \alpha_{n-1}$ such that:*

1. *$(y_1, ..., y_n)$ is the solution of the Cauchy problem $y' = p(y, t)$ with initial condition $(\alpha_1, ..., \alpha_{n-1}, p_0(x))$ set at time $t_0 = 0$*
2. *There are $i, j \in \{1, ..., n\}$ such that $\lim_{t\to\infty} y_j(t) = 0$ and $|f(x) - y_i(t)| \leq y_j(t)$ for all $x \in [a, b]$ and all $t \in [0, +\infty)$.*

We remark that $\alpha_1, \ldots, \alpha_{n-1}$ are auxiliary parameters needed to compute f.

The following is true with this notion of computation:

▶ **Proposition 8** ([27])**.** *The Gamma function Γ is GPAC-computable.*

And more generally:

▶ **Theorem 9.** *Let a and b be computable reals. A function $f : [a, b] \to \mathbb{R}$ is computable in the sense of computable analysis iff it is GPAC-computable.*

Figure 3 Graphical illustration of Definition 7.

Figure 4 A continuous system before and after an exponential time speed-up.

More importantly, to prove so, we developed some *ad hoc* techniques to program with pODEs: how to realize *assignments*, *iterations*, *loops,* etc. with differential equations. This led us to understand that pODEs play a very particular role, in particular, if one wants to go to complexity theory, and not only computability theory.

4 Complexity theory with polynomial ordinary differential equations

This connection between Turing computability and the GPAC was recently refined [9, 8] at the level of complexity, with a characterization of the class PTIME and polynomial time computable real functions.

Defining a robust time complexity notion for continuous time systems was a well-known open problem [4], with several attempts, but with no generic solution provided. In short, the difficulty is that the naive idea of using the time variable of the ODE as a measure of "time complexity" is problematic, since time can be arbitrarily contracted in a continuous system due to the "Zeno phenomena" (here *time-contraction*). For example, consider a continuous system defined by an ODE $y' = f(y)$ where $f : \mathbb{R} \to \mathbb{R}$ and with solution $\theta : \mathbb{R} \to \mathbb{R}$. Now consider the following system $y' = f(y)z$, $z' = z$ with solution $\phi : \mathbb{R}^2 \to \mathbb{R}^2$. This system re-scales the time variable and that its solution $\phi = (\phi_1, \phi_2)$ is given by $\phi_2(t) = e^t$ and $\phi_1(t) = \theta(e^t)$ (see Figure 2). Therefore, the second ODE simulates the first ODE, with an exponential acceleration. Similarly, it is also possible to present an ODE which has a solution with a component $\varphi_1 : \mathbb{R} \to \mathbb{R}$ such that $\varphi_1(t) = \phi(\tan t)$, i.e. it is possible to contract the whole real line into a bounded set. Thus any language computable by the first system (or, in general, by a continuous system) can be computed by another continuous system in time $O(1)$. This problem appears not only for PIVPs (or, equivalently, GPACs), but also for many continuous models (see e.g. [40], [41], [35], [3], [14], [21], [18], [19]). We illustrated here *time contraction*, but a *space contraction* (or both simultaneously) is also possible, using a change of variable on space variable y.

We solved the above mentioned open problem for time complexity and proved that a robust notion of *computation time* for a pIVP is provided by *the length* of the solution (up to polynomial time) in [9, 8].

This has been established by proving that a time t Turing machine can be simulated (we established this direction using ad hoc suitable pODE programming) in a length $poly(t)$, and conversely that pODE can be solved (we established this direction using an original numerical method) by a Turing machine in a time polynomial in the length of the solution.

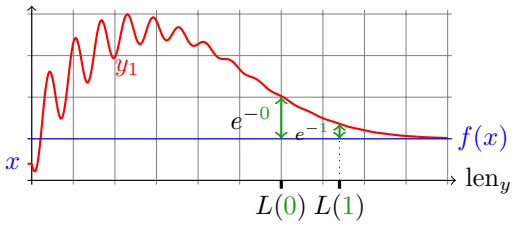


Figure 5 Graphical representation of Theorem 2: on input x, starting from initial condition $(x, 0, \dots, 0)$, the pODE $y' = p(y)$ ensures that $y_i(t)$ gives $f(x)$ with accuracy better than $e^{-\mu}$ as soon as the length of y (from 0 to t) is greater than $L(\mu)$. Other variables $y_1, \dots, y_{i-1}, y_{i+1}, \dots, y_n$ are not plotted and the horizontal axis measures the length of y (instead of time t).

Formally: We recall that the length of a curve $y \in C^1(I, \mathbb{R}^n)$ defined over some interval $I = [a, b]$ is given by $\text{len}_y(a, b) = \int_I \|y'(t)\|_2 \, dt$.

▶ **Theorem 10** ([8], Equivalence between GPAC and CA (Complexity)). *A function $f : [a, b] \to \mathbb{R}$ is computable in polynomial time (in the sense of Computable Analysis) if and only if there exists a polynomial $L : \mathbb{R}_+ \to \mathbb{R}_+$, an integer d, $i \in \{1, 2, \dots, d\}$ and a vector of polynomials $p : \mathbb{R}^d \to \mathbb{R}^d$ with coefficients in $\mathbb{Q}$, such that for any $x \in [a, b]$, the unique solution $y : \mathbb{R}_+ \to \mathbb{R}^d$ to*

$$y(0) = (x, 0, \dots, 0), \qquad y'(t) = p(y(t))$$

satisfies for all $t \in \mathbb{R}_+$:

- *for any $\mu \in \mathbb{R}_+$, if $\text{len}_y(0, t) \geqslant L(\mu)$ then $|f(x) - y_i(t)| \leqslant e^{-\mu}$,*
- *$\|y'(t)\| \geqslant 1$.*

In other words, the precision of y_i increases with the length of the curve. More precisely, as soon as the length between 0 and t is at least $L(\mu)$, the precision is at least $e^{-\mu}$. Notice how rescaling the curve would not help here since it does not change the length of y. The second condition on the derivative of y prevents some pathological cases and ensures that curve has infinite length, and thus that y_i indeed converges to $f(x)$. It is possible to extend this equivalence to multivariate functions and unbounded input domains such as $\mathbb{R}$, by making L take into account the norm of x.

Previous statement is a characterization of functions computable in polynomial time, i.e. $\text{FPTIME}_{\mathbb{R}}$ over the reals. If one wants to talk about decision problems over alphabet $\{0, 1\}$, it possible to define the class PTIME directly in terms of differential equations, by encoding words with rational numbers. Again the length plays a crucial role, but since a differential equation does not "stop", the component y_i is used to signal that it accepts ($y_i \geqslant 1$) or rejects ($y_i \leqslant -1$).

▶ **Theorem 11** ([8], Analog characterization of PTIME). *A language $\mathcal{L} \subseteq \{0, 1\}^*$ belongs to* PTIME*, the class of polynomial time decidable languages, if and only if there exist a polynomial $L : \mathbb{N} \to \mathbb{N}$, an integer d, $i \in \{1, 2, \dots, d\}$ and a vector of polynomials $p : \mathbb{R}^d \to \mathbb{R}^d$ with coefficients in $\mathbb{Q}$, such that for all words $w \in \{0, 1\}^*$, the unique solution $y : \mathbb{R}_+ \to \mathbb{R}^d$ to*

$$y(0) = (0, \dots, 0, |w|, \psi(w)), \qquad y'(t) = p(y(t))$$

where $\psi(w) = \sum_{i=1}^{|w|} w_i 2^{-i}$, satisfies for all $t \in \mathbb{R}_+$:

- *if $|y_i(t)| \geqslant 1$ then $|y_i(u)| \geqslant 1$ for all $u \geqslant t \geqslant 0$ (and similarly for $|y_i(t)| \leqslant -1$),*
- *if $w \in \mathcal{L}$ (resp. $\notin \mathcal{L}$) and $\text{len}_y(0, t) \geqslant L(|w|)$ then $y_i(t) \geqslant 1$ (resp. $\leqslant -1$),*
- *$\|y'(t)\| \geqslant 1$.*

One clear interest of the previous statements is that they provide a way to define classical concepts from the theory of computation (computable function, polynomial time computable functions) only using concepts from analysis, namely polynomial ordinary differential equations.

This has many consequences: This provides a way to define classical concepts from the theory of computation using only concepts from analysis: Theorem 10 provides a definition of computability for functions more natural than the classical digression through type 2 computability (see e.g. [43]), in particular when presenting the concept to people working in analysis. This transforms the question whether analog models can solve faster problems (even about the discrete) than classical means to a question about lengths of solutions. This proves that pODE solving is (PTIME-)complete in its length and solves the issue of avoidance of the so-called Zeno's phenomenon in the verification community in models of hybrid and cyberphysical systems [8]. This also brings new perspectives on some philosophical concepts such as time for systems of our physical world.

5 Some applications of pODE programming

Actually, more generally, ODEs is *universal language* in many domains, in particular in applied fields (physics, biology, ...). We already used our *"ODE programming technology"* to solve several open problems in various fields. We illustrate this with several examples in the rest of this article.

5.1 A universal ordinary differential equation

In computer algebra, a well-known surprising result is due to L. A. Rubel who proved in 1981 in [39] that there exists a fixed non-trivial fourth-order polynomial differential algebraic equation (DAE)

$$p(y, y', \ldots, y^d) = 0$$

such that for any continuous positive function φ on the reals, and for any continuous positive function $\epsilon(t)$, it has a $\mathcal{C}^\infty$ solution with $|y(t) - \varphi(t)| < \epsilon(t)$ for all t. Lee A. Rubel provided an explicit example of such a polynomial DAE [39]. Other examples of universal DAE have later been proposed by other authors.

However, while this result may seem very surprising, its proof is quite simple, is frustrating, and is mainly an indication that DAE is too loose a model for modeling purpose:

- First, the involved notions of universality is far from usual notions of universality in computability theory because the proofs heavily rely on the fact that constructed DAE does not have unique solutions for a given initial data. Indeed, in general a DAE may not have a unique solution, given some initials conditions. But Rubel's DAE *never* has a unique solution, even with a countable number of conditions of the form $y^{(k_i)}(a_i) = b_i$. This is very different from usual notions of universality where one would expect that there is clear unambiguous notion of evolution for a given initial data, for example as in computability theory.
- Second, the proofs usually rely on solutions that are piecewise defined. Hence they cannot be analytic, while analycity is often a key expected property in experimental sciences.
- Third, the proofs of these results can be interpreted more as the fact that (fourth-order) polynomial algebraic differential equations is a too loose a model compared to classical ordinary differential equations. In particular, one may challenge whether the result is really a universality result.

The question whether one can require the solution that approximates φ to be the unique solution for a given initial data was a well-known open problem [39, page 2], [1, Conjecture 6.2]. In [10], using ODE programming, we solved it and showed that Rubel's statement holds for polynomial ordinary differential equations, and since polynomial ODEs have a unique solution given an initial data, this positively answers Rubel's open problem:

▶ **Theorem 12** ([11], Universal PIVP). *There exists a* fixed *polynomial vector p in d variables with rational coefficients such that for any functions $f \in C^0(\mathbb{R})$ and $\varepsilon \in C^0(\mathbb{R}, \mathbb{R}_+^*)$, there exists $\alpha \in \mathbb{R}^d$ such that there exists a unique solution $y : \mathbb{R} \to \mathbb{R}^d$ to $y(0) = \alpha$, $y' = p(y)$. Furthermore, this solution satisfies that $|y_1(t) - f(t)| \leqslant \varepsilon(t)$ for all $t \in \mathbb{R}$, and it is analytic.*

Furthermore, α can be computed from f and ε (in the sense of Computable Analysis).

Using the fact that polynomial ODEs can be transformed into differentially algebraic equations (see [15]), the following then follows.

▶ **Theorem 13** ([11], Universal DAE). *There exists a* fixed *polynomial p in $d+1$ variables with rational coefficients such that for any functions $f \in C^0(\mathbb{R})$ and $\varepsilon \in C^0(\mathbb{R}, \mathbb{R}_+^*)$, there exists $\alpha_0, \ldots, \alpha_{d-1} \in \mathbb{R}$ such that there exists a unique* analytic *solution $y : \mathbb{R} \to \mathbb{R}$ to $y(0) = \alpha_0, y'(0) = \alpha_1, \ldots, y^{(d-1)}(0) = \alpha_{d-1}$, $p(y, y', \ldots, y^d) = 0$. Furthermore, this solution satisfies that $|y(t) - f(t)| \leqslant \varepsilon(t)$ for all $t \in \mathbb{R}$.*

Furthermore, α can be computed from f and ε (in the sense of Computable Analysis).

5.2 Strong Turing completeness of biochemical reactions

Ordinary differential equations is a well-used models for modeling the dynamics of chemical reactions. In particular, the Turing completeness of kinetic reactions was an open problem (see e.g. [17, Section 8]) that we solved using previous polynomial ODE programming technology: we prove the Turing completeness of chemical reaction networks over a finite set of molecular species under the differential semantics in [23].

A biochemical reaction system is a *positive* dynamical system living in the cone $\mathbb{R}_+^n$, where the state is defined by the positive concentration values of the molecular species. We restrict to elementary reaction systems, governed by the mass-action-law kinetics and where each reaction has at most two reactants.

In short, let $\mathcal{M}$ be a *finite set* of n molecular species $\{y_1, \ldots, y_n\}$.

▶ **Definition 14** ([22]). *A* reaction *is a triple (R, P, f), where $R : \mathcal{M} \to \mathbb{N}$ is a multiset of reactants, $P : \mathcal{M} \to \mathbb{N}$ is a multiset of products and $f : \mathbb{R}_+^n \to \mathbb{R}_+$, called the* rate function*, is a partially differentiable function verifying $R(y_i) > 0$ iff $\frac{\partial f}{\partial y_i}(y) > 0$ for some $y \in \mathbb{R}_+^n$.*

A reaction system *is a finite set of reactions.*

A mass-action-law reaction *is a reaction in which the rate function f is a monomial of the form $k * \Pi_{y \in \mathcal{M}} y^{R(y)}$ where k is called the* rate constant*.*

An elementary reaction *is a mass-action-law reaction with at most two reactants.*

▶ **Definition 15** ([22]). *The differential semantics of a reaction system $\{(R_i, P_i, f_i)\}_{i \in I}$ is the ODE system*

$$\{y' = \Sigma_{i \in I}(R_i(y) - Pi(y)) * f_i\}_{y \in \mathcal{M}}.$$

The dynamics given by the law of mass action leads to a polynomial ODE system of the form $y'(t) = p(y(t))$ with $p(y)_i = \sum_j (P_j(y_i) - R_j(y_i) * k_j * \Pi_{i=1}^n {y_i}^{R_j(y_i)}$. There are thus additional constraints, compared to general polynomial ordinar differential equations: the components y_i must always be positive, and the monomials of p_i whose coefficient is

negative must have a non-zero y_i exponent. These constraints are necessary conditions for the existence of a set of biochemically realizable reactions that react according to the dynamics $y' = p(y)$.

Interestingly, we prove that the previous computability and complexity results can be generalized to elementary biochemical reaction systems. First, the restriction to positive systems is shown complete, by encoding each component y_i by the difference between two positive components y_i^+ and y_i^-, which can be normalized by a mutual annihilation reaction, $y_i^+ + y_i^- \Rightarrow \_$, so that one variable is null.

▶ **Definition 16.** *A function $f : \mathbb{R}_+ \to \mathbb{R}_+$ is* chemically-computable *if there exist a mass-action-law reaction system $\{(R_i, P_i, f_i)\}_{i\in I}$ over some molecular species $\{y_1, ..., y_n\}$, and a polynomial $q \in {\mathbb{R}_+}^n[\mathbb{R}_+]$ defining the initial concentration values, such that f is GPAC-computed by q and its (polynomial) differential semantics $p \in {\mathbb{R}_+}^n[{\mathbb{R}_+}^n]$.*

A function $f : \mathbb{R}_+ \to \mathbb{R}$ is chemically-computable *if there exists a chemically computable function $f^+ : \mathbb{R}_+ \to {\mathbb{R}_+}^2$ over $\{y_1^+, ..., y_n^+, y_1^-, ..., y_n^-\}$ such that $f = f_1^+ - f_2^-$.*

In this definition, to compute $f(x)$, one has thus to design a reaction system over a finite set of molecular species, initialized to some values defined by a vector of polynomials $q(x)$ (e.g. following [12, 16]), which guarantees that the result is obtained in the concentration of one distinguished molecular species, with a precision indicated by another distinguished molecular species (see Definition 7).

We prove.

▶ **Theorem 17** ([23]). *Any GPAC-computable function can be computed by a mass-action-law reaction system under the differential semantics preserving the polynomial length complexity.*

▶ **Theorem 18** ([23]). *Elementary reaction systems on finite universes of molecules are Turing-complete under the differential semantics.*

5.3 Revisiting classical computability theory with discrete ODEs

We also came to a discrete counterpart of classical continuous ODEs: discrete ODEs: Its associated derivative notion, called *finite differences*, has been widely studied in numerical optimization for function approximation [25] and is reminiscent in *discrete calculus* [30, 26, 32, 33] for combinatorial analysis. Similarities between discrete and continuous statements have also been historically observed, under the terminology of *umbral* or *symbolic calculus* as early as in the 19th century. However, even if the underlying computational content of finite differences theory is clear and has been pointed out many times, no fundamental connections with algorithms and complexity have been exhibited so far.

We started to demonstrate that discrete ODEs is a very natural tool for algorithm design and to prove that complexity and computability notions can be elegantly and simply captured using discrete ordinary differential equations. We illustrated this by providing a characterization of FPTIME, the class of polynomial time computable functions, and of its non deterministic analog FNP. To this aim, we also demonstrated how some notions from the analog world such as linearity of differential equations or derivatives along some particular functions (i.e. changes of variables) are representative of a certain computational hardness and can be used to solve efficiently some (classical, digital) problems.

More concretely, we focus on functions over the integers of type $f : \mathbb{N}^p \to \mathbb{Z}^d$, or functions of type $f : \mathbb{Z}^p \to \mathbb{Z}^d$, for some integers p, d. The basic idea is to consider the following concept of derivative, introduced here for the case where $p = 1$. We will later on consider more general functions, with partial derivatives instead of derivatives (defined then as expected).

▶ **Definition 19** (Discrete Derivative). *The discrete derivative of $f(x)$ is defined as $\Delta f(x) = f(x+1) - f(x)$. We will also write f' for $\Delta f(x)$ to help to understand statements with respect to their classical continuous counterparts.*

We proposed to introduce the following variation on the notion of derivation: derivation along some function $\mathcal{L}(x, y)$.

▶ **Definition 20** ($\mathcal{L}$-ODE). *Let $\mathcal{L} : \mathbb{N}^{p+1} \to \mathbb{Z}$. We write*

$$\frac{\partial f(x,y)}{\partial \mathcal{L}} = \frac{\partial f(x,y)}{\partial \mathcal{L}(x,y)} = h(f(x,y), x, y), \tag{4}$$

as a formal synonym for $f(x+1, y) = f(x, y) + (\mathcal{L}(x+1, y) - \mathcal{L}(x, y)) \cdot h(f(x, y), x, y)$.

▶ Remark 21. This is motivated by the fact that the latter expression is similar to classical formula for classical continuous ODEs:

$$\frac{\delta f(x,y)}{\delta x} = \frac{\delta \mathcal{L}(x,y)}{\delta x} \cdot \frac{\delta f(x,y)}{\delta \mathcal{L}(x,y)}.$$

This allows us to simulate suitable changes of variables using this analogy. We talk about $\mathcal{L}$-Initial Value Problem (IVP) when some initial condition is added. An important special case is when $\mathcal{L}(x, y)$ corresponds to the bit length $\mathcal{L}(x, y) = \ell(x)$ function: we call this special case length-ODEs. Let $\mathsf{sg}(x) : \mathbb{Z} \to \mathbb{Z}$ denotes the function that takes value 1 for $x > 0$ and 0 in the other case.

▶ **Definition 22.** *A sg-polynomial expression $P(x_1, ..., x_h)$ is an expression built-on $+, -, \times$ (often denoted $\cdot$) and $\mathsf{sg}()$ functions over a set of variables $V = \{x_1, ..., x_h\}$ and integer constants. The degree $\deg(x, P)$ of a term $x \in V$ in P is defined inductively as follows:*

- $\deg(x, x) = 1$ *and for $x' \in X \cup \mathbb{Z}$ such that $x' \neq x$,* $\deg(x, x') = 0$
- $\deg(x, P + Q) = \max\{\deg(x, P), \deg(x, Q)\}$
- $\deg(x, P \times Q) = \deg(x, P) + \deg(x, Q)$
- $\deg(x, \mathsf{sg}(P)) = 0$

A sg-polynomial expression P is essentially constant *in x if* $\deg(x, P) = 0$.

Compared to the classical notion of degree in polynomial expression, all subterms that are within the scope of a sign function contributes 0 to the degree. A vectorial function (resp. a matrix or a vector) is said to be a sg-polynomial expression if all its coordinates (resp. coefficients) are. It is said to be *essentially constant* if all its coefficients are.

▶ **Definition 23.** *A (possibly vectorial) sg-polynomial expression $g(f(x, y), x, y)$ is* essentially linear *in $f(x, y)$ if it is of the form $g(f(x, y), x, y) = A[f(x, y), x, y] \cdot f(x, y) + B[f(x, y), x, y]$ where A and B are sg-polynomial expressions essentially constant in $f(x, y)$.*

▶ **Example 24.** The expression $P(x, y, z) = x \cdot \mathsf{sg}((x^2 - z) \cdot y) + y^3$ is linear in x, essentially constant in z and not linear in y. The expression $P(x, 2^{\ell(y)}, z) = \mathsf{sg}(x^2 - z) \cdot z^2 + 2^{\ell(y)}$ is essentially constant in x, essentially linear in $2^{\ell(y)}$ (but not essentially constant) and not essentially linear in z. The expression: $\mathsf{if}(x, y, z) = y + \bar{\mathsf{sg}}(x) \cdot (z - y) = y + (1 - \mathsf{sg}(x)) \cdot (z - y)$ is essentially constant in x and linear in y and z.

▶ **Definition 25.** *Function f is linear $\mathcal{L}$-ODE definable (from u, g and h) if it corresponds to the solution of $\mathcal{L}$-IVP*

$$\frac{\partial f(x,y)}{\partial \mathcal{L}} = u(f(x,y), h(x,y), x, y) \qquad f(0, y) = g(y) \tag{5}$$

where u is essentially linear *in $f(x, y)$. When $\mathcal{L}(x, y) = \ell(x)$, such a system is called linear length-ODE.*

▶ **Definition 26** ($\mathbb{DL}$). *Let $\mathbb{DL}$ be the smallest subset of functions, that contains* $\mathbf{0}$, $\mathbf{1}$, *projections π_i^p, the length function $\ell(x)$, the addition function $x+y$, the subtraction function $x-y$, the multiplication function $x \times y$ (often denoted $x \cdot y$), the sign function $\mathsf{sg}(x)$ and closed under composition (when defined) and linear length-ODE scheme.*

▶ **Theorem 27** ([7]). $\mathbb{DL} = \text{FPTIME}$

Previous ideas used here for FPTIME can also be extended to provide a characterization of other complexity classes. This includes the possibility of characterizing the class FNP as we did in [7]. Our current investigations concern $\text{PTIME}_{[0,1]}$ of functions computable in polynomial time over the reals in the sense of computable analysis, or more general classes of classical discrete complexity theory such as FPSPACE.

More generally, it is also very instructive to revisit classical algorithmic under this viewpoint, and for example one may realize that even inside class PTIME, the Master Theorem (see e.g. [20, Theorem 4.1] for a formal statement) can be basically read as a result on (the growth of) a particular class of discrete time length ODEs. Several recursive algorithms can then be reexpressed as particular discrete ODEs of specific type.

References

1 Michael Boshernitzan. Universal formulae and universal differential equations. *Annals of mathematics*, 124(2):273–291, 1986.

2 Daniel Bournez, Olivierand Graça and Amaury Pouly. On the Functions Generated by the General Purpose Analog Computer. *Information and Computation*, 257:34–57, 2017. doi:10.1016/j.ic.2017.09.015.

3 Olivier Bournez. Achilles and the Tortoise climbing up the hyper-arithmetical hierarchy. *Theoretical Computer Science*, 210(1):21–71, 1999.

4 Olivier Bournez and Manuel L. Campagnolo. *New Computational Paradigms. Changing Conceptions of What is Computable*, chapter A Survey on Continuous Time Computations, pages 383–423. Springer-Verlag, New York, 2008.

5 Olivier Bournez, Manuel L. Campagnolo, Daniel Graça, and Emmanuel S. Hainry. Polynomial differential equations compute all real computable functions on computable compact intervals. *Journal of Complexity*, 23(3):317–335, 2007.

6 Olivier Bournez, Manuel L. Campagnolo, Daniel S. Graça, and Emmanuel Hainry. Polynomial differential equations compute all real computable functions on computable compact intervals. *jcomp*, 23(3):317–335, 2007.

7 Olivier Bournez and Arnaud Durand. Recursion schemes, discrete differential equations and characterization of polynomial time computation. In *MFCS 2019: 44th International Symposium on Mathematical Foundations of Computer Science*, 2019.

8 Olivier Bournez, Daniel S. Graça, and Amaury Pouly. Polynomial Time corresponds to Solutions of Polynomial Ordinary Differential Equations of Polynomial Length. *Journal of the ACM*, 64(6):38:1–38:76, 2017. doi:10.1145/3127496.

9 Olivier Bournez, Daniel S. Graça, and Amaury Pouly. Polynomial Time corresponds to Solutions of Polynomial Ordinary Differential Equations of Polynomial Length. The General Purpose Analog Computer and Computable Analysis are two efficiently equivalent models of computations. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 109:1–109:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

10 Olivier Bournez and Amaury Pouly. A universal ordinary differential equation. In *International Colloquium on Automata Language Programming, ICALP'2017*, 2017.

11 Olivier Bournez and Amaury Pouly. A universal ordinary differential equation. *To appear in Logical Methods in Computer Science*, abs/1702.08328, 2020. arXiv:1702.08328.

12 H. J. Buisman, H. M. M. ten Eikelder, P. A. J. Hilbers, and A. M. L. Liekens. Computing algebraic functions with biochemical reaction networks. *Artificial Life*, 15(1):5–19, 2009. doi:10.1162/artl.2009.15.1.15101.

13 Vannevar Bush. The differential analyzer. A new machine for solving differential equations. *J. Franklin Inst.*, 212:447–488, 1931.

14 C. S. Calude and B. Pavlov. Coins, quantum measurements, and Turing's barrier. *Quantum Information Processing*, 1(1-2):107–127, 2002.

15 David C. Carothers, G. Edgar Parker, James S. Sochacki, and Paul G. Warne. Some properties of solutions to polynomial systems of differential equations. *Electronic Journal of Differential Equations*, 2005(40):1–17, 2005.

16 Ho-Lin Chen, David Doty, and David Soloveichik. Rate-independent computation in continuous chemical reaction networks. In *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science*, ITCS '14, pages 313–326, New York, NY, USA, 2014. ACM. doi:10.1145/2554797.2554827.

17 Matthew Cook, David Soloveichik, Erik Winfree, and Jehoshua Bruck. Programmability of chemical reaction networks. In *Algorithmic Bioprocesses*, pages 543–584. Springer, 2009.

18 B. Jack Copeland. Even Turing machines can compute uncomputable functions. In C.S. Calude, J. Casti, and M.J. Dinneen, editors, *Unconventional Models of Computations*. Springer-Verlag, 1998.

19 B. Jack Copeland. Accelerating Turing machines. *Minds and Machines*, 12:281–301, 2002.

20 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms (third edition)*. MIT press, 2009.

21 E. B. Davies. Building infinite machines. *The British Journal for the Philosophy of Science*, 52:671–682, 2001.

22 François Fages, Steven Gay, and Sylvain Soliman. Inferring reaction systems from ordinary differential equations. *Theoretical Computer Science*, 599:64–78, 2015. doi:10.1016/j.tcs.2014.07.032.

23 Francois Fages, Guillaume Le Guludec, Olivier Bournez, and Amaury Pouly. Strong turing completeness of continuous chemical reaction networks and compilation of mixed analog-digital programs. In *Computational Methods in Systems Biology-CMSB 2017*, 2017.

24 Andrei Gabrielov and Nicolai Vorobjov. Complexity of computations with pfaffian and noetherian functions. *Normal forms, bifurcations and finiteness problems in differential equations*, pages 211–250, 2004.

25 Aleksandr Gelfond. *Calculus of finite differences*. Hindustan Publ. Corp., 1971.

26 David Gleich. Finite calculus: A tutorial for solving nasty sums. *Stanford University*, 2005.

27 Daniel S. Graça. Some recent developments on Shannon's General Purpose Analog Computer. *Mathematical Logic Quaterly*, 50(4-5):473–485, 2004.

28 Daniel S. Graça, Manuel L. Campagnolo, and J. Buescu. Computability with polynomial differential equations. *Advances in Applied Mathematics*, 2007. To appear.

29 Daniel S. Graça and José Félix Costa. Analog computers and recursive functions over the reals. *Journal of Complexity*, 19(5):644–664, 2003.

30 Ronald L Graham, Donald E Knuth, Oren Patashnik, and Stanley Liu. Concrete mathematics: a foundation for computer science. *Computers in Physics*, 3(5):106–107, 1989.

31 Otto Hölder. Über die eigenschaft der gamma funktion keiner algebraische differentialgleichung zu genügen. *Math. Ann.*, 28:1–13, 1887.

32 FA Izadi, N Aliev, and G Bagirov. *Discrete Calculus by Analogy*. Bentham Science Publishers, 2009.

33 Gustavo Lau. Discrete calculus. URL: http://www.acm.ciens.ucv.ve/main/entrenamiento/material/DiscreteCalculus.pdf.

34 Lenoard. Lipshitz and Lee A. Rubel. A differentially algebraic replacement theorem, and analog computability. *Proceedings of the American Mathematical Society*, 99(2):367–372, 1987.

35 Cristopher Moore. Recursion theory on the reals and continuous-time computation. *Theoretical Computer Science*, 162(1):23–44, 1996.

36 André Platzer and Yong Kiam Tan. Differential equation axiomatization: The impressive power of differential ghosts. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 819–828. ACM, 2018.

37 Marian B. Pour-El. Abstract computability and its relation to the general purpose analog computer (some connections between logic, differential equations and analog computers). *Transactions of the American Mathematical Society*, 199:1–28, 1974.

38 L. A. Rubel. A survey of transcendentally transcendental functions. *American Mathematical Monthly*, 96(9):777–788, 1989.

39 Lee A. Rubel. A universal differential equation. *Bulletin of the American Mathematical Society*, 4(3):345–349, 1981.

40 Keijo Ruohonen. Undecidability of event detection for ODEs. *Journal of Information Processing and Cybernetics*, 29:101–113, 1993.

41 Keijo Ruohonen. Event detection for ODEs and nonrecursive hierarchies. In *Proceedings of the Colloquium in Honor of Arto Salomaa. Results and Trends in Theoretical Computer Science (Graz, Austria, June 10-11, 1994)*, volume 812 of *Lecture Notes in Computer Science*, pages 358–371. Springer-Verlag, Berlin, 1994. URL: `http://springerlink.metapress.com/openurl.asp?genre=article&issn=0302-9743&volume=812&spage=358`.

42 Claude E. Shannon. Mathematical theory of the differential analyser. *Journal of Mathematics and Physics MIT*, 20:337–354, 1941.

43 Klaus Weihrauch. *Computable Analysis: an Introduction*. Springer, 2000.

Graphical Models: Queries, Complexity, Algorithms

Martin C. Cooper
Université Fédérale de Toulouse, ANITI, IRIT, Toulouse, France
cooper@irit.fr

Simon de Givry
Université Fédérale de Toulouse, ANITI, INRAE, UR 875, Toulouse, France
`https://miat.inrae.fr/degivry/`
Simon.de-Givry@inrae.fr

Thomas Schiex[1]
Université Fédérale de Toulouse, ANITI, INRAE, UR 875, Toulouse, France
`http://www.inrae.fr/mia/T/schiex`
Thomas.Schiex@inrae.fr

Abstract

Graphical models (GMs) define a family of mathematical models aimed at the concise description of multivariate functions using decomposability. We restrict ourselves to functions of discrete variables but try to cover a variety of models that are not always considered as "Graphical Models", ranging from functions with Boolean variables and Boolean co-domain (used in automated reasoning) to functions over finite domain variables and integer or real co-domains (usual in machine learning and statistics). We use a simple algebraic semi-ring based framework for generality, define associated queries, relationships between graphical models, complexity results, and families of algorithms, with their associated guarantees.

2012 ACM Subject Classification Theory of computation → Discrete optimization; Mathematics of computing → Graph algorithms

Keywords and phrases Computational complexity, tree decomposition, graphical models, submodularity, message passing, local consistency, artificial intelligence, valued constraints, optimization

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.4

Category Tutorial

Supplement Material `http://github.com/toulbar2/toulbar2`

Funding The authors have received funding from the French "Investing for the Future – PIA3" program under the Grant agreement ANR-19-PI3A-000, in the context of the ANITI project.
Martin C. Cooper: Funded by ANR-18-CE40-0011
Simon de Givry: Funded by ANR-16-CE40-0028.
Thomas Schiex: Funded by ANR-16-CE40-0028.

1 Introduction

Graphical models are descriptions of decomposable multivariate functions. A variety of famous frameworks in Computer Science, Logic, Constraint Satisfaction/Programming, Machine Learning, Statistical Physics and Artificial Intelligence can be considered as specific sub-classes of graphical models. In this paper, we restrict ourselves to models describing functions of variables having each a finite (therefore discrete) domain with a totally ordered co-domain, that may be finite or not. This excludes, for example, "Gaussian Graphical Models" which are used in statistics, describing continuous probability densities.

[1] Corresponding author

A graphical model over a set of variables is defined by a finite set of small functions which are combined together using a binary associative and commutative operator, defining a joint multivariate function. The combined functions may be small because they involve few variables or because they are described using a restricted language.

As an example, propositional logic (PL) is organized around the idea of describing logical properties as functions over Boolean variables. From the values of these variables, one should be able to determine if a property is true or not. A usual way to achieve this is to use clauses (disjunction of variables or their negation) as "small functions" and combine them with conjunction, defining the Conjunctive Normal Form. The resulting joint function defines the truth value of the formula. Various queries on such graphical models have been considered: existence of a variable assignment that optimizes the joint function, the SAT/PL problem, model counting (the #P-complete #-SAT/PL problem), etc.

If we instead use tensors to describe Boolean functions over sets of variables of bounded cardinality and combine these functions with conjunction again, we obtain a constraint network (CN) and the existence of an assignment that optimizes the joint function is the Constraint Satisfaction Problem (CSP/CN).

In the extreme, we may use tensors to describe small real (in $\mathbb{R}$) functions combined using addition or multiplication, enabling the description of discrete probability distributions, as done with Markov Random Fields (MRFs) and Bayesian Networks (BNs) [74, 16]. The associated optimization problem is the Maximum A Posteriori (MAP/MRF) problem and weighted counting allows to compute the partition function [74] (or normalizing constant), another #P-complete problem with important applications in statistical physics and machine learning. The terminology of "graphical models" comes from the stochastic facet of GMs. Stochastic graphical models are of specific interest because they can be learned (or estimated) from data. Assuming that an i.i.d. sample of an unknown probably distribution is available, one may look for a graphical model that gives maximum probability to the sample. This maximum-likelihood approach has attractive asymptotic properties [74]. Efficient approximate estimation algorithms based on convex optimization exist [95]. Because of the unavoidable sampling noise, exact optimization algorithms are usually not sought. In this paper, we assume that the graphical models are known, either because they describe known rules or functions or have been previously estimated from data.

In this stochastic case, algorithms that provide some form of guarantee remain desirable, especially if the considered graphical model combines logical (deterministic) information, describing known/required properties, with probabilistic information learned from data. Indeed, logical information cannot be approximated without a complete loss of semantics.

In the rest of this paper, we first give an algebraic definition of what a graphical model is and consider the most usual queries. We then introduce the two main families of algorithms that can be used to exactly solve such queries: tree search and non-serial dynamic programming. Restricted to small sub-problems, we show how dynamic programming can provide fast approximate algorithms, called message-passing or local consistency enforcing algorithms, including associated polynomial classes. We then consider some empirically useful hybrid algorithms, with associated solvers, based on these approaches and recent applications in structural biology.

2 Notations, Definition

Variables are denoted as capital variables $X, Y, Z, \ldots$ that may be optionally indexed (X_i or just i). Variables can be assigned values from their domain. The domain of a variable X is denoted as D_X or D_i for variable X_i. Actual values are represented as $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{g}, \mathtt{r}, \mathtt{t}, \mathtt{1} \ldots$

and an unknown value denoted as $u, v, w, x, y, z \ldots$. Sequences or sets of objects are denoted in bold. A sequence of variables is denoted as $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}, \ldots$ A sequence of values is denoted as $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \ldots$. The domain of a sequence of variables $\boldsymbol{X}$ is denoted as $D_{\boldsymbol{X}}$, the Cartesian product of the domains of all the variables in $\boldsymbol{X}$. An assignment $\boldsymbol{u_X}$ is an element of $D_{\boldsymbol{X}}$ which defines an assignment for all the variables in $\boldsymbol{X}$. Finally, we denote by $\boldsymbol{u_X}[\boldsymbol{Y}]$ (or $\boldsymbol{u_Y}$) the *projection* of $\boldsymbol{u_X}$ on $\boldsymbol{Y} \subseteq \boldsymbol{X}$: the sequence of values that the variables in $\boldsymbol{Y}$ takes in $\boldsymbol{u_X}$.

▶ **Definition 1** (Graphical Model). *A Graphical Model $\mathcal{M} = \langle \boldsymbol{V}, \Phi \rangle$ with co-domain B and a combination operator $\oplus$ is defined by:*

- *a sequence of n variables $\boldsymbol{V}$, each with an associated finite domain of size less then d.*
- *a set of e functions (or factors) Φ. Each function $\varphi_{\boldsymbol{S}} \in \Phi$ is a function from $D_{\boldsymbol{S}} \to B$. $\boldsymbol{S}$ is called the scope of the function and $|\boldsymbol{S}|$ its arity.*

$\mathcal{M}$ defines a joint function:

$$\Phi_{\mathcal{M}}(\boldsymbol{v}) = \bigoplus_{\varphi_{\boldsymbol{S}} \in \Phi} \varphi_{\boldsymbol{S}}(\boldsymbol{v}[\boldsymbol{S}])$$

In this definition we assume that the co-domain B is totally ordered (by $\prec$) with a minimum and maximum element. Arbitrary elements of B will be denoted by Greek letters $(\alpha, \beta, \ldots)$. The combination operator $\oplus$ is required to be associative, commutative, monotonic and has an identity element $\mathbf{0} \in B$, the minimum element of B. The maximum element $\top$ of B is required to be absorbing ($\alpha \oplus \top = \top$). This set of axioms defines what is called a valuation structure [104], introduced in AI to represent graded beliefs or costs in Constraint Networks. It is closely related to triangular co-norms, often using $B = [0, 1]$ [72]. It is also known as a tomonoid in algebra and includes tropical algebra [56]. Commutativity and associativity make the combination insensitive to the order of application of $\oplus$. Monotonicity captures the fact that adding more information can only strengthen it. Finally, the specific roles of the maximum and minimum elements of B are here to express deterministic information: $\mathbf{0}$ represents the fact that a function has reached an absolute minimum and this has no effect on the value of the joint function value Φ. Instead, any combined function taking value $\top$ will lead to a joint function that is also equal to $\top$. For simplicity, we assume that elements of B can be represented in constant space and that $a \oplus b$ can be computed in constant time. An important additional property of $\oplus$ is often considered: idempotency ($\alpha \oplus \alpha = \alpha$). As section 6.1 will show, it has strong algorithmic implications. Finally, such valuation structures are said to be *fair* if, for any elements of B such that $\beta \preccurlyeq \alpha$, there exists a maximum γ such that $\beta \oplus \gamma = \alpha$ (such a γ may not exist in infinite structures). This element γ is denoted $\alpha \ominus \beta$ and defines a pseudo-inverse operator $\ominus$ (we have $\beta \oplus (\alpha \ominus \beta) = \alpha$).

Table 1 Some valuation structures. Idemp. indicates the idempotency of $\oplus$. See [33] for details.

Structure (GM)	B	$a \oplus b$	$\prec$	$\mathbf{0}$	$\top$	Idemp.	$a \ominus b$
Boolean (PL,CN)	$\{t, f\}$	$a \wedge b$	$t < f$	t	f	yes	a
Additive (GAI)	$\bar{\mathbb{N}}$	$a + b$	$<$	0	$+\infty$	no	$a - b$
Weighted (CFN)	$\{0,1,\ldots,k\}$	$\min(k, a+b)$	$<$	0	k	no	$(a=k\ ?\ k\ :\ a-b)$
Probabilistic (MRF,BN)	$[0, 1]$	$a \times b$	$>$	1	0	no	a/b
Possibilistic (PCN)	$[0, 1]$	$\max(a, b)$	$<$	0	1	yes	$\max(a, b)$
Fuzzy (FCN)	$[0, 1]$	$\min(a, b)$	$>$	1	0	yes	$\min(a, b)$

Valuation structures have been analyzed in detail [33, 31]. We know that any fair and countable valuation structure can be viewed as a stack of additive/weighted valuation structures, which interact with each other as an idempotent structure (thus using $\oplus = \max$).

The case $\oplus = \max$ being very close to the case of classical Constraint Networks [88, page 293], most of the work has since then focused on the weighted and probabilistic cases.

The mathematical definition of graphical models above needs to be further refined by defining how the functions $\varphi_{\boldsymbol{S}} \in \Phi$ are represented. Thanks to the assumption of finite domains, a universal representation of functions can be obtained using tensors (or multi-dimensional tables) that map any sequence of values $\boldsymbol{u}_{\boldsymbol{S}} \in D_{\boldsymbol{S}}$ to an element of B. This representation requires $O(d^{|\boldsymbol{S}|})$ space where d is the maximum domain size. When useful, alternative specialized representations may be used, with possibly different complexities for various elementary operations and queries.

Altogether, this definition of graphical models covers a variety of well-studied frameworks, including constraint networks [101] (tensors + Boolean), propositional logic [14] (Boolean domains, clauses + Boolean), generalized additive independence models [7] (tensors + additive), weighted propositional logic (Boolean domains, clauses + additive), Markov Random Fields [71, 74] (tensors + Probabilistic), Bayesian networks [74] (MRFs with normalized tensors describing conditional probabilities, organized according to a directed acyclic graph), Possibilistic and Fuzzy Constraint Networks [48] (tensors + Possibilistic/Fuzzy). There are various related models, such as *Ceteris Paribus* networks (or CP-nets [47]) that could be covered using a weaker set of axioms, possibly too weak to prove interesting results.

We are biased in the paper towards the family of Cost Function Networks (CFN) which are GMs using tensors (extended with so-called "global functions", see e.g. [81, 3]) and the Weighted structure. This combines the ability of combining additive finite weights with a finite upper bound cost that appears naturally in many situations.

▶ **Definition 2.** *Two graphical models $\mathcal{M} = \langle \boldsymbol{V}, \Phi \rangle$ and $\mathcal{M}' = \langle \boldsymbol{V}, \Phi' \rangle$, with the same variables and valuation structure are equivalent iff they define the same joint function: $\forall \boldsymbol{v} \in D_{\boldsymbol{V}}, \Phi_{\mathcal{M}}(\boldsymbol{v}) = \Phi_{\mathcal{M}'}(\boldsymbol{v})$.*

CFNs have a tight relationship with stochastic graphical models. Since $\log(a \times b) = \log(a) + \log(b)$, a $-\log$ transform applied to elements of B followed by suitable scaling, shifting and rounding can transform any stochastic graphical model into a Cost Function Network defining a joint function that represents a controlled approximation of the $-\log()$ of the joint function of the stochastic model. The function log being monotonic, MAP/MRF and WCSP/CFN are essentially the same problems.

▶ **Definition 3.** *Given two graphical models $\mathcal{M} = \langle \boldsymbol{V}, \Phi \rangle$ and $\mathcal{M}' = \langle \boldsymbol{V}, \Phi' \rangle$, with the same variables and valuation structure, $\mathcal{M}$ is a relaxation of $\mathcal{M}'$ iff $\forall \boldsymbol{v} \in D_{\boldsymbol{V}}, \Phi_{\mathcal{M}}(\boldsymbol{v}) \preccurlyeq \Phi_{\mathcal{M}'}(\boldsymbol{v})$.*

In a graphical model, the set of variables $\boldsymbol{V}$ and the set of scopes $\boldsymbol{S}$ of the different functions $\varphi_{\boldsymbol{S}} \in \Phi$ define a hyper-graph whose vertices are the variables of $\boldsymbol{V}$ and the hyper-edges the different scopes. In the case in which the maximum arity is limited to 2, this hyper-graph is a graph, called the constraint graph or problem graph (hence the name graphical model). When larger arities are present, one can use the 2-section of the hypergraph as an approximation of it, often called the primal or moral graph of the GM. The bipartite incidence graph of this hypergraph has elements of $\boldsymbol{V}$ and Φ as vertices, an edge connecting $X \in \boldsymbol{V}$ to $\varphi_{\boldsymbol{S}}$ iff $X \in \boldsymbol{S}$ and is often called the factor graph [1] of the graphical model. This hyper-graph or its incidence graph only captures the overall structure of the model but neither the domains nor the functions. Therefore, a more fine-grained representation, known as the micro-structure graph, is often used to represent (pairwise) graphical models, vertices representing values and weighted edges the value of functions on pairs of values.

3 Queries

A query over a graphical model asks to compute simple statistics over the function such as its minimum or average value. Such queries already cover a wide range of practical usages. Assuming that `true` $\prec$ `false`, the SAT/PL [14] and CSP/CN [101] problems are equivalent to finding the minimum of the joint function, which is also the case for (Weighted) Max-SAT, WCSP/CFN, MAP/MRF (or MPE/BN aka Maximum Probability Explanation in Bayesian Networks). In these optimization problems, we want to compute $\min_{\boldsymbol{v}\in D_{\boldsymbol{V}}} \Phi(\boldsymbol{v})$. Alternatively, for numerical B, counting problems require to compute $\sum_{\boldsymbol{v}\in D_{\boldsymbol{V}}} \Phi(\boldsymbol{v})$. A relatively general situation can be captured by introducing a so-called "elimination" operator $\otimes$, assumed to be associative, commutative and such that $\oplus$ distributes over $\otimes$: $\alpha \oplus (\beta \otimes \gamma) = (\alpha \oplus \beta) \otimes (\alpha \oplus \gamma)$. The two operations $(\otimes, \oplus)$ and their associated properties have been studied, with some variations, under a variety of names, often in relation with non-serial dynamic programming [110, 17, 1, 73, 72, 56]. The query to answer becomes $\bigotimes_{\boldsymbol{v}\in D_{\boldsymbol{V}}} \oplus_{\varphi_{\boldsymbol{S}}\in\Phi} \varphi_{\boldsymbol{S}}(\boldsymbol{v}[\boldsymbol{S}])$.

A powerful toolbox on graphical models can be built from three simple operations: assignment (or conditioning), combination and elimination.

Given a function $\varphi_{\boldsymbol{S}}$, it is possible to assign a variable $X_i \in \boldsymbol{S}$ with a value $a \in D_i$. We obtain a function on $\boldsymbol{T} = \boldsymbol{S} - \{X_i\}$ defined by $\varphi_{\boldsymbol{T}}(\boldsymbol{v}) = \varphi_{\boldsymbol{S}}(\boldsymbol{v} \cup \{X_i = a\})$. If we assign all the variables of a function, we obtain a function with an empty scope, denoted $\varphi_\varnothing$. This operation has negligible complexity (we directly access a part of the original cost function).

The second operation is the equivalent of the relational join operation in databases: it combines two functions $\varphi_{\boldsymbol{S}}$ and $\varphi_{\boldsymbol{S}'}$ into a single function, which is equivalent to their combination by $\oplus$. The resulting function has the scope $\boldsymbol{S} \cup \boldsymbol{S}'$ and is defined by $(\varphi_{\boldsymbol{S}} \oplus \varphi_{\boldsymbol{S}'})(\boldsymbol{v}) = \varphi_{\boldsymbol{S}}(\boldsymbol{v}[\boldsymbol{S}]) \oplus \varphi_{\boldsymbol{S}'}(\boldsymbol{v}[\boldsymbol{S}'])$. The calculation of the combination of the two functions is exponential in time and space $(O(d^{|S\cup S'|}))$. Observe that we can replace a set of functions by their combination without changing the joint function defined by the graphical model (preserving equivalence).

Finally, the elimination operation consists in summarising the information from a function $\varphi_{\boldsymbol{S}}$ on a subset of variables $\boldsymbol{T} \subset \boldsymbol{S}$. The elimination of the variables of $\boldsymbol{S} - \boldsymbol{T}$ in $\varphi_{\boldsymbol{S}}$ leads to its so-called projection (or marginal) onto $\boldsymbol{T}$, the function $\varphi_{\boldsymbol{T}}(\boldsymbol{u}) = \bigotimes_{\boldsymbol{v}\in D_{\boldsymbol{S}-\boldsymbol{T}}} \varphi_{\boldsymbol{S}}(\boldsymbol{u} \cup \boldsymbol{v})$. Since elimination requires enumerating the whole domain $D_{\boldsymbol{S}}$ of $\varphi_{\boldsymbol{S}}$, it has time complexity $O(d^{|\boldsymbol{S}|})$. The resulting function requires $O(d^{|\boldsymbol{T}|})$ space. If $\boldsymbol{S} - \boldsymbol{T}$ is a singleton $\{X_i\}$, we will denote by $\varphi_{\boldsymbol{S}}[-X_i] = \varphi_{\boldsymbol{S}}[\boldsymbol{S} - \{X_i\}]$ the result of the elimination of the single variable X_i.

These complexities are given for functions represented as tensors. They may change if other languages such as clauses, analytic representation or compact data structures such as weighted automata or decision diagrams [38, 51] are used to represent functions.

We close this section by expressing simple graph problems as WCSP/CFNs.

Given an undirected graph $\mathcal{G} = \langle \boldsymbol{V}, \boldsymbol{E} \rangle$ with vertex set $\boldsymbol{V}$ and edge set $\boldsymbol{E}$, we can define the following WCSP/CFN $\mathcal{M} = \langle \boldsymbol{V}, \Phi \rangle$ problems, having one variable per vertex:

- s–t Min-Cut: $\forall i \in \boldsymbol{V}\backslash\{s,t\}, D_i = \{a,b\}$, $D_s = \{a\}$, $D_t = \{b\}$, and $\Phi = \{\varphi_{ij} \mid (i,j) \in \boldsymbol{E}\}$, with $\varphi_{ij}(a,a) = \varphi_{ij}(b,b) = 0$ and $\varphi_{ij}(a,b) = \varphi_{ij}(b,a) = 1$.
- Max-Cut: $\forall i \in \boldsymbol{V}, D_i = \{a,b\}$, $\Phi = \{\varphi_{ij} \mid (i,j) \in \boldsymbol{E}\}$, with $\varphi_{ij}(a,a) = \varphi_{ij}(b,b) = 1$ and $\varphi_{ij}(a,b) = \varphi_{ij}(b,a) = 0$.
- Vertex Cover: $\forall i \in \boldsymbol{V}, D_i = \{a,b\}$, $\Phi = \{\varphi_i \mid i \in \boldsymbol{V}\} \bigcup \{\varphi_{ij} \mid (i,j) \in \boldsymbol{E}\}$, with $\varphi_i(a) = 0$, $\varphi_i(b) = 1$, $\varphi_{ij}(a,b) = \varphi_{ij}(b,a) = \varphi_{ij}(b,b) = 0$ and $\varphi_{ij}(a,a) = \top$.
- Max-Clique: $\forall i \in \boldsymbol{V}, D_i = \{a,b\}$, $\Phi = \{\varphi_i \mid i \in \boldsymbol{V}\} \bigcup \{\varphi_{ij} \mid (i,j) \notin \boldsymbol{E}\}$, with $\varphi_i(a) = 0$, $\varphi_i(b) = 1$, $\varphi_{ij}(a,b) = \varphi_{ij}(b,a) = \varphi_{ij}(b,b) = 0$ and $\varphi_{ij}(a,a) = \top$.

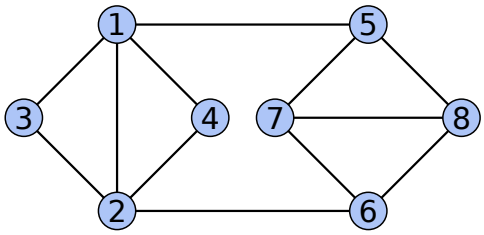


Figure 1 Simple graph example (from Wikipedia DSATUR). Optimum value and corresponding solution for Min-Cut ($s=1, t=8$): $\Phi_{\mathcal{M}}(aaaabbbb)=2$, Max-Cut: $\Phi_{\mathcal{M}}(aabbbbaa)=2$ (i.e., a maximum cut involving $12-2=10$ edges), Vertex Cover: $\Phi_{\mathcal{M}}(bbaaaabb)=4$, Max-Clique: $\Phi_{\mathcal{M}}(aaabbbbb)=5$ (i.e., a maximum clique of size $8-5=3$), 3-Coloring: $\Phi_{\mathcal{M}}(abccccab)=0$, and Min-Sum 3-Coloring: $\Phi_{\mathcal{M}}(bcaaaabc)=14$.

- Graph Coloring with 3 colors: $\forall i \in \boldsymbol{V}, D_i = \{a, b, c\}$, $\Phi = \{\varphi_{ij} \mid (i,j) \in \boldsymbol{E}\}$, with $\forall u, v$, $\varphi_{ij}(u,v) = 0$ if $u \neq v$ and $\varphi_{ij}(u,v) = \top$ otherwise.
- Min-Sum Coloring with 3 colors: $\forall i \in \boldsymbol{V}, D_i = \{a, b, c\}$, $\Phi = \{\varphi_i \mid i \in \boldsymbol{V}\} \bigcup \{\varphi_{ij} \mid (i,j) \in \boldsymbol{E}\}$, with $\varphi_i(a) = 1$, $\varphi_i(b) = 2$, $\varphi_i(c) = 3$, and $\forall u, v$, $\varphi_{ij}(u,v) = 0$ if $u \neq v$ and $\varphi_{ij}(u,v) = \top$ otherwise.

An example of optimal solution for each problem is given in Fig.1.

4 Tree search

One of the most basic algorithms to answer the query $\bigotimes_{\boldsymbol{v} \in D_{\boldsymbol{V}}} \oplus_{\varphi_{\boldsymbol{S}} \in \Phi}$ relies on conditioning and can be described as brute force tree-search. It relies on the fact that if all variables in a graphical model are assigned (all variables have domain size 1), the answer can be obtained by just computing Φ on the assignment. If a given model has a variable $X_i \in \boldsymbol{V}$ with a larger domain, the query can be reduced to solving two queries on simpler models, one where X_i is assigned to some value $a \in D_i$ and the other where a is removed from D_i. The result of these queries is then combined using $\otimes$. This can be understood as the exploration of a binary tree whose root is the original graphical model and where leaves are fully assigned models on which the query can be answered. This tree can be explored using various strategies (Depth-First, Breadth-First, Iterative [94],...), DFS has $O(nd)$ space and $\theta(\exp(n))$ time.

In the case of optimization ($\otimes = \min$) on a GM $\mathcal{M}$ it is possible to exploit any available lower bound on the values of the leaves below a given node in the tree for pruning. If this lower bound is larger than or equal to the value of the joint function on a best known leaf, then the sub-tree can be pruned. If $\mathcal{M}$ contains a constant function $\varphi_{\varnothing}$ with empty scope, then the value of this function provides such a lower bound, thanks to monotonicity and non-negativity. With pruning, the order in which this tree is explored (the choice of the branching variable X_i and value a) becomes crucial for empirical efficiency.

5 Non-serial Dynamic Programming

An alternative approach for answering the query above consists in using non-serial dynamic programming (DP) [11], also called Variable-Elimination (VE), bucket elimination [44], among other names [74, 110, 1]. The process relies on the distributivity of $\otimes$ over $\oplus$ and has also been described as the Generalized Distributive Law[1]. To our knowledge, its axiomatic requirements in a general situation were first studied in [110]. The fundamental idea itself is reminiscent of famous elimination algorithms (e.g. Gaussian elimination) and was already

described in 1972 under the name of Non-Serial Dynamic Programming [11], which we use. This book also defines a graph parameter (DIMENSION), and is the same as tree-width [19].

In its simplest variable elimination variant, non serial DP works variable per variable, replacing a variable X and all functions that involve it by a function on the neighbors of X. Following the stochastic GMs terminology, this new function will be called a "message". Given a graphical model $\mathcal{M} = \langle \boldsymbol{V}, \Phi \rangle$, a variable $X \in \boldsymbol{V}$, let Φ^X be the set of functions $\varphi_{\boldsymbol{S}} \in \Phi$ such that $X \in \boldsymbol{S}$ and T the neighbors of X in the graph of the problem, we define the message $m_{\boldsymbol{T}}^{\Phi^X}$ from Φ^X to $\boldsymbol{T}$ as:

$$m_{\boldsymbol{T}}^{\Phi^X} = (\bigoplus_{\varphi_{\boldsymbol{S}} \in \Phi^X} \varphi_{\boldsymbol{S}})[-X] \tag{1}$$

By distributivity, it is now possible to rewrite our query as:

$$\bigotimes_{\boldsymbol{v} \in D_{\boldsymbol{V}}} \oplus_{\varphi_{\boldsymbol{S}} \in \Phi} \varphi_{\boldsymbol{S}}(\boldsymbol{v}[\boldsymbol{S}]) = \bigotimes_{\boldsymbol{v} \in D_{\boldsymbol{V}-\{X\}}} \oplus_{\varphi_{\boldsymbol{S}} \in \Phi - \Phi^X \cup \{m_{\boldsymbol{T}}^{\Phi^X}\}} \varphi_{\boldsymbol{S}}(\boldsymbol{v}[\boldsymbol{S}])$$

This equality shows that our initial query can be answered by solving a similar query on a graphical model that has one variable less and where all functions involving this variable have been replaced by a new function (or message) $m_{\boldsymbol{T}}^{\Phi^X}$. This operation can be applied to every variable successively until a final graphical model with no variable and a constant function $\varphi_\varnothing$ is obtained. The value of this function is the answer to the initial query.

When tensors are used to represent functions, the successive applications of combination and elimination show that computing $m_{\boldsymbol{T}}^X$ is $O(d^{|T+1|})$ time and space but space can be easily reduced to $O(d^{|T|})$ if elimination is performed incrementally. This can be different with other representations. In propositional logic, if ℓ is a literal over variable X and if Φ^X contains the two clauses $\varphi_{\boldsymbol{S}} = (\ell \vee \boldsymbol{L})$ and $\varphi_{\boldsymbol{S'}} = (\neg\ell \vee \boldsymbol{L'})$, where $\boldsymbol{L}$ and $\boldsymbol{L'}$ are clauses, then the message $m_{\boldsymbol{T}}^X$ obtained by combining the two clauses and eliminating X is the function represented by the clause $\boldsymbol{L} \vee \boldsymbol{L'}$: the resolution principle [99] performs efficient variable elimination [45].

It is well-known that the order in which the variables are eliminated can have a strong influence on the complexity of the whole process. Each elimination creates a new function whose scope is known. As each step is exponential in the number of variables in the neighbourhood of the eliminated variable X which come after X in the elimination order, one may prefer to simulate the process (play the elimination game) and get an estimate of the global complexity without actually performing the calculations. It is the step for which the number w of future neighbors is maximum which determines the complexity (spatial and temporal): this complexity is exponential in w. The parameter w is called the induced width of the graph of the GM but is better known as its tree-width for the given elimination order. Minimizing w is known to be NP-hard, but useful heuristics exist [18].

This algorithm defines the first non-trivial tractable class for the query above: graphical models with a bounded tree-width graph.

5.1 Message passing

The bounded tree-width class is specifically interesting for pairwise models $\mathcal{M}$ with an acyclic graph (tree-width one). In this case, the query above can be solved by rooting the tree in any variable $X \in \boldsymbol{V}$ and iteratively eliminating the leaves of this graph until only the root remains. In this case, messages $m_{\boldsymbol{T}}^X$ are such that $|\boldsymbol{T}| = 1$ and the message that corresponds

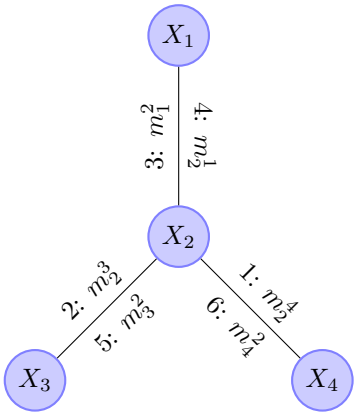


Figure 2 The graph of a GM with three pairwise functions. The marginal of Φ on X_2 can be computed by combining all three incoming messages: m_2^3, m_2^1, m_2^4.

to the elimination of variable X_i with parent X_j through edge φ_{ij} is just denoted m_j^i (a message from i to j). Denoting by $neigh(X)$ the set of neighbor variables of X in the tree, the message computed by applying Eq. 1 is:

$$m_j^i = \bigotimes_{X_i}(\varphi_i \oplus \varphi_{ij} \bigoplus_{X_o \in neigh(X_i), o \neq j} m_i^o)$$

In the end, one variable X remains and the joint function Φ it defines describes the so-called *marginal* of the joint function defined by $\mathcal{M}$. This is a function of X that answers the query for every possible assignment of X.

These marginal functions are of specific interest in counting queries over stochastic models as they provide marginal probabilities. Specific algorithms have therefore been developed to compute the marginals over all the variables of acyclic models. This requires only two steps (instead of one per variable). In this algorithm, messages are computed and kept aside and no function is removed from the graphical model. We therefore have an initially empty, growing list L of computed messages. If a variable has received messages from all its neighbors but one, it becomes capable of computing the message for this variable and adds it to the list of computed messages. The algorithm therefore starts by computing messages from leaves and ultimately computes two messages m_j^i and m_i^j for every edge φ_{ij}. For any variable X_i, the marginal of the joint function Φ on X_i is directly accessible as $\varphi_i \oplus_{o \in neigh(X_i)} m_i^o$. It is easy to check that the gathered messages for every vertex X are exactly those that would have been done if the tree had been rooted in X.

If the initial graph is cyclic, a tree decomposition of the graph can be identified and the algorithm above used to send messages $m_{\boldsymbol{S}}^{\boldsymbol{C}}$ by combining all functions in cluster $\boldsymbol{C}$ and projecting to separator $\boldsymbol{S}$. The resulting algorithms similarly computes marginals over clusters which can be further projected onto every variable inside [74]. This has two advantages over the pure variable elimination algorithm above: the space complexity is $O(\exp(s))$ where s is the size of the largest separator in the tree decomposition and all the marginals are accessible. In its "one-pass" variant, this algorithm is the "block-by-block" elimination introduced in 1972 by Bertelé and Brioschi [11]. Despite the improved space complexity (but unchanged time complexity), this algorithm is restricted to problems with very small tree-width (especially with large domain sizes).

6 Loopy Belief propagation and Local Consistency

The Loopy Belief Propagation heuristic can be used when the algorithm above cannot finish in reasonable time. In a pairwise model, the list L is initialized with all possible constant empty messages $m_i^j = m_j^i = \mathbf{0}$ for all $\varphi_{ij} \in \Phi$. Every message is then updated using the same equation as above: $m_j^i = \otimes_{X_i}(\varphi_i \oplus \varphi_{ij} \oplus_{X_o \in neigh(X_i), o \neq j} m_i^o)$. These updates can be done synchronously (new versions are all computed simultaneously and replace those of the previous iteration) or asynchronously. This is repeated until quiescence or after a finite number of iterations as the algorithm has no guarantee of termination in general. Loopy BP was introduced by Pearl [96] and later used for signal coding and encoding defining Turbo-decoding [10]. Despite its limited theoretical properties, it is one of the most used algorithms over stochastic graphical models, probably the most cited (implemented in every GSM phone). It has been the object of intense study [117]. It can be used to get heuristic answers for optimization or counting queries under the name of the max-sum, min-sum or sum-prod algorithm [1].

6.1 Local consistency and Filtering

For optimization ($\otimes = \min$), if $\oplus$ is either idempotent or fair, these algorithms can be transformed into well-behaved polytime reformulation techniques that provide incremental lower bounds, for Branch and Bound algorithms.

▶ **Theorem 4.** *Suppose that $\oplus$ is idempotent, and consider $\mathcal{M} = \langle \boldsymbol{V}, \Phi \rangle$ and $\mathcal{M}' = \langle \boldsymbol{V}, \Phi' \rangle$ a relaxation of $\mathcal{M}$. Then $\mathcal{M}'' = \langle \boldsymbol{V}, \Phi \cup \Phi' \rangle$ is equivalent to $\mathcal{M}$.*

Proof. From the axioms of an idempotent valuation structure, consider two valuations $\mathbf{0} \preccurlyeq \beta \preccurlyeq \alpha \in B$. Then $\alpha = \alpha \oplus \mathbf{0} \preccurlyeq \alpha \oplus \beta \preccurlyeq \alpha \oplus \alpha = \alpha$ by identity, monotonicity and idempotency successively. Then $\Phi_{\mathcal{M}''} = \Phi_{\mathcal{M}} \oplus \Phi_{\mathcal{M}'}$. The results follows from the fact that $\mathcal{M}'$ is a relaxation of $\mathcal{M}$. ◀

▶ **Theorem 5.** *If $\otimes = \min$, any message $m_{\boldsymbol{T}}^X$ computed by elimination is a relaxation of Φ^X and therefore of $\mathcal{M}$.*

This follows directly from the definition of a message and the fact that $\otimes = \min$. Together, these results show that any message (computed by Eq. 1) can be added to an idempotent graphical model, without changing its meaning. This result applies to the Boolean, Possibilistic and Fuzzy cases. In the Boolean case, we recover the trivial fact that a logical consequence (relaxation) of a formula can be added to it without changing its semantics (as achieved by the resolution principle [99] and the seminal Davis and Putnam algorithm [40]). It also immediately provides algorithms for Possibilistic and Fuzzy graphical models [49].

These observations are very useful in the context of message passing algorithms. If $\oplus$ is idempotent, it becomes possible to add the generated messages to the set of functions Φ of the graphical model with the guarantee that equivalence will be preserved. This idea underlies all the "(Boolean) constraint propagation" algorithms (e.g., unit propagation in PL and arc consistency filtering in CNs) that have been actively developed in the last decades. We now give a non-standard definition of Arc Consistency in binary idempotent graphical models to illustrate this:

▶ **Definition 6.** *A graphical model $\mathcal{M} = \langle \boldsymbol{V}, \Phi \rangle$ with idempotent $\oplus$ is arc-consistent iff every variable $X \in \boldsymbol{V}$ is arc consistent w.r.t. every function $\varphi_{\boldsymbol{S}}$ s.t. $X \in \boldsymbol{S}$. A variable X_i is arc-consistent w.r.t. a function φ_{ij} iff the message m_i^j is a relaxation of φ_i.*

This local consistency property can be satisfied (or not) by an idempotent graphical model. When it's not satisfied, there is an obvious brute force algorithm that can transform the non AC model into an equivalent AC model: it suffices to apply message passing and directly combine all messages m_i^j in $\mathcal{M}$ until the φ_i do not change. One can show that this algorithm must stop in polytime and that the process is confluent (there is only one fixpoint). This algorithm is not optimal however and several generations of AC algorithms for e.g. Constraint Networks have been proposed until the first time optimal algorithm (AC4 [89]), the first space and time optimal (AC6 [13]) and the simplest empirically most efficient time/space optimal variant (AC2001 aka AC3.1) came out [12, 118].

Arc consistency and Unit Propagation in SAT are fundamental in efficient CSP and SAT provers. Adding messages (new unary functions) to the problem makes these unary functions increasingly tight. Ultimately, it may be the case that some $\varphi_i \in \Phi$ becomes such that $\forall a \in D_i, \varphi_i(a) = \top$. Because equivalence is preserved, it is then known that the joint function of the original graphical model is always equal to $\top$ (it is inconsistent in logical terminology). Naturally, being a polytime process, arc consistency enforcing cannot provide such an inconsistency proof in all cases (assuming P $\neq$ NP). Stronger, more expensive, messages can be obtained using messages $m_{\boldsymbol{T}}^{\boldsymbol{C}}$ combining functions in $\boldsymbol{C}$ projected onto $\boldsymbol{T}$. In pairwise GMs, when $\boldsymbol{C}$ is defined by all functions whose scope is included in any set of cardinality less than i projected on any sub-scope of cardinality $i-1$, this is called i-consistency enforcing [30, 5], which solves idempotent GMs with tree-width less than i in polynomial time.

6.2 The non idempotent fair case

When optimizing, if $\oplus$ is not idempotent, it becomes impossible to simply add messages to the graphical model being processed and preserve equivalence. However, the pseudo-difference operator $\ominus$ of fair structures makes it possible to add the message to the model and compensate for this by subtracting the message from its source [103]. Such a generalization may lead to loss of guaranteed termination since information can both increase and decrease at different local levels. While usual local consistency enforcing takes a set of functions and adds the message m obtained by eliminating a subset of variables in the problem, what has been called an *Equivalence Preserving Transformation* (EPT) combines a set of functions into a new function φ, computes the associated message (or any relaxation of it) and replaces the original set of functions by m and $\varphi \ominus m$. When scopes are unchanged, this has been called reparameterizations in MRFs [115].

If the problem of computing an optimal *sequence* of EPTs (in the sense that applying these messages reformulate the GM into another one that has a maximum $\varphi_\varnothing$) is NP-hard for integer costs, finding an optimal *set* of messages using rational costs can be reduced to a linear programming instance of polynomial size [32], a reduction which was previously published in 1976 [109], in Russian. Schlesinger's team has produced a variety of results on GMs that have been reviewed in English [116]. The LP-based algorithm has the ability to exactly optimize additive or weighted [29] GMs with a tree-structure or with only submodular functions but solving the LP is empirically too slow to be of practical use in most cases. This LP has been shown to be universal, in the sense that any reasonable LP can be reduced to (the dual of) such an LP in linear time, with a constructive proof [97].

▶ **Definition 7.** *Function $\varphi_{\boldsymbol{S}}$ is submodular if* $\forall \boldsymbol{u}, \boldsymbol{v} \in D_{\boldsymbol{S}}, \varphi_{\boldsymbol{S}}(\min(\boldsymbol{u}, \boldsymbol{v})) + \varphi_{\boldsymbol{S}}(\max(\boldsymbol{u}, \boldsymbol{v})) \leq \varphi_{\boldsymbol{S}}(\boldsymbol{u}) + \varphi(\boldsymbol{v})$.

This assumes that domains are ordered, max and min being applied pointwise. If it exists, a witness order for submodularity can be easily found [108] defining the polynomial class of weighted GMs with permutated submodular functions [107].

The best known general algorithm for Boolean submodular function minimization is $O(en^3 \log^{O(1)} n)$ [82]. For CFNs, practical algorithms can be obtained by exploiting a connection with idempotent GMs (here CNs). Any weighted function $\varphi_{\boldsymbol{S}}$ can be transformed into a Boolean function $\text{Bool}(\varphi_S)$ which is `true` iff φ_S is non $\mathbf{0}$.

▶ **Definition 8** ([34, 29]). *Given a weighted GM (or CFN)* $\mathcal{M} = \langle \boldsymbol{V}, \Phi \rangle$, $\text{Bool}(\mathcal{M}) = \langle \boldsymbol{V}, \{\text{Bool}(\varphi_S) \mid |\boldsymbol{S}| > 0\} \rangle$ *(a constraint network).*

▶ **Definition 9** (Virtual Arc Consistency (VAC)[34]). *A weighted GM* $\mathcal{M} = \langle \boldsymbol{V}, \Phi \rangle$ *is Virtual Arc Consistent iff enforcing AC on* $\text{Bool}(\mathcal{M})$ *does not prove inconsistency.*

If enforcing AC on $\text{Bool}(\mathcal{M})$ yields an inconsistency proof, it is possible to extract a minimal sub-proof and transform it in a set of EPTs that will increase $\varphi_\varnothing$ when applied on $\mathcal{M}$. The repeated application of this principle leads to an $O(ed^2m/\varepsilon)$ time, $O(ed)$ space VAC enforcing algorithm (where ε is a required precision). It can be seen as a sophistication of max-flow algorithms using "augmenting proofs" [77, 116]. On Min-Cut problems, proofs can have the expected augmenting path structure. Related algorithms, using a Block Coordinate Descent approach to approximately solve the LP above have also been investigated in Image processing: the TRW-S algorithm [75] fixpoints satisfy the VAC property (called Weak Tree Agreement in MRFs) which can be exploited to detect strongly persistent assignments [57] (subsets of variables having the same value in all optimal solutions). On GMs with Boolean variables, the bound $\varphi_\varnothing$ is related to the max-flow roof-dual lower bound of Quadratic Pseudo-Boolean Optimization [20].

All these algorithms ultimately produce strengthened lower-bounds $\varphi_\varnothing$. In the context of Branch & Bound, looser approximations of the dual are often used because of their high incrementality. One of the most useful is existential directional arc consistency, with $O(ed^2 \max(nd, m))$, $O(nd)$ space complexity on pairwise models [80], that solves tree-structured problems.

7 Hybrid algorithms

In this section, we rapidly review a subset of the large number of hybrid algorithms that have been designed to rigorously answer optimization queries over graphical models. The arena of existing algorithms is currently clearly dominated by a family of algorithms that combines tree search with local consistency enforcing. Local consistency enforcing strengthens the obvious lower bound defined by $\varphi_\varnothing$. It reformulates the graphical model associated with each node of the search-tree incrementally. It provides improved information with tightened unary φ_i. This information can be used in cheap variable and value ordering heuristics, to decide which variable to explore first, along which value. These heuristics are crucial for the empirical efficiency of the algorithms (as evaluated on large sets of benchmarks, which are increasingly available in this data era). They have have become adaptive [90, 21]: they are parameterized and these parameters change during tree search, trying to identify and favor the selection of variables that belong to the regions of the graphical model that contain strong information, variables which once assigned would lead to rapid backtracking.

These algorithms can be (empirically) enhanced by hybrid branching strategies mixing depth and best-first, restarts (preserving information not invalidated by restart), stronger message passing at the root node, dominance analysis (showing a value `a` can always be replaced by value `b` without degrading Φ allows us to remove `a`).

In the idempotent $\oplus$ case, conflict analysis [15] is a powerful technique that allows to produce a new informative relaxation (or logical consequence) of the model when a conflict occurs during tree search (a backtrack must occur). Initially developed [42] and improved [106] for Constraint Networks, this has been adapted to SAT and is now a fundamental ingredient of the modern CDCL (Conflict Directed Clause Learning) SAT provers that obsoleted the 90's generation of advanced DPLL (Davis, Putnam, Logemann, Loveland) provers [39].

At this point, one may note that for optimization at least, the empirically most efficient algorithms are algorithms that have $O(exp(n))$ worst-case complexity while algorithms with better worst-case complexity (based on non-serial dynamic programming) are restricted to a small subset of problems having (small) bounded tree-width (depending on d). A series of proposals tried to define algorithms that combined the empirical efficiency of enhanced tree-search algorithms with tree-width based bounds. To the best of our knowledge, the first algorithm along this line is the Pseudo-Tree search algorithm [53], for solving the CSP in constraint networks. Defined in 1985, this algorithm relies on a so-called pseudo-tree (defined below). It is shown to have a complexity which is exponential in the height of the pseudo-tree and uses only linear space. This pseudo-tree height was later shown to be related by a $\log(n)$ factor to induced width (a synonymous of tree-width) in 1995 [9, 105]. Pseudo-tree height seems to be the exact equivalent of tree-depth [92].

▶ **Definition 10** (pseudo-tree, pseudo-tree height [53])**.** *A pseudo-tree arrangement of a graph G is a rooted tree with the same vertices as G and the property that adjacent vertices in G reside in the same branch of the tree. The pseudo-tree height of G is the minimum, over all pseudo-tree arrangements of G of the height of the pseudo-tree arrangement.*

The idea was revived in the context of counting problems in the "Recursive Conditioning" algorithm, relying on the related notion of d-tree [37]. This was quickly followed by the proposal of two algorithms for CSP/CN, WCSP/CFN and MAP/MRF relying either on pseudo-trees, leading to the family of AND-OR tree and graph search algorithms [84, 86, 85, 87] or tree-decompositions, leading to the Backtrack Tree-Decomposition algorithm (BTD [66, 67, 41]) . These algorithms enhance the original pseudo-tree search along two lines: they add some form of local consistency enforcing to prune the search tree, with associated non trivial per-cluster lower-bound management and may also memorize information, leading to algorithms with worst-case time/space complexity that reach those of the best elimination algorithms, with much better empirical complexity, thanks to pruning and variable ordering heuristics.

These algorithms take as input a graphical model (we assume a pairwise GM for simplicity) and a tree decomposition of its graph. The tree decomposition is rooted in a chosen cluster. Then a usual (hybrid) tree-search algorithm is used but the variable assignment order is constrained by a rule that forbids to assign a variable from a cluster if all the variables of its parent cluster have not already been assigned. When this happens, the variables in the separators between the parent cluster and any of its sons are assigned: the functions that connect the parent and son clusters can be cheaply eliminated, the clusters disconnect and can be solved independently given the current assignment of the separator. This principle is applied recursively on sub-problems. For every first visit of a given assignment of a separator, it is possible to memorize the value of the query for this assignment. When the same assignment is revisited later, it can be directly reused. This is what the AND/OR graph search and BTD algorithms do, at the cost of an $O(d^s)$ space complexity. Their time complexity, in the simplest case of the CSP problem, is just exponential in the tree-width instead of the pseudo-tree height. Alternatively, no space is used in the AND/OR tree search algorithm and the worst-case space and time complexity become those of the Pseudo-Tree Search algorithm (but with improved empirical complexity thanks to *mini-bucket elimination*, a specific form of message passing that provides the bounds required for pruning [43, 46]).

Thanks to the bounds and reformulations provided by local consistency enforcing, these algorithms will typically explore a tiny fraction of the separators, empirically requiring much less space than a pure non-serial dynamic programming algorithm. The constraint that the rooted tree decomposition imposes on variable assignment ordering will however play a possibly negative role. Finally, because space is an abrupt constraint in practice, tree-decompositions with large separators may be unexploitable in practice if a space-intensive approach is used. One usual way to deal with space constraints is to use a bounded amount of space for *caching* separator assignments and corresponding query answers and have some dedicated cache management strategy. In the end, the best tree-decompositions for hybrid algorithms such as BTD or Pseudo-Tree Search will usually not be minimum tree-width decompositions (if they could be computed). The usual approach here is to rely on simple heuristics such as min-degree, Maximum Cardinality Search [100] or min-fill to heuristically build a decomposition. This may be randomized and iterated [70]. The result may then be improved by cluster-merging operations, trying to minimize separator sizes while preserving sufficient cluster size so that variable ordering heuristics do not get too constrained. Not only we do not have any final definition of what is an optimal tree decomposition but even a simple min-fill heuristics may be too long in practice to be useful (on large GMs, executing the polytime min-fill algorithm on the GM graph may take more time than solving the later NP-hard query on the GM). For this reason, new approaches that directly and efficiently build empirically useful decompositions for GM optimization queries are now being introduced [64, 63]. To better exploit the fact that different assignments may lead to different subproblems, with different graphs at the micro-structure level, dynamic decomposition is also explored. Some of these strategies rely on iterated (hyper)graph-partitioning using large-graphs dedicated heuristics such as Metis [69] or PaToH [24, 79], in the spirit of the seminal Nested Dissection algorithm [54].

Table 2 Time and space complexities of exact methods for a CFN with tree-width w, s maximum separator size ($s \leq w \leq n$), and initial problem upper bound k.

Exact Method	Time	Space
Depth First Branch & Bound	$O(\exp(n))$	$O(n)$
Hybrid Best-First Search [4]	$O(\exp(n))$	$O(\exp(n))$
Variable elimination [11]	$O(n \exp(w))$	$O(n \exp(w))$
Block-by-block elimination [11]	$O(n \exp(w))$	$O(n \exp(s))$
Pseudo-Tree Search [53, 9]	$O(n \exp(w \log(n)))$	$O(n)$
AND/OR tree search [84, 86]	$O(n \exp(w \log(n)))$	$O(n)$
AND/OR graph search [85, 87]	$O(n \exp(w))$	$O(n \exp(s))$
BTD and variants [66, 112, 41, 102, 4]	$O(k\, n \exp(w))$	$O(n \exp(s))$

In the *idempotent* cases, the exploitation of tree-decompositions is implicit in algorithms relying on a combination of clause/constraint learning and restarts (confirmed in practice [62]).

▶ **Theorem 11** ([6, 62]). *A standard randomized CDCL SAT-solver with a suitable learning strategy will decide the consistency of any pairwise Constraint Network instance with tree-width w with $O(n^{2w}d^{2w})$ expected restarts.*

8 Additional complexity results

Although NP-hard in general, under certain restrictions, calculating the global value of a GM can be achieved in polynomial time. We have seen that this is the case when the tree-width of the primal graph is bounded by a constant. In terms of graph structure, the family of GMs

with bounded tree-width has been clearly the most exploited in practice (with the closely related branch-width [98]). In the non-pairwise case, parameters such as hypertree-width are implicitly exploited by hybrid algorithms such as BTD [65].

In the case of CFNs, restrictions on the language of cost functions can also define tractable classes. Restrictions which are not exclusively concerned with the graph, nor exclusively concerned with the language of cost functions define what are known as hybrid classes. We consider language-based tractable classes first.

The characterisation of all tractable languages of cost functions was a long-standing problem which has recently been solved thanks to the characterisation of tractable languages in the last remaining (and most difficult) case of Boolean valuation structures [22, 119]. This latter result resolved positively the so-called Feder and Vardi Conjecture [52] that there is a P/NP-complete dichotomy for the constraint satisfaction problem parameterized by the language of possible constraint relations [23].

There is only one non-trivial class of cost functions defining a tractable subproblem of CFNs which comes out of the study of Max-SAT: the class of submodular functions (see Definition 7). The class of submodular functions includes all unary functions, the binary functions $\sqrt{x^2+y^2}$, $((x \geq y)?(x-y)^t : k)$ (for $t \geq 1$), $K - xy$, the cut function of a graph and the rank function of a matroid [55]. Efficient algorithms have been developed in Operations Research to minimize submodular functions [82, 25]. Another approach [34] consists in establishing virtual arc consistency (which preserves the submodularity of the cost functions). By the definition of VAC, the arc-consistency closure $\mathcal{Q}$ of Bool($\mathcal{P}$) is non-empty and, by definition of submodularity, its cost functions are both *min-closed* and *max-closed* [61]; to find a solution of $\mathcal{Q}$ (which is necessarily an optimal solution of $\mathcal{P}$), it suffices to assign always the minimum (or always the maximum) value in each domain of $\mathcal{Q}$.

A CFN can be coded as an integer programming problem (whose variables include $v_{ia} \in \{0, 1\}$ which is equal to 1 if and only if $X_i = a$ in the original CFN instance [60]). The linear relaxation of this integer programming problem (in which v_{ia} is now a real number in the interval $[0, 1]$) has integer optimal solutions if all cost functions are submodular, meaning that the CFN can be solved by linear programming. The dual of this relaxation is exactly the linear program used by OSAC [29] to find an optimal transformation (set of EPTs) [116]. CFNs restricted to a language of finite-valued cost functions over the valuation structure $\mathbb{Q}^{\geq 0} \cup \{\infty\}$ is tractable if and only if it is solved by this linear program [113]. This notably includes languages of cost functions that are submodular on arbitrary lattices. State-of-the-art results concerning the tractability of languages of cost functions are covered in detail in a recent survey article [78].

These theoretical results demonstrate the importance of submodularity and linear programming in the search for tractable subproblems of the CFN. However, we should mention that there are tractable languages of cost functions other than the class of submodular functions. For example, replacing the functions min and max in the definition of submodularity by *any* pair of commutative conservative functions f, g (where conservative means that $\forall x, y$, $f(x, y), g(x, y) \in \{x, y\}$) gives rise to a tractable class [28].

Hybrid classes [36] may be defined by restrictions on the micro-structure of the CFN. To illustrate this, we describe the hybrid tractable class defined by the *joint winner property* (JWP) [35]. A class of *binary* CFNs satisfies the JWP if for any three variable-value assignments (to three distinct variables), the multiset of pairwise costs imposed by the binary cost functions does not have a unique minimum. If there is no cost function explicitly defined on a pair of variables, then it is considered to be a constant-$\mathbf{0}$ binary cost function. Note that the unary cost functions in a CFN that satisfies the JWP can be arbitrary. It has been

shown that there is a close link between the JWP and $M^\sharp$-convex functions studied in [91]. Indeed, a function satisfying the JWP can be transformed into a function represented as the sum of two quadratic M-convex functions, which can be minimized in polynomial time via an M-convex intersection algorithm. This leads to a larger tractable class of binary finite-valued CFNs, which properly contains the JWP class [59].

9 Solvers and applications

Thanks to a simple algebraic formulation of graphical models and queries over GMs, this review covers a very large variety of algorithms and approaches that have been developed in probabilistic reasoning, propositional logic and constraint programming. On the non stochastic side, SAT and CSP solvers have had a significant impact in several areas such as Electronic Design Automation (where SAT solvers are used as oracles to solve complex Bounded Model Checking problems), Software Verification (where suitable Abstraction Refinement Strategies are used to manage possibly very large instances [27, 58]) or for task scheduling [8] or planning [114] among many others. Stochastic variants are one of many modeling tools available in Machine Learning [16], but Markov Random Fields have been more specifically applied in Image processing for tasks such as segmentation [83]. In most Image Processing applications, a variable of the GM is associated with a single pixel, leading to very large MAP/MRF optimization problems that are never solved exactly (with a proof or theoretical guarantee), using instead heuristic or primal/dual approaches offering a final optimality gap [68]. However, the class of submodular problems with Boolean domains, solvable in polytime using a min-cut (max-flow) algorithm has been exploited intensely under the name of the "Graph-Cut" algorithm [76, 120].

In this final section, we rapidly describe recent results we obtained using an exact WCSP/CFN solver. Toulbar2 implements most of all the algorithms we have described above to solve the WCSP/CFN problem, which is essentially equivalent (up lo a $-\log$ transform and bounded precision description of real numbers) to MAP/MRF. Toulbar2 is becoming increasingly famous for its ability to solve "Computational Protein Design" problem instances. Proteins are linear polymers made of elementary bricks called amino acids. Each amino acid a in a protein can be chosen among a fixed alphabet of 20 natural amino acids and each choice in this alphabet is defined by the combination of a fixed "backbone" part and a variable side-chain, a highly flexible part on the molecule. The computational protein design problem starts from the three-dimensional description of a protein backbone and asks to determine the nature and orientation of all side-chains that minimizes the energy of the resulting molecule (a minimum of energy defining a most stable choice for a given rigid backbone). The combination of dedicated pairwise separable (non-convex) energy functions with a discretization of orientations in side chains (so-called rotamer libraries) allows to reduce this problem to a pure WCSP/CFN problem. On such problems, Toulbar2 has been used to bring to light the limitations of dedicated highly optimized Simulated annealing implementations [111], being capable of providing guaranteed optimal solutions for problems with search space larger than 400^{100} in reasonable time on a single core, on problems that cannot be tackled by state-of-the-art ILP, Weighted MaxSAT or quadratic boolean solvers, even those based on powerful SDP-based bounds [2]. The instances are not permutated submodular [108] and do not have a sparse graph that would make dynamic programming feasible. This has enabled the design of a real new self-assembling protein [93].

Conclusion

Algorithms for processing graphical models have been explored mostly independently in the stochastic and deterministic cases. In this review, we have tried, using simple algebra, to cover a large set of modeling frameworks with associated NP-hard queries and showed that there is a strong common core for several algorithms in either case: non serial dynamic programming. Restricted to sub-problems, combined with tree-search, learned heuristics, and other tricks, these bricks have generated impressive algorithmic progress in the exact solving of SAT/PL, CSP/CN, WCSP/CFN aka MAP/MRF or MPE/BN problems and even for #P-complete problems [26, 50].

References

1 Srinivas M Aji and Robert J McEliece. The generalized distributive law. *IEEE transactions on Information Theory*, 46(2):325–343, 2000.

2 David Allouche, Isabelle André, Sophie Barbe, Jessica Davies, Simon de Givry, George Katsirelos, Barry O'Sullivan, Steve Prestwich, Thomas Schiex, and Seydou Traoré. Computational protein design as an optimization problem. *Artificial Intelligence*, 212:59–79, 2014.

3 David Allouche, Christian Bessiere, Patrice Boizumault, Simon De Givry, Patricia Gutierrez, Jimmy HM Lee, Ka Lun Leung, Samir Loudni, Jean-Philippe Métivier, Thomas Schiex, and Yi Wu. Tractability-preserving transformations of global cost functions. *Artificial Intelligence*, 238:166–189, 2016.

4 David Allouche, Simon De Givry, George Katsirelos, Thomas Schiex, and Matthias Zytnicki. Anytime hybrid best-first search with tree decomposition for weighted CSP. In *Principles and Practice of Constraint Programming*, pages 12–29. Springer, 2015.

5 Albert Atserias, Andrei Bulatov, and Victor Dalmau. On the power of k-consistency. In *International Colloquium on Automata, Languages, and Programming*, pages 279–290. Springer, 2007.

6 Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. *Journal of Artificial Intelligence Research*, 40:353–373, 2011.

7 Fahiem Bacchus and Adam Grove. Graphical models for preference and utility. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 3–10. Morgan Kaufmann Publishers Inc., 1995.

8 Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-based scheduling: applying constraint programming to scheduling problems*, volume 39. Springer Science & Business Media, 2012.

9 Roberto Bayardo and Daniel Miranker. On the space-time trade-off in solving constraint satisfaction problems. In *Proc. of the 14^{th} IJCAI*, pages 558–562, Montréal, Canada, 1995.

10 Claude Berrou, Alain Glavieux, and Punya Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo-codes. 1. In *Proceedings of ICC'93-IEEE International Conference on Communications*, volume 2, pages 1064–1070. IEEE, 1993.

11 Umberto Bertelé and Francesco Brioshi. *Nonserial Dynamic Programming*. Academic Press, 1972.

12 C. Bessière and J-C. Régin. Refining the basic constraint propagation algorithm. In *Proc. IJCAI'2001*, pages 309–315, 2001.

13 Christian Bessière. Arc-consistency and arc-consistency again. *Artificial Intelligence*, 65:179–190, 1994.

14 Armin Biere, Marijn Heule, and Hans van Maaren, editors. *Handbook of Satisfiability*, volume 185. IOS press, 2009.

15 Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. Conflict-driven clause learning sat solvers. *Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications*, pages 131–153, 2009.

16 Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

17 Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM (JACM)*, 44(2):201–236, 1997.

18 H L Bodlaender and A M C A Koster. Treewidth Computations I. Upper Bounds. Technical Report UU-CS-2008-032, Utrecht University, Department of Information and Computing Sciences, Utrecht, The Netherlands, September 2008. URL: `http://www.cs.uu.nl/research/techreps/repo/CS-2008/2008-032.pdf`.

19 Hans L Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical computer science*, 209(1-2):1–45, 1998.

20 E. Boros and P. Hammer. Pseudo-Boolean Optimization. *Discrete Appl. Math.*, 123:155–225, 2002.

21 Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *ECAI*, volume 16, page 146, 2004.

22 Andrei A. Bulatov. A dichotomy theorem for nonuniform csps. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 319–330. IEEE Computer Society, 2017. `doi:10.1109/FOCS.2017.37`.

23 Andrei A. Bulatov. A short story of the CSP dichotomy conjecture. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, page 1. IEEE, 2019. `doi:10.1109/LICS.2019.8785678`.

24 Ümit Çatalyürek and Cevdet Aykanat. Patoh (partitioning tool for hypergraphs). *Encyclopedia of Parallel Computing*, pages 1479–1487, 2011.

25 Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. Subquadratic submodular function minimization. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1220–1231. ACM, 2017. `doi:10.1145/3055399.3055419`.

26 Supratik Chakraborty, Kuldeep S Meel, and Moshe Y Vardi. A scalable approximate model counter. In *International Conference on Principles and Practice of Constraint Programming*, pages 200–216. Springer, 2013.

27 Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *International Conference on Computer Aided Verification*, pages 154–169. Springer, 2000.

28 David A. Cohen, Martin C. Cooper, and Peter Jeavons. Generalising submodularity and horn clauses: Tractable optimization problems defined by tournament pair multimorphisms. *Theor. Comput. Sci.*, 401(1-3):36–51, 2008. `doi:10.1016/j.tcs.2008.03.015`.

29 M. Cooper, S. de Givry, M. Sanchez, T. Schiex, M. Zytnicki, and T. Werner. Soft arc consistency revisited. *Artificial Intelligence*, 174:449–478, 2010.

30 M C. Cooper. An optimal k-consistency algorithm. *Artificial Intelligence*, 41:89–95, 1989.

31 M C. Cooper. High-order consistency in Valued Constraint Satisfaction. *Constraints*, 10:283–305, 2005.

32 M C. Cooper, S. de Givry, and T. Schiex. Optimal soft arc consistency. In *Proc. of IJCAI'2007*, pages 68–73, Hyderabad, India, January 2007.

33 Martin Cooper and Thomas Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154(1-2):199–227, 2004.

34 Martin C Cooper, Simon de Givry, Martí Sánchez, Thomas Schiex, and Matthias Zytnicki. Virtual Arc Consistency for Weighted CSP. In *AAAI*, volume 8, pages 253–258, 2008.

35 Martin C. Cooper and Stanislav Zivny. Hybrid tractability of valued constraint problems. *Artif. Intell.*, 175(9-10):1555–1569, 2011. `doi:10.1016/j.artint.2011.02.003`.

36 Martin C. Cooper and Stanislav Zivny. Hybrid tractable classes of constraint problems. In Andrei A. Krokhin and Stanislav Zivny, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 113–135. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/DFU.Vol7.15301.4.

37 Adnan Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1-2):5–41, 2001.

38 Adnan Darwiche and Pierre Marquis. Compiling propositional weighted bases. *Artificial Intelligence*, 157(1-2):81–113, 2004.

39 Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

40 Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3):201–215, 1960.

41 S. de Givry, T. Schiex, and G. Verfaillie. Exploiting Tree Decomposition and Soft Local Consistency in Weighted CSP. In *Proc. of the National Conference on Artificial Intelligence, AAAI-2006*, pages 22–27, 2006.

42 Rina Dechter. Learning while searching in constraint satisfaction problems. In *Proc. of AAAI'86*, pages 178–183, Philadelphia, PA, 1986.

43 Rina Dechter. Mini-buckets: A general scheme for generating approximations in automated reasoning. In *IJCAI*, pages 1297–1303, 1997.

44 Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1–2):41–85, 1999.

45 Rina Dechter and Irina Rish. Directional resolution: The Davis-Putnam procedure, revisited. *KR*, 94:134–145, 1994.

46 Rina Dechter and Irina Rish. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM (JACM)*, 50(2):107–153, 2003.

47 C. Domshlak, F. Rossi, KB Venable, and T. Walsh. Reasoning about soft constraints and conditional preferences: complexity results and approximation techniques. In *Proc. of the 18^{th} IJCAI*, pages 215–220, Acapulco, Mexico, 2003.

48 Didier Dubois, Hélene Fargier, and Henri Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *Second IEEE International Conference on Fuzzy Systems*, pages 1131–1136. IEEE, 1993.

49 Didier Dubois, Hélène Fargier, and Henri Prade. Propagation and satisfaction of fuzzy constraints. In Yager R.R. and Zadeh L.A., editors, *Fuzzy Sets, Neural Networks and Soft Computing*, pages 166–187. Kluwer Acad., 1993.

50 Stefano Ermon, Carla Gomes, Ashish Sabharwal, and Bart Selman. Taming the curse of dimensionality: Discrete integration by hashing and optimization. In *International Conference on Machine Learning*, pages 334–342, 2013.

51 Hélene Fargier, Pierre Marquis, Alexandre Niveau, and Nicolas Schmidt. A knowledge compilation map for ordered real-valued decision diagrams. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

52 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. doi:10.1137/S0097539794266766.

53 Eugene C. Freuder and Michael J. Quinn. Taking advantage of stable sets of variables in constraint satisfaction problems. In *Proc. of the 9^{th} IJCAI*, pages 1076–1078, Los Angeles, CA, 1985.

54 Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973.

55 M. Gondran and M. Minoux. *Graphes et algorithmes*. Lavoisier, EDF R&D, 2009.

56 Michel Gondran and Michel Minoux. *Graphs, dioids and semirings: new models and algorithms*, volume 41. Springer Science & Business Media, 2008.

57 Stefan Haller, Paul Swoboda, and Bogdan Savchynskyy. Exact map-inference by confining combinatorial search with lp relaxation. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

58 Marijn JH Heule and Oliver Kullmann. The science of brute force. *Commun. ACM*, 60(8):70–79, 2017.

59 Hiroshi Hirai, Yuni Iwamasa, Kazuo Murota, and Stanislav Zivny. A tractable class of binary VCSPs via M-convex intersection. *ACM Trans. Algorithms*, 15(3):44:1–44:41, 2019. doi:10.1145/3329862.

60 Barry Hurley, Barry O'Sullivan, David Allouche, George Katsirelos, Thomas Schiex, Matthias Zytnicki, and Simon de Givry. Multi-language evaluation of exact solvers in graphical model discrete optimization. *Constraints*, pages 1–22, 2016.

61 Peter Jeavons and Martin C. Cooper. Tractable constraints on ordered domains. *Artif. Intell.*, 79(2):327–339, 1995. doi:10.1016/0004-3702(95)00107-7.

62 Peter Jeavons and Justyna Petke. Local consistency and SAT-solvers. *Journal of Artificial Intelligence Research*, 43:329–351, 2012.

63 Philippe Jégou, Hanan Kanso, and Cyril Terrioux. Towards a dynamic decomposition of CSPs with separators of bounded size. In *International Conference on Principles and Practice of Constraint Programming*, pages 298–315. Springer, 2016.

64 Philippe Jégou, Hélène Kanso, and Cyril Terrioux. On the relevance of optimal tree decompositions for constraint networks. In *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 738–743. IEEE, 2018.

65 Philippe Jégou, Samba Ndojh Ndiaye, and Cyril Terrioux. A new evaluation of forward checking and its consequences on efficiency of tools for decomposition of CSPs. In *2008 20th IEEE International Conference on Tools with Artificial Intelligence*, volume 1, pages 486–490. IEEE, 2008.

66 Philippe Jégou and Cyril Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146(1):43–75, 2003.

67 Philippe Jégou and Cyril Terrioux. Decomposition and good recording for solving Max-CSPs. In *Proceedings of the 16th European Conference on Artificial Intelligence, IOS Press*, pages 196–200, 2004.

68 Joerg Kappes, Bjoern Andres, Fred Hamprecht, Christoph Schnorr, Sebastian Nowozin, Dhruv Batra, Sungwoong Kim, Bernhard Kausler, Jan Lellmann, Nikos Komodakis, et al. A comparative study of modern inference techniques for discrete energy minimization problems. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1328–1335, 2013.

69 George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

70 Kalev Kask, Andrew Gelfand, Lars Otten, and Rina Dechter. Pushing the power of stochastic greedy ordering schemes for inference in graphical models. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

71 Ross Kindermann. *Markov random fields and their applications*. American Mathematical Society, 1980.

72 E.P. Klement, R. Mesiar, and E. Pap. *Triangular Norms*. Kluwer Academic Publishers, 2000.

73 Juerg Kohlas and Nic Wilson. Semiring induced valuation algebras: Exact and approximate local computation algorithms. *Artificial Intelligence*, 172(11):1360–1399, 2008.

74 Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

75 Vladimir Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(10):1568–1583, 2006.

76 Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 2:147–159, 2004.

77 V.K. Koval and M.I. Schlesinger. Dvumernoe programmirovanie v zadachakh analiza izobrazheniy (two-dimensional programming in image analysis problems),. *USSR Academy of Science, Automatics and Telemechanics*, 8:149–168, 1976.

78 Andrei A. Krokhin and Stanislav Zivny. The complexity of valued CSPs. In Andrei A. Krokhin and Stanislav Zivny, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 233–266. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/DFU.Vol7.15301.9.

79 Jean-Marie Lagniez and Pierre Marquis. An improved decision-DNNF compiler. In *IJCAI*, pages 667–673, 2017.

80 J. Larrosa, S. de Givry, F. Heras, and M. Zytnicki. Existential arc consistency: getting closer to full arc consistency in weighted CSPs. In *Proc. of the 19th IJCAI*, pages 84–89, Edinburgh, Scotland, August 2005.

81 Jimmy Ho-Man Lee and Ka Lun Leung. Consistency techniques for flow-based projection-safe global cost functions in weighted constraint satisfaction. *Journal of Artificial Intelligence Research*, 43(1):257–292, 2012.

82 Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 1049–1065. IEEE, 2015.

83 Stan Z Li. *Markov random field modeling in image analysis*. Springer Science & Business Media, 2009.

84 R. Marinescu and R. Dechter. AND/OR branch-and-bound for graphical models. In *Proc. of the 19th IJCAI*, page 224, Edinburgh, Scotland, 2005.

85 Radu Marinescu and Rina Dechter. Memory intensive branch-and-bound search for graphical models. In *Proc. of AAAI'06*, page 1200. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

86 Radu Marinescu and Rina Dechter. And/or branch-and-bound search for combinatorial optimization in graphical models. *Artificial Intelligence*, 173(16-17):1457–1491, 2009.

87 Radu Marinescu and Rina Dechter. Memory intensive and/or search for combinatorial optimization in graphical models. *Artificial Intelligence*, 173(16-17):1492–1524, 2009.

88 P. Meseguer, F. Rossi, and T. Schiex. Soft constraints processing. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 9. Elsevier, 2006.

89 R. Mohr and T.C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28(2):225–233, 1986.

90 Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001.

91 Kazuo Murota. *Discrete Convex Analysis*. SIAM, 2003.

92 Jaroslav Nešetřil and Patrice Ossona De Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *European Journal of Combinatorics*, 27(6):1022–1041, 2006.

93 Hiroki Noguchi, Christine Addy, David Simoncini, Staf Wouters, Bram Mylemans, Luc Van Meervelt, Thomas Schiex, Kam YJ Zhang, Jeremy RH Tame, and Arnout RD Voet. Computational design of symmetrical eight-bladed β-propeller proteins. *IUCrJ*, 6(1), 2019.

94 Abdelkader Ouali, David Allouche, Simon de Givry, Samir Loudni, Yahia Lebbah, Lakhdar Loukil, and Patrice Boizumault. Variable neighborhood search for graphical model energy minimization. *Artificial Intelligence*, 278:103194, 2020. doi:10.1016/j.artint.2019.103194.

95 Youngsuk Park, David Hallac, Stephen Boyd, and Jure Leskovec. Learning the network structure of heterogeneous data via pairwise exponential Markov random fields. *Proceedings of machine learning research*, 54:1302, 2017.

96 Judea Pearl. *Probabilistic Reasoning in Intelligent Systems, Networks of Plausible Inference*. Morgan Kaufmann, Palo Alto, 1988.

97 Daniel Prusa and Tomas Werner. Universality of the local marginal polytope. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1738–1743, 2013.

98 Neil Robertson and Paul D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *J. Comb. Theory, Ser. B*, 52(2):153–190, 1991. doi:10.1016/0095-8956(91)90061-N.

99 J. Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–44, 1965.

100 D.J. Rose. Tringulated graphs and the elimination process. *Journal of Mathematical Analysis and its Applications*, 32, 1970.

101 F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.

102 M. Sanchez, D. Allouche, S. de Givry, and T. Schiex. Russian doll search with tree decomposition. In *Proc. IJCAI'09*, pages 603–608, San Diego (CA), USA, 2009.

103 T. Schiex. Arc consistency for soft constraints. In *Principles and Practice of Constraint Programming - CP 2000*, volume 1894 of *LNCS*, pages 411–424, Singapore, September 2000.

104 T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: hard and easy problems. In *Proc. of the 14^{th} IJCAI*, pages 631–637, Montréal, Canada, August 1995.

105 Thomas Schiex. A note on csp graph parameters. Technical report, Citeseer, 1999.

106 Thomas Schiex and Gérard Verfaillie. Nogood recording for static and dynamic constraint satisfaction problems. *International Journal on Artificial Intelligence Tools*, 3(02):187–207, 1994.

107 D. Schlesinger. Exact Solution of Permuted Submodular MinSum Problems. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, number 4679/2007 in LNCS, pages 28–38, August 2007.

108 Dmitrij Schlesinger. Exact solution of permuted submodular minsum problems. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 28–38. Springer, 2007.

109 M.I. Schlesinger. Sintaksicheskiy analiz dvumernykh zritelnikh signalov v usloviyakh pomekh (Syntactic analysis of two-dimensional visual signals in noisy conditions). *Kibernetika*, 4:113–130, 1976.

110 G. Shafer. An axiomatic study of computation in hypertrees. Working paper 232, University of Kansas, School of Business, Lawrence, 1991.

111 David Simoncini, David Allouche, Simon de Givry, Céline Delmas, Sophie Barbe, and Thomas Schiex. Guaranteed discrete energy optimization on large protein design problems. *Journal of Chemical Theory and Computation*, 11(12):5980–5989, 2015. doi:10.1021/acs.jctc.5b00594.

112 C. Terrioux and P. Jegou. Bounded backtracking for the valued constraint satisfaction problems. In *Proc. of the Ninth International Conference on Principles and Practice of Constraint Programming (CP-2003)*, 2003.

113 Johan Thapper and Stanislav Zivny. The complexity of finite-valued CSPs. *J. ACM*, 63(4):37:1–37:33, 2016. doi:10.1145/2974019.

114 Peter Van Beek and Xinguang Chen. Cplan: A constraint programming approach to planning. In *AAAI/IAAI*, pages 585–590, 1999.

115 Martin Wainwright, Tommi Jaakkola, and Alan Willsky. Tree-based reparameterization analysis of belief propagation and related algorithms for approximate inference on graphs with cycles. In *Proceedings IEEE International Symposium on Information Theory*, page 113. IEEE, 2002.

116 T. Werner. A Linear Programming Approach to Max-sum Problem: A Review. *IEEE Trans. on Pattern Recognition and Machine Intelligence*, 29(7):1165–1179, July 2007. doi:10.1109/TPAMI.2007.1036.

117 Jonathan S Yedidia, William T Freeman, and Yair Weiss. Bethe free energy, Kikuchi approximations, and belief propagation algorithms. *Advances in neural information processing systems*, 13, 2001.

118 Yuanlin Zhang and Roland HC Yap. Making AC-3 an optimal algorithm. In *IJCAI*, volume 1, pages 316–321, 2001.

119 Dmitriy Zhuk. A proof of CSP dichotomy conjecture. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 331–342. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.38.

120 Stanislav Živnỳ and Peter G Jeavons. Classes of submodular constraints expressible by graph cuts. *Constraints*, 15(3):430–452, 2010.

Inapproximability Results for Scheduling with Interval and Resource Restrictions

Marten Maack
Department of Computer Science, Kiel University, Kiel, Germany
mmaa@informatik.uni-kiel.de

Klaus Jansen
Department of Computer Science, Kiel University, Kiel, Germany
kj@informatik.uni-kiel.de

Abstract

In the restricted assignment problem, the input consists of a set of machines and a set of jobs each with a processing time and a subset of eligible machines. The goal is to find an assignment of the jobs to the machines minimizing the makespan, that is, the maximum summed up processing time any machine receives. Herein, jobs should only be assigned to those machines on which they are eligible. It is well-known that there is no polynomial time approximation algorithm with an approximation guarantee of less than 1.5 for the restricted assignment problem unless P=NP. In this work, we show hardness results for variants of the restricted assignment problem with particular types of restrictions.

For the case of interval restrictions – where the machines can be totally ordered such that jobs are eligible on consecutive machines – we show that there is no polynomial time approximation scheme (PTAS) unless P=NP. The question of whether a PTAS for this variant exists was stated as an open problem before, and PTAS results for special cases of this variant are known.

Furthermore, we consider a variant with resource restriction where the sets of eligible machines are of the following form: There is a fixed number of (renewable) resources, each machine has a capacity, and each job a demand for each resource. A job is eligible on a machine if its demand is at most as big as the capacity of the machine for each resource. For one resource, this problem has been intensively studied under several different names and is known to admit a PTAS, and for two resources the variant with interval restrictions is contained as a special case. Moreover, the version with multiple resources is closely related to makespan minimization on parallel machines with a low rank processing time matrix. We show that there is no polynomial time approximation algorithm with a rate smaller than $48/47 \approx 1.02$ or 1.5 for scheduling with resource restrictions with 2 or 4 resources, respectively, unless P=NP. All our results can be extended to the so called Santa Claus variants of the problems where the goal is to maximize the minimal processing time any machine receives.

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

Keywords and phrases Scheduling, Restricted Assignment, Approximation, Inapproximability, PTAS

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.5

Related Version A full version of the paper is available at `http://arxiv.org/abs/1907.03526`.

Funding *Marten Maack*: German Academic Exchange Service (DAAD) scholarship
Klaus Jansen: German Research Foundation (DFG) project JA 612/15-2

Acknowledgements We thank Malin Rau and Lars Rohwedder for helpfull discussions on the problem.

1 Introduction

Consider the restricted assignment problem: Given a set of machines $\mathcal{M}$ and a set of jobs $\mathcal{J}$ with a processing time or size p_j and a subset of eligible machines $\mathcal{M}(j) \subseteq \mathcal{M}$ for each job j, the goal is to find a schedule $\sigma : \mathcal{J} \to \mathcal{M}$ with $\sigma(j) \in \mathcal{M}(j)$ for each job j and minimizing the makespan $C_{\max}(\sigma) = \max_{i \in \mathcal{M}} \sum_{j \in \sigma^{-1}(i)} p_j$.

In a seminal work, Lenstra, Shmoys and Tardos [19] presented a 2-approximation for restricted assignment and also showed that there is no polynomial time approximation algorithm with rate smaller than 1.5 for the problem, unless P=NP. Closing this gap is a prominent open problem in approximation and scheduling theory [26, 33]. If there are no restrictions, i.e., $\mathcal{M}(j) = \mathcal{M}$ for each job j, we have the classical problem of makespan minimization on identical parallel machines (machine scheduling) which is already strongly NP-hard. On the other hand, machine scheduling is well-known to admit a polynomial time approximation scheme (PTAS) due to a classical result by Hochbaum and Shmoys [10]. In recent years, the approximability of special cases of restricted assignment has been intensively studied (see, e.g., [3, 7, 12, 14]) with one line of research focusing on the existence of approximation schemes (see, e.g., [8, 13, 23, 24]). The present work seeks to contribute in this research direction. Omitted details and proofs can be found in the long version of the paper [22].

Interval Restrictions. Arguably one of the most natural variants of the restricted assignment problem is the case of scheduling with interval restrictions (RAI). In this variant, the machines are totally ordered and each job is eligible on consecutive machines. More precisely, we have $\mathcal{M} = \{M_1, \ldots, M_m\}$, and for each job j we have some indices $\ell, r \in [m]$ such that $\mathcal{M}(j) = \{M_\ell, \ldots, M_r\}$. Several special cases of RAI are known to admit a PTAS: the hierarchical case [24], where for each job the interval of eligible machines starts with the first machine; the nested case [23, 8], where $\mathcal{M}(j) \subseteq \mathcal{M}(j')$, $\mathcal{M}(j') \subseteq \mathcal{M}(j)$ or $\mathcal{M}(j) \cap \mathcal{M}(j') = \emptyset$ for each pair of jobs (j, j'); and the inclusion-free case [27, 17], where $\mathcal{M}(j) \subseteq \mathcal{M}(j')$ implies that j and j' share either their first or last eligible machine. Furthermore, for general RAI, a $2 - 2/(\max_{j \in \mathcal{J}} p_j)$-approximation due to Schwarz [27] is known (assuming integral processing times); and the special case with two distinct processing times is even polynomial time solvable [32]. Note that the problem has also been studied in the context of online algorithms (see [18, 21]).

The question of whether there is a PTAS for RAI has been posed by several authors [15, 27, 32]. As the main result of the present work, we resolve this question in the negative:

▶ **Theorem 1.** *There is no PTAS for scheduling with interval restrictions unless P=NP.* [1]

We prove this theorem in Section 2.

Resource Restrictions. The second variant considered in this work, is the problem of scheduling with resource restrictions with R resources (RAR(R)). Herein, a set $\mathcal{R}$ of R (renewable) resources is given, each machine i has a resource capacity $c_r(i)$ and each job j has a resource demand $d_r(j)$ for each $r \in \mathcal{R}$. Job j is eligible on machine i if $d_r(j) \leq c_r(i)$ for each resource r. For $R = 1$, the problem is equivalent to the mentioned hierarchical case and

[1] There is a paper [16] claiming to have found a PTAS for RAI. However, according to [29], the result is not correct and the authors published a revised version of the paper [17] claiming a less general result, namely, a PTAS for the inclusion-free case.

has been studied intensively [20, 21]. Furthermore, it is not hard to see that RAI is properly placed between RAR(1) and RAR(2) and hence there is a close relationship between the two problems. For arbitrary R, the problem was mentioned in a work by Bhaskara et el. [1] under the name of geometrically restricted scheduling[2] but to the best of our knowledge it has not been further studied up to now. There is, however, a close relationship to the low rank version of makespan minimization on unrelated parallel machines (unrelated scheduling) introduced in [1]. For scheduling with resource restrictions, we show:

▶ **Theorem 2.** *There is no approximation algorithm with rate less than* $48/47 \approx 1.02$ *or* 1.5 *for scheduling with resource restrictions with* 2 *or* 4 *resources, respectively, unless P=NP.*

We prove this theorem in Section 3. In the long version of the paper [22], we provide more details concerning RAR(R), e.g., a comparative analysis of RAR(R), RAI and low rank unrelated scheduling.

Santa Claus. The problems of restricted assignment and unrelated scheduling are also studied with the reverse objective of maximizing the minimal machine load $\min_{i \in \mathcal{M}} \sum_{j \in \sigma^{-1}(i)} p_{ij}$. Machine scheduling problems with this objective are sometimes called the Santa Claus version of the respective problem. We remark that all our results can be transferred to the Santa Claus versions of the considered problems in a straight-forward fashion.

Further Related Work. First note that if the number of machines is constant, there is a fully polynomial time approximation scheme (FPTAS) already for unrelated scheduling [11]. Furthermore, for some broad overview concerning parallel machine scheduling with different kinds of restrictions in the context of online and approximation algorithms, we refer to the surveys by Lee et al. [18] and Leung and Li [20, 21].

We already discussed many variants of restricted assignment that admit a PTAS. In particular, Ou, Leung and Li [24] presented a PTAS for the hierarchical case; Epstein and Levin [8] and Muratore, Schwarz and Woeginger [23] for the nested case; and Schwarz [27] and Khodamoradi et al. [17] for the inclusion-free case. Another case that has been studied in the literature is the tree-hierarchical case, where the machines can be arranged in a rooted tree such that for each job the set of eligible machines corresponds to a path starting at the root. It was shown to admit a PTAS by Epstein and Levin [8] and Schwarz [28]. It is not hard to see that all of the above cases contain the hierarchical case as a subcase, and that the tree-hierarchical, nested and inclusion-free case are distinct. There is, however, a variant admitting a PTAS that covers both the nested and the tree-hierarchical case: For each instance of the restricted assignment problem the corresponding incidence graph is a bipartite graph whose nodes are given by the jobs and machines and a job j is adjacent to a machine i if j is eligible on i. Jansen, Maack and Solis-Oba [13] showed that there is PTAS for restricted assignment for the case that the clique- or rank-width of the incidence graph is constant. Furthermore, if the incidence graph is a bi-cograph the clique-width is well-known to be small and this case covers the nested and tree-hierarchical case. The inclusion-free case, on the other hand, is equivalent to the case that the incidence graph is a bipartite permutation graph [17] which does not have a bounded clique-width [2]. Note that RAR(1) or RAI are equivalent to the cases that the incidence graph is a chain [9] or convex graph [16], respectively.

[2] The demands $d(j)$ and capacities $c(i)$ may be interpreted as points in R-dimensional space.

Preliminaries. We consider polynomial time approximation algorithms: Given an instance I of an optimization problem, an α-approximation A for this problem produces a solution in time poly($|I|$), where $|I|$ denotes the input length. For the objective function value $A(I)$ of this solution it is guaranteed that $A(I) \leq \alpha \textsc{opt}(I)$, in the case of an minimization problem, or $A(I) \geq (1/\alpha)\textsc{opt}(I)$, in the case of an maximization problem, where $\textsc{opt}(I)$ is the value of an optimal solution. We call α the *approximation guarantee* or *rate* of the algorithm. In some cases a polynomial time approximation scheme (PTAS) can be achieved, that is, an $(1+\varepsilon)$-approximation for each $\varepsilon > 0$. If for such a family of algorithms the running time is polynomial in both $1/\varepsilon$ and $|I|$ it is called *fully polynomial* (FPTAS).

Nearly all the reductions in this work follow the same pattern: Given an instance I of the starting problem, we construct an instance I' of the variant of the restricted assignment problem considered in the respective case. For I', all job sizes are integral and upper bounded by some constant T such that the overall size of the jobs equals $|\mathcal{M}|T$. Obviously, if for such an instance a machine receives jobs with overall size more or *less* than T, the makespan of the schedule is greater than T. Then we show that that there exists a schedule with makespan T for I', if and only if I is a yes-instance. This rules out the existence of an approximation algorithm with rate smaller than $(T+1)/T$ (or $T/(T-1)$ for the Santa Claus version) and a PTAS in particular.

2 Interval Restrictions

The sole goal of this section is to prove Theorem 1, that is, the non-existence of a PTAS for RAI (given P≠NP). Our starting point for the reduction is a satisfiability problem 3-SAT* that we tailor to our needs. We show that 3-SAT* is NP-hard via a straight forward reduction from the 1-in-3-SAT problem, which is well-know to be NP-complete [25] and discussed in more detail below. Next, we provide a reduction from 3-SAT* to the classical restricted assignment problem (with arbitrary sets of eligible machines). This reduction introduces some of the needed gadgets and ideas for the main result which is discussed in detail thereafter.

Starting Point. An instance of 1-in-3-SAT is a conjunction of clauses with 3 literals each. Each clause is a formula depending on 3 literals that is satisfied if and only if exactly one of its literals takes the value $\top$. We call such formulas 1-in-3-clauses in the following and define 2-in-3-clauses correspondingly. Note that we denote the truth values "true" and "false" by $\top$ and $\bot$ in the following. An instance of the problem 3-SAT* also is a conjunction of clauses with exactly 3 literals each. However, each of the clauses is either a 1-in-3-clause or a 2-in-3-clause and there are as many clauses of the first as of the second type. Furthermore, we require that each literal occurs *exactly twice*. In the following, we denote a 1-in-3-clause or 2-in-3-clause with literals z_1, z_2 and z_3 by $(z_1, z_2, z_3)_1$ or $(z_1, z_2, z_3)_2$, respectively.

To see that 3-SAT* is NP-hard, consider an instance of 1-in-3-SAT with n variables $x_1, \dots, x_n$ and m clauses. We now construct an equivalent 3-SAT* instance. Let d_i be the number of times the variable x_i occurs in the given 1-in-3-SAT formula. For each variable x_i, we introduce new variables $x_{i,1}, \dots, x_{i,d_i}$ and $y_{i,1}, \dots, y_{i,d_i}$ along with clauses $(x_{i,1}, \neg x_{i,2}, y_{i,1})_2, \dots, (x_{i,d_i-1}, \neg x_{i,d_i}, y_{i,d_i-1})_2$, $(x_{i,d_i}, \neg x_{i,1}, y_{i,d_i})_2$ and clauses $(y_{i,j}, \neg y_{i,j}, \neg y_{i,j})_1$ for each $j \in [d_i]$. Note that each variable $y_{i,j}$ has to take the value $\top$ in a satisfying assignment, due to the clause $(y_{i,j}, \neg y_{i,j}, \neg y_{i,j})_1$. The remaining clauses ensure, that for each i the variables $x_{i,1}, \dots x_{i,d_i}$ have the same value in a satisfying assignment. Furthermore, for each of the clauses of the original problem, we introduce one 1-in-3-clause and one 2-in-3-clause. The 1-in-3-clauses are obtained by exchanging the j-th occurrence of each variable x_i with $x_{i,j}$.

Table 1 The sizes and sets of eligible machines of the jobs in the simple reduction. The entry for $\texttt{CMach}_{i,s}$ marks the private load of the machine. The target makespan is given by $T = 322$.

Job	Size	Eligible Machines
$\texttt{CMach}_{i,s}$	111	$\texttt{CMach}_{i,s}$
$\texttt{CJob}^{\top}_{i,s'}$	100	$\texttt{CMach}_{i,1}$, $\texttt{CMach}_{i,2}$, $\texttt{CMach}_{i,3}$
$\texttt{CJob}^{\bot}_{i,s'}$	101	$\texttt{CMach}_{i,1}$, $\texttt{CMach}_{i,2}$, $\texttt{CMach}_{i,3}$
$\texttt{TJob}^{\top}_{j}$	100	$\texttt{TMach}_{j,1}$, $\texttt{TMach}_{j,2}$
$\texttt{TJob}^{\bot}_{j}$	102	$\texttt{TMach}_{j,1}$, $\texttt{TMach}_{j,2}$
$\texttt{VJob}^{\top}_{j,t}$	111	$\texttt{TMach}_{j,\lceil t/2 \rceil}$, $\texttt{CMach}_{\kappa(j,t)}$
$\texttt{VJob}^{\bot}_{j,t}$	110	$\texttt{TMach}_{j,\lceil t/2 \rceil}$, $\texttt{CMach}_{\kappa(j,t)}$

Moreover, the 2-in-3-clauses are obtained by copying the new 1-in-3-clauses, negating all the literals and turning them into a 2-in-3-clause. Hence, each 2-in-3-clauses evaluates to $\top$, if and only if its corresponding 1-in-3-clause does. It is not hard to verify the correctness of the reduction. Similar constructions are widely used, see, e.g., [31] or [5]. The remarkable aspect of the present construction lies in its symmetrical structure which helps to avoid additional dummy gadgets in the following reductions.

Simple Reduction. In the following, we assume that an instance of 3-SAT* with m 1-in-3-clauses $C_1, \dots, C_m$, m 2-in-3-clauses $C_{m+1}, \dots, C_{2m}$ and n variables $x_1, \dots, x_n$ is given. Note that we have $2m$ clauses with 3 literals each, and $4n$ occurring literals in total, hence $3m = 2n$. In addition to the ordering of the variables and clauses, we fix an ordering of the literals belonging to each clause, and an ordering of the occurrences of each variable by assigning an index $t \in [4]$ to each of them. In particular, for each variable x_j, $t = 1, 2$ correspond to the first and second positive and $t = 3, 4$ to the first and second negative occurrence of x_j. Furthermore, let $\kappa : [n] \times [4] \to [2m] \times [3]$ be the bijection defined as follows: $\kappa(j, t) = (i, s)$ implies that the t-th occurrence of x_j is positioned in clause C_i on position s.

We now define the restricted assignment instance. For some of the machines, we introduce *private loads* which is a synonym for jobs of the corresponding size that have to be scheduled on the respective machine because its the only eligible one. The sizes and sets of eligible machines of the introduced jobs are presented in Table 1 and the target makespan is given by $T = 322$.

- For each clause C_i, there are three *clause machines* $\texttt{CMach}_{i,s}$ with $s \in [3]$ corresponding to its three literals, as well as three *clause jobs* $\texttt{CJob}^{\circ_{s'}}_{i,s'}$ with $s' \in [3]$ and $\circ_{s'} \in \{\top, \bot\}$. We have $\circ_1 = \top$ and $\circ_3 = \bot$, as well as $\circ_2 = \bot$ if C_i is a 1-in-3 clause, and $\circ_2 = \top$ otherwise. Furthermore, each clause machine has a private load of 111.
- For each variable x_j, there are two *truth assignment machines* $\texttt{TMach}_{j,q}$ with $q \in [2]$ corresponding to the positive ($q = 1$) and negative ($q = 2$) literal of x_j, as well as 2 *truth assignment jobs* $\texttt{TJob}^{\circ}_{j}$ with $\circ \in \{\top, \bot\}$.
- For each variable x_j, there are eight *variable jobs* $\texttt{VJob}^{\circ}_{j,t}$ with $t \in [4]$ and $\circ \in \{\top, \bot\}$ corresponding to the two occurrences of the positive ($t \in \{1, 2\}$) and negative ($t \in \{3, 4\}$) literal of x_j.

Counting the different machines and jobs and adding up the job sizes yields:

▷ Claim 3. The overall size of all the jobs is exactly $|\mathcal{M}|T$.

We will show that there is a satisfying truth assignment for the 3-SAT* instance if and only if there is a schedule in which each machine receives jobs with load exactly T.

Table 2 Each set indicates one of the possible job assignments for each machine in a schedule with makespan T.

Machine	Possible Schedules
$\texttt{TMach}_{j,1}$	$\{\texttt{TJob}_j^\top, \texttt{VJob}_{j,1}^\top, \texttt{VJob}_{j,2}^\top\}$, $\{\texttt{TJob}_j^\bot, \texttt{VJob}_{j,1}^\bot, \texttt{VJob}_{j,2}^\bot\}$
$\texttt{TMach}_{j,2}$	$\{\texttt{TJob}_j^\top, \texttt{VJob}_{j,3}^\top, \texttt{VJob}_{j,4}^\top\}$, $\{\texttt{TJob}_j^\bot, \texttt{VJob}_{j,3}^\bot, \texttt{VJob}_{j,4}^\bot\}$
$\texttt{CMach}_{i,s}$ (1-in-3-clause)	$\{\texttt{VJob}_{\kappa^{-1}(i,s)}^\top, \texttt{CJob}_{i,1}^\top\}$, $\{\texttt{VJob}_{\kappa^{-1}(i,s)}^\bot, \texttt{CJob}_{i,2}^\bot\}$, $\{\texttt{VJob}_{\kappa^{-1}(i,s)}^\bot, \texttt{CJob}_{i,3}^\bot\}$
$\texttt{CMach}_{i,s}$ (2-in-3-clause)	$\{\texttt{VJob}_{\kappa^{-1}(i,s)}^\top, \texttt{CJob}_{i,1}^\top\}$, $\{\texttt{VJob}_{\kappa^{-1}(i,s)}^\top, \texttt{CJob}_{i,2}^\top\}$, $\{\texttt{VJob}_{\kappa^{-1}(i,s)}^\bot, \texttt{CJob}_{i,3}^\bot\}$

For any job $\texttt{Job}^\circ$ with $\circ \in \{\top, \bot\}$, we refer to $\circ$ as its *truth configuration* and say that $\texttt{Job}^\circ$ has $\circ$-configuration. The rationale of the reduction is as follows: Each clause machine $\texttt{CMach}_{i,s}$ should receive exactly one variable job corresponding to the literal placed in position s in the clause. The truth configuration of this variable job should correspond to the truth value the variable contributes to the clause. To ensure that the jobs $\texttt{VJob}_{j,t}^\top$ belonging to variable x_j contribute consistent truth values, the truth assignment jobs and machines are introduced. In the following, we sometimes talk about the truth assignment gadget and thus refer to these jobs and machines. Similarly, the clause machines and jobs are sometimes called the clause gadget. Note that the basic approach of using some kind of truth assignment and clause gadget for reductions in the context of restricted assignment has been used before, see, e.g., [7, 4, 5].

Next, we present a sequence of easy claims concerning the properties of a schedule for the above instance with makespan T. Due to Table 1 and Claim 3, we have:

▷ Claim 4. Each machine receives exactly 3 jobs (including private loads).

Since each digit of each occurring size is upper bounded by 2, the above claim implies that there can be no carryover when adding up job sizes of jobs scheduled on each machine. Hence the digits of the numbers involved can be considered independently, e.g., there can be at most two jobs with a 1 in the third (or second) digit of its size scheduled on any machine. This together with the given job restrictions already implies:

▷ Claim 5. Each truth assignment machine receives exactly one truth assignment and two variable jobs; and each clause machine receives exactly one clause and one variable job.

▷ Claim 6. The jobs scheduled on a truth assignment or clause machine all have the same truth configuration (excluding private loads).

▷ Claim 7. Let $j \in [n]$. The truth configuration of any job scheduled on $\texttt{TMach}_{j,1}$ is distinct from the truth configuration of any job scheduled on $\texttt{TMach}_{j,2}$.

The resulting possible schedules for each machine are summed up in Table 2, and Figure 1 depicts the resulting two possible schedules for each pair of truth assignment machines. Lastly, we have:

▷ Claim 8. For each $i \in [2m]$, the three clause machines corresponding to i receive exactly one variable job with $\top$-configuration if C_i is a 1-in-3-clause and exactly two such jobs if C_i is a 2-in-3-clause.

Using the above claims, we can easily show:

▶ **Proposition 9.** *There is a satisfying truth assignment for the given* 3-SAT* *instance if and only if there is a schedule with makespan T for the described restricted assignment instance.*

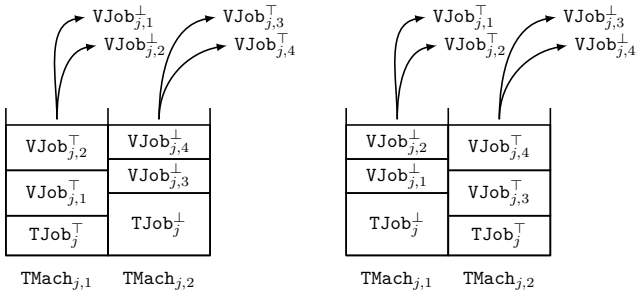


Figure 1 The truth assignment gadget: There are two possible schedules of the truth assignment machines $\texttt{TMach}_{j,1}$ and $\texttt{TMach}_{j,2}$ that already determine the schedule of the variable jobs.

Refined Reduction. When trying to adapt the above reduction to the more restricted problem of RAI, we obviously have less leeway defining the restrictions. To deal with this, we introduce additional gadgets and encode much more information into the job sizes. The idea of the reduction can be described as follows. We arrange the truth assignment gadgets on the left and the clause gadgets on the right. Consider the case that a truth assignment decision is made in the left most truth assignment gadget. Information about this decision – called signal in the following – has to be passed on to the proper clause gadgets passing multiple other truth assignment and clause gadgets on the way. This signal in the simple reduction simply corresponds to a variable job that is to be scheduled on its corresponding clause machine, and in order to prevent interaction with other gadgets, we could encode information about the corresponding variable into the size of the variable job. However, this would lead to a super constant number of job sizes. To avoid this, we introduce a new gadget called the bridge and highway gadget. Very roughly speaking, the signal is passed on to the *highway* via *gateways*; the highway passes each following truth assignment gadget using *bridges* and carries the signal to the proper clauses. Next, we give a detailed description and analysis of the refined reduction.

We adopt all the machines and jobs introduced in the simple reduction, but change the sizes and sets of eligible machines and introduce additional jobs and machines as well as private loads for *every* machine. We introduce the following jobs and machines:

- For each $j \in [n]$ and $t \in [4]$, we introduce one *gateway machine* $\texttt{GMach}_{j,t}$.
- For each $j \in [n]$, $t \in [4]$ and $j' \in \{j+1, \dots n\}$, we introduce two *bridge machines* $\texttt{BMachIn}_{j,t,j'}$ and $\texttt{BMachOut}_{j,t,j'}$. Furthermore, we introduce two *bridge jobs* $\texttt{BJob}^{\top}_{j,t,j'}$ and $\texttt{BJob}^{\perp}_{j,t,j'}$.
- For each $j \in [n]$, $t \in [4]$ and $j' \in \{j, \dots n\}$, we introduce two *highway jobs* $\texttt{HJob}^{\top}_{j,t,j'}$ and $\texttt{HJob}^{\perp}_{j,t,j'}$.

In order to define the intervals of eligible machines, we first need a total order of the machines. We partition the machines into blocks, define an internal order for each block, and then define an order of the blocks. Remember that $\kappa : [n] \times [4] \to [2m] \times [3]$ is a bijection indicating the positions of the occurrences of variables in the clauses. In particular, $\kappa(j,1) = (i,s)$ indicates that the first positive occurrence of variable x_j is in clause C_i on position s, and $\kappa(j,2)$, $\kappa(j,3)$, and $\kappa(j,4)$ indicate analogue information for the second positve, first negative, and second negative occurrence of x_j.

- For each $j \in [n]$, we have a truth assignment block $\mathcal{T}_j$ containing the truth assignment machines $\texttt{TMach}_{j,1}$ and $\texttt{TMach}_{j,2}$ in this order.
- For each $i \in [2m]$, we have a clause block $\mathcal{C}_i$ containing the clause machines $\texttt{CMach}_{i,s}$ for each $s \in [3]$ and ordered increasingly by s.

- For each $j \in [n]$, we have a successor block $\mathcal{S}_j$ containing the gateway machines $\texttt{GMach}_{j,t}$ for each $t \in [4]$ and the bridge machines $\texttt{BMachOut}_{j',t,j}$ for each $t \in [4]$ and $j' < j$. For each machine, we define an index, namely $\kappa(j,t)$ for $\texttt{GMach}_{j,t}$ and $\kappa(j',t)$ for $\texttt{BMachOut}_{j',t,j}$, and order the machines by the *decreasing* lexicographical ordering of their indices. For example, if $\texttt{BMachOut}_{j_1,t_1,j}, \texttt{BMachOut}_{j_2,t_2,j}, \texttt{GMach}_{j,t_3} \in \mathcal{S}_j$ and $\kappa(j_1,t_1) = (1,2)$, $\kappa(j_2,t_2) = (1,1)$ and $\kappa(j,t_3) = (2,3)$, then $\texttt{GMach}_{j,t_3}$ precedes $\texttt{BMachOut}_{j_1,t_1,j}$ which in turn precedes $\texttt{BMachOut}_{j_2,t_2,j}$.
- For each $j \in [n]$ with $j > 1$, we have a predecessor block $\mathcal{P}_j$ containing the bridge machines $\texttt{BMachIn}_{j',t,j}$ for each $t \in [4]$ and $j' < j$. Machine $\texttt{BMachIn}_{j',t,j}$ has index $\kappa(j',t)$ and the machines are ordered by the *increasing* lexicographical ordering of their indices.

The blocks are ordered as follows:

$$(\mathcal{T}_1, \mathcal{S}_1, \mathcal{P}_2, \mathcal{T}_2, \mathcal{S}_2, \ldots, \mathcal{P}_n, \mathcal{T}_n, \mathcal{S}_n, \mathcal{C}_1, \ldots, \mathcal{C}_{2m})$$

The sets of eligible machines are specified in Table 3, and in Table 4 the job sizes as well as the target makespan T are given[3]. Figure 2 gives some intuition on the overall structure. Due to counting and basic arithmetic, we have:

▷ Claim 10. The overall size of the jobs is exactly $|\mathcal{M}|T$.

Like for the simple reduction, we prove a sequence of easy claims concerning the properties of a schedule for the constructed instance with makespan T. First note:

▷ Claim 11. Each machine receives exactly 4 jobs if it is a truth assignment machine and exactly 3 jobs otherwise (including private loads).

Since each machine receives at most 4 jobs and each digit in the job sizes is bounded by 2, we may consider each digit of the involved numbers independently, e.g., if two jobs and the makespan have a 1 at the ℓ-th digit, we already know that these jobs cannot be scheduled on the same machine. This already implies a series of claims:

▷ Claim 12. The jobs $\texttt{TJob}_j^{\top}$ and $\texttt{TJob}_j^{\bot}$ can exclusively be scheduled on $\texttt{TMach}_{j,1}$ and $\texttt{TMach}_{j,2}$, for each $j \in [n]$, and each of the two machines receives exactly one of the two jobs.

▷ Claim 13. The jobs $\texttt{VJob}_{j,t}^{\top}$ and $\texttt{VJob}_{j,t}^{\bot}$ can exclusively be scheduled on $\texttt{TMach}_{j,\lceil t/2 \rceil}$ and $\texttt{GMach}_{j,t}$, for each $j \in [n]$ and $t \in [4]$, and each of the two machines receives exactly one of the two jobs.

▷ Claim 14. Bridge jobs can exclusively be scheduled on bridge machines and each bridge machine receives exactly one bridge job.

▷ Claim 15. Highway jobs can exclusively be scheduled on bridge, gateway and clause machines and each such machine receives exactly one highway job.

▷ Claim 16. Each clause machine $\texttt{CMach}_{i,s}$ receives exactly one of the corresponding clause jobs $\texttt{CJob}_{i,s'}^{\circ_{s'}}$ with $s' \in [3]$.

At this point, we already know that variable (and truth assignment) jobs can exclusively be scheduled on the first or last machine of their respective interval of eligible machines. The next step is to show that the same holds for highway and bridge jobs. To do so, the ordering of the bridge and highway machines is of critical importance.

[3] Note that we have prioritized comprehensibility over small sizes. For instance, it is not hard to see that the columns in Table 4 corresponding to the highway and clause jobs could be deleted and the reduction would still work.

Table 3 The sets of eligible machines for each job or job type, defined by the first and last eligible machine in the ordering. Note that in case of the highway jobs all four combinations of first and last machine are possible.

Job	First machine	Last machine
Clause job $\texttt{CJob}^{\circ s}_{i,s}$	$\texttt{CMach}_{i,1}$	$\texttt{CMach}_{i,3}$
Truth assignment job $\texttt{TJob}^{\circ}_{j}$	$\texttt{TMach}_{j,1}$	$\texttt{TMach}_{j,2}$
Variable job $\texttt{VJob}^{\circ}_{j,t}$	$\texttt{TMach}_{j,\lceil t/2 \rceil}$	$\texttt{GMach}_{j,t}$
Bridge job $\texttt{BJob}^{\circ}_{j,t,j'}$	$\texttt{BMachIn}_{j,t,j'}$	$\texttt{BMachOut}_{j,t,j'}$
Highway job $\texttt{HJob}^{\circ}_{j,t,j'}$	$\texttt{BMachOut}_{j,t,j'}$, if $j' > j$, $\texttt{GMach}_{j,t}$, if $j' = j$	$\texttt{BMachIn}_{j,t,j'+1}$ if $j' < n$, $\texttt{CMach}_{\kappa(j,t)}$, if $j' = n$

Table 4 Table of job and machine types with job sizes and private loads and the makespan. The second column states the number of jobs and machines of the respective types. Each horizontal sequence of numbers following the second column indicates the size of the respective job or private load. Each of the corresponding columns serves a function in the reduction: the first bounds the number of jobs on each machines; the following eight implement restrictions for the bridge, highway, clause, truth assignment and variable jobs; and the last encodes truth values.

	#		B	H	C	T	V	V	V	V	
$\texttt{CJob}^{\top}_{i,s}$	$3m = 2n$	1	0	0	1	0	0	0	0	0	0
$\texttt{CJob}^{\bot}_{i,s}$	$3m = 2n$	1	0	0	1	0	0	0	0	0	1
$\texttt{TJob}^{\top}_{j}$	n	1	0	0	0	1	0	0	0	0	2
$\texttt{TJob}^{\bot}_{j}$	n	1	0	0	0	1	0	0	0	0	0
$\texttt{VJob}^{\top}_{j,1}$	n	1	0	0	0	0	1	0	0	0	0
$\texttt{VJob}^{\top}_{j,2}$	n	1	0	0	0	0	0	1	0	0	0
$\texttt{VJob}^{\top}_{j,3}$	n	1	0	0	0	0	0	0	1	0	0
$\texttt{VJob}^{\top}_{j,4}$	n	1	0	0	0	0	0	0	0	1	0
$\texttt{VJob}^{\bot}_{j,1}$	n	1	0	0	0	0	1	0	0	0	1
$\texttt{VJob}^{\bot}_{j,2}$	n	1	0	0	0	0	0	1	0	0	1
$\texttt{VJob}^{\bot}_{j,3}$	n	1	0	0	0	0	0	0	1	0	1
$\texttt{VJob}^{\bot}_{j,4}$	n	1	0	0	0	0	0	0	0	1	1
$\texttt{BJob}^{\top}_{j,t,j'}$	$2n(n-1)$	1	1	0	0	0	0	0	0	0	0
$\texttt{BJob}^{\bot}_{j,t,j'}$	$2n(n-1)$	1	1	0	0	0	0	0	0	0	1
$\texttt{HJob}^{\top}_{j,t,j'}$	$2n(n+1)$	1	0	1	0	0	0	0	0	0	1
$\texttt{HJob}^{\bot}_{j,t,j'}$	$2n(n+1)$	1	0	1	0	0	0	0	0	0	0
$\texttt{CMach}_{i,s}$	$6m = 4n$	1	1	0	0	1	1	1	1	1	1
$\texttt{TMach}_{j,1}$	n	0	1	1	1	0	0	0	1	1	0
$\texttt{TMach}_{j,2}$	n	0	1	1	1	0	1	1	0	0	0
$\texttt{GMach}_{j,1}$	n	1	1	0	1	1	0	1	1	1	1
$\texttt{GMach}_{j,2}$	n	1	1	0	1	1	1	0	1	1	1
$\texttt{GMach}_{j,3}$	n	1	1	0	1	1	1	1	0	1	1
$\texttt{GMach}_{j,4}$	n	1	1	0	1	1	1	1	1	0	1
$\texttt{BMachIn}_{j,t,j'}$	$2n(n-1)$	1	0	0	1	1	1	1	1	1	1
$\texttt{BMachOut}_{j,t,j'}$	$2n(n-1)$	1	0	0	1	1	1	1	1	1	1
Makespan T		3	1	1	1	1	1	1	1	1	2

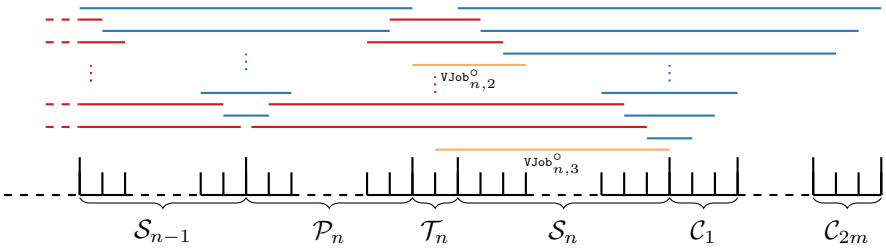


Figure 2 The bridge and highway gadget. The intervals of eligible machines of highway, bridge and variable jobs are depicted in blue, red and orange, respectively. In this example, variable x_n occurs for the second time in its positive form in the last clause at the first position, and for the first time in its negative form in the first clause at the first position.

▷ Claim 17. The jobs $\texttt{BJob}^{\top}_{j,t,j'}$ and $\texttt{BJob}^{\bot}_{j,t,j'}$ can exclusively be scheduled on $\texttt{BMachIn}_{j,t,j'}$ and $\texttt{BMachOut}_{j,t,j'}$, for each $j \in [n]$, $j' \in \{j+1,\dots,n\}$ and $t \in [4]$, and each of the two machines receives exactly one of the two jobs.

Proof. The claim can be proved with a simple inductive argument: Let $j' \in \{2,\dots,n\}$ and, furthermore, $(j_\ell, t_\ell) \in [j'-1] \times [4]$ denote the ℓ-th element from $[j'-1] \times [4]$ when ordering the pairs $(j,t) \in [j'-1] \times [4]$ by the increasing lexicographical ordering of the pairs $\kappa(j,t)$. Considering the ordering of the machines and the job restrictions, $\texttt{BJob}^{\top}_{j_1,t_1,j'}$ and $\texttt{BJob}^{\bot}_{j_1,t_1,j'}$ are the only bridge jobs that can be scheduled on $\texttt{BMachIn}_{j_1,t_1,j'}$ and $\texttt{BMachOut}_{j_1,t_1,j'}$ (see Figure 2). Hence, the claim has to hold for (j_1,t_1). But then again $\texttt{BJob}^{\top}_{j_2,t_2,j'}$ and $\texttt{BJob}^{\bot}_{j_2,t_2,j'}$ are the only remaining bridge jobs that can be scheduled on $\texttt{BMachIn}_{j_2,t_2,j'}$ and $\texttt{BMachOut}_{j_2,t_2,j'}$, and so on. ◁

▷ Claim 18. The jobs $\texttt{HJob}^{\top}_{j,t,j'}$ and $\texttt{HJob}^{\bot}_{j,t,j'}$ can exclusively be scheduled on machine $\texttt{X}$ and $\texttt{Y}$, for each $j \in [n]$, $j' \geq j$, and $t \in [4]$; where $\texttt{X} = \texttt{BMachOut}_{j,t,j'}$ if $j' > j$, and $\texttt{X} = \texttt{GMach}_{j,t}$ otherwise, and $\texttt{Y} = \texttt{BMachIn}_{j,t,j'+1}$ if $j' < n$, and $\texttt{Y} = \texttt{CMach}_{\kappa(j,t)}$ otherwise. Furthermore, each of the two machines receives exactly one of the two jobs.

Proof. We can use the same argument (with reversed orderings) as we did in the last claim. It is only slightly more complicated, because more machine types are involved. ◁

Summing up, each job except for clause jobs may only be scheduled on the first or last machine of their interval of eligible machines, and each of these machines receives either the respective job in $\top$- or $\bot$-configuration. Considering this distribution of the jobs and the last digit of the size vectors, we get the following two claims:

▷ Claim 19. For any machine, the jobs assigned to this machine all have the same truth configuration (excluding private loads).

▷ Claim 20. For each $i \in [2m]$, the three clause machines corresponding to i receive exactly one highway job with $\top$-configuration, if C_i is a 1-in-3-clause, and exactly two such jobs, if C_i is a 2-in-3-clause.

The former property together with the possible job distribution determined so far implies that there are only few possible schedules for each machine. We summarize these schedules in Table 5. Furthermore, we can infer that the truth assignment gadget works essentially the same as before (see Figure 1):

▷ Claim 21. Let $j \in [n]$. The truth configuration of any job scheduled on $\texttt{TMach}_{j,1}$ is distinct from the truth configuration of any job scheduled on $\texttt{TMach}_{j,2}$.

Table 5 For each machine there are only few possible jobs that may be assigned to it in a schedule with makespan T. Each set corresponds to one of the possible schedules.

Machine	Possible Schedule
$\texttt{TMach}_{j,1}$	$\{\texttt{TJob}_j^{\top}, \texttt{VJob}_{j,1}^{\top}, \texttt{VJob}_{j,2}^{\top}\}$, $\{\texttt{TJob}_j^{\bot}, \texttt{VJob}_{j,1}^{\bot}, \texttt{VJob}_{j,2}^{\bot}\}$
$\texttt{TMach}_{j,2}$	$\{\texttt{TJob}_j^{\top}, \texttt{VJob}_{j,3}^{\top}, \texttt{VJob}_{j,4}^{\top}\}$, $\{\texttt{TJob}_j^{\bot}, \texttt{VJob}_{j,3}^{\bot}, \texttt{VJob}_{j,4}^{\bot}\}$
$\texttt{GMach}_{j,t}$	$\{\texttt{VJob}_{j,t}^{\top}, \texttt{HJob}_{j,t,j}^{\top}\}$, $\{\texttt{VJob}_{j,t}^{\bot}, \texttt{HJob}_{j,t,j}^{\bot}\}$
$\texttt{BMachIn}_{j,t,j'}$	$\{\texttt{BJob}_{j,t,j'}^{\top}, \texttt{HJob}_{j,t,j'-1}^{\top}\}$, $\{\texttt{BJob}_{j,t,j'}^{\bot}, \texttt{HJob}_{j,t,j'-1}^{\bot}\}$
$\texttt{BMachOut}_{j,t,j'}$	$\{\texttt{BJob}_{j,t,j'}^{\top}, \texttt{HJob}_{j,t,j'}^{\top}\}$, $\{\texttt{BJob}_{j,t,j'}^{\bot}, \texttt{HJob}_{j,t,j'}^{\bot}\}$
$\texttt{CMach}_{i,s}$ (1-in-3)	$\{\texttt{CJob}_{i,1}^{\top}, \texttt{HJob}_{\kappa^{-1}(i,s),n}^{\top}\}$, $\{\texttt{CJob}_{i,2}^{\bot}, \texttt{HJob}_{\kappa^{-1}(i,s),n}^{\bot}\}$,$\{\texttt{CJob}_{i,3}^{\bot}, \texttt{HJob}_{\kappa^{-1}(i,s),n}^{\bot}\}$
$\texttt{CMach}_{i,s}$ (2-in-3)	$\{\texttt{CJob}_{i,1}^{\top}, \texttt{HJob}_{\kappa^{-1}(i,s),n}^{\top}\}$, $\{\texttt{CJob}_{i,2}^{\top}, \texttt{HJob}_{\kappa^{-1}(i,s),n}^{\top}\}$,$\{\texttt{CJob}_{i,3}^{\bot}, \texttt{HJob}_{\kappa^{-1}(i,s),n}^{\bot}\}$

Lastly, we can show that the bridge and highway gadget works as well:

▷ Claim 22. Let $j \in [n]$ and $t \in [4]$. The variable job scheduled on $\texttt{TMach}_{j,\lceil t/2 \rceil}$ and the highway job scheduled on $\texttt{CMach}_{\kappa(j,t)}$ have the same truth configuration.

Proof. Note that the truth configuration of the variable job scheduled on $\texttt{GMach}_{j,t}$ compared with the one of the variable job scheduled on $\texttt{TMach}_{j,\lceil t/2 \rceil}$ is reversed. Hence, the highway job scheduled on $\texttt{GMach}_{j,t}$ also has the reversed truth-configuration while the highway job that is passed on again has the original truth-configuration. This argument can be repeated with the bridge and highway jobs in the following, yielding the asserted claim. ◁

Using the above claims, we can conclude the proof of Theorem 1 via the following Lemma:

▶ **Lemma 23.** *There is a satisfying truth assignment for the given* 3-SAT* *instance, if and only if there is a schedule with makespan T for the constructed* RAI *instance.*

Proof. First, we consider the case that a schedule with makespan T for the constructed RAI instance is given. For each variable x_j and occurrence $t \in [4]$, let $\texttt{HJob}_{j,t,n}^{\circ_{j,t}}$ be the highway job scheduled on $\texttt{CMach}_{\kappa(j,t)}$ (see Table 5). We choose the truth value of x_j to be $\circ_{j,1}$. Considering the distribution of jobs on the truth assignment machines (see Table 5), as well as Claim 21 and 22, we know that for each variable x_j and occurrence $t \in [4]$, the truth configuration $\circ_{j,t}$ corresponds exactly to the truth value x_j contributes to the clause given by $\kappa(j,t)$. Furthermore, we know that for each clause C_i, there are exactly three variable jobs scheduled on the corresponding clause machines, and exactly one or two of these has $\top$-configuration, if C_i is a 1-in-3-clause or 2-in-3-clause, respectively (Claim 20). Hence, C_i is satisfied.

Now, let there be a satisfying truth assignment, and $\triangleleft_j$ be the corresponding truth value of variable x_j and $\triangleright_j$ its negation. We set $\triangle_{j,t} = \triangleleft_j$ for $t \in \{1,2\}$ and $\triangle_{j,t} = \triangleright_j$ for $t \in \{3,4\}$ and assign $\texttt{HJob}_{j,t,n}^{\triangle_{j,t}}$ to $\texttt{CMach}_{\kappa(j,t)}$. Let $\triangledown_{jt}$ be the negation of $\triangle_{jt}$. All the other jobs are assigned as indicated by the claims and Table 5 in particular: Each machine receives its private load; $\texttt{CMach}_{\kappa(j,t)}$ additionally receives one of the eligible remaining clause jobs with $\triangle_{j,t}$-configuration (this can be done because the truth assignment is satisfying); $\texttt{BMachOut}_{j,t,j'}$ receives $\texttt{HJob}_{j,t,j'}^{\triangledown_{j,t}}$ and $\texttt{BJob}_{j,t,j'}^{\triangledown_{j,t}}$; $\texttt{BMachIn}_{j,t,j'}$ receives $\texttt{HJob}_{j,t,j'-1}^{\triangle_{j,t}}$ and $\texttt{BJob}_{j,t,j'}^{\triangle_{j,t}}$; $\texttt{GMach}_{j,t}$ receives $\texttt{HJob}_{j,t,j}^{\triangledown_{j,t}}$ and $\texttt{VJob}_{j,t}^{\triangledown_{j,t}}$; $\texttt{TMach}_{j,1}$ receives $\texttt{VJob}_{j,1}^{\triangle_{j,1}}$, $\texttt{VJob}_{j,2}^{\triangle_{j,2}}$ and $\texttt{TJob}_{j}^{\triangle_{j,1}}$; and $\texttt{TMach}_{j,2}$ receives $\texttt{VJob}_{j,3}^{\triangle_{j,3}}$, $\texttt{VJob}_{j,4}^{\triangle_{j,4}}$ as well as $\texttt{TJob}_{j}^{\triangle_{j,3}}$. It is easy to verify, that all jobs are assigned and each machine has a load of T. ◀

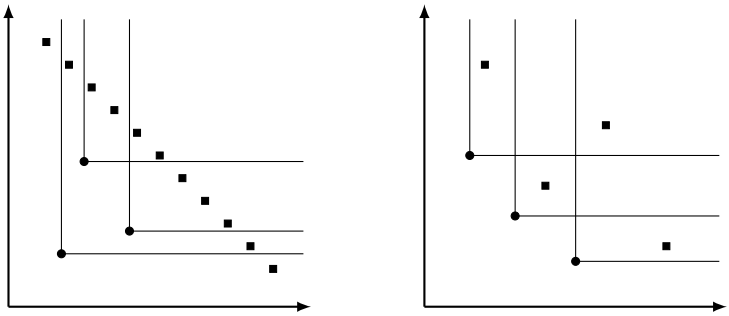


Figure 3 The left picture visualizes that each RAI instance can be seen as a RAR(2) instance and the right one depicts an RAR(2) instance that is not a RAI instance. In both pictures, each dimension corresponds to a resource, the squares mark the capacities of machines and the circles the demands of jobs. If the capacity of a machine is at least as big as the demand of a job in both dimension, the job is eligible on the machine.

3 Resource Restrictions

In this section, we briefly discuss scheduling with resource restrictions and provide the proof of Theorem 2 in particular. First note that RAI is properly placed between RAR(1) and RAR(2), that is, with a slight abuse of notation, RAR(1) $\subset$ RAI $\subset$ RAR(2). The ideas needed to see RAI $\subset$ RAR(2) are given in Figure 3.

The result in Theorem 2 concerning 4 resources is proven by showing that the restrictions in a reduction due to Ebenlendr et al. [7] can be modeled using 4 resources. Concerning the result for 2 resources, we first discuss the corresponding result for 3 resources. The reduction is based on the classical result by Lenstra et al. [19] and very similar to a reduction by Bhaskara et al. [1] for rank four unrelated scheduling. However, there is a problem with the choice of processing times in the latter reduction (see [22]), and the present result can be used to fix it. Combining the ideas in that reduction with a result by Chen et al. [6] yields the result for 2 resources.

Four Resources. In the classical 3-SAT problem, a conjunction of m clauses is given and each clause is a disjunction of at most three literals of variables $x_1, \ldots, x_n$. In the result due to Ebenlendr et al. [7], the modified 3-SAT problem, where each variable occurs exactly three and each literal at most two times in the formula, is reduced to the graph balancing problem, that is, restricted assignment with the additional property that each job is eligible on at most two machines. To show that the modified 3-SAT problem is NP-hard, we can use techniques already applied in Section 2: We may replace the d_j occurrences of variable x_j with new variables $z_{j1}, \ldots, z_{jd_j}$ and add new clauses $(z_{j1} \wedge \neg z_{j2}), \ldots (z_{jd_{j-1}} \wedge \neg z_{jd_j}), (z_{jd_j} \wedge \neg z_{j1})$.

▶ **Theorem 24** ([7]). *There is no polynomial time approximation algorithm with rate smaller than* 1.5 *for the graph balancing problem unless P=NP.*

Proof. Given an instance of modified 3-SAT, we introduce clause machines v_i corresponding to the clauses C_i, and literal machines $u_{j,1}$ and $u_{j,0}$ corresponding to the literals x_j and $\neg x_j$. Furthermore, we introduce truth assignment jobs e_j for each variable x_j with size 2 and eligible on $u_{j,1}$ and $u_{j,0}$; and clause jobs $f_{i,j,\alpha}$ for each clause C_i and literal y_j occurring in C_i with $\alpha = 1$ if $y_j = x_j$ and $\alpha = 0$ if $y_j = \neg x_j$. The job $f_{i,j,\alpha}$ has size 1 and is eligible on v_i and $u_{j,\alpha}$. Lastly, we introduce a dummy job d_i for each clause C_i with less then three literals. Its size is 1 if C_i contains two literals, and 2 if C_i contains only one literal.

In a schedule with makespan 2, there is at least one clause job $f_{i,j,\alpha}$ for each v_i that is scheduled on $u_{j,\alpha}$ and not on v_i. Hence, the job e_j has to be scheduled on $u_{j,|\alpha-1|}$. Now, it is easy to see that there is a schedule with makespan 2, if and only if there is a satisfying assignment. The construction works as follows: Given a schedule with makespan 2, we set variable x_j to $\top$ if e_j is scheduled on $u_{j,0}$, and to $\bot$ otherwise. Moreover, given a satisfying truth assignment we assign the truth assignment jobs correspondingly, and the machines $u_{j,\alpha}$ that did not receive a truth assignment job receives all eligible clause jobs (at most two). ◀

We reproduce the restrictions in the above reduction using four resources and get:

▶ **Corollary 25.** *There is no polynomial time approximation algorithm for* RAR(4) *with rate smaller than* 1.5 *unless P=NP.*

Proof. We set $\mathcal{R} = [4]$. The clause machine v_i has a resource capacity vector of $(2n + 1, 2n + 1, i, (m + 1) - i)$, and the literal machine $u_{j,\alpha}$ has capacity vectors $(2j - \alpha, (2n + 1) - (2j - \alpha), m + 1, m + 1)$. Furthermore, the truth assignment job e_j has a resource demand vector of $(2j - 1, (2n + 1) - 2j, m + 1, m + 1)$; the clause job $f_{i,j,\alpha}$ has a demand vector of $(2j - \alpha, (2n + 1) - (2j - \alpha), i, (m + 1) - i)$; and the dummy job d_i has a demand vector of $(2n + 1, 2n + 1, i, (m + 1) - i)$. It is easy to verify that the resulting sets of eligible machines are the same as described in Theorem 24. ◀

Three Resources. In the 3-DM problem, the input consists of three disjoint sets A, B and C with $|A| = |B| = |C| = n \in \mathbb{N}$, as well as a set of triplets $E \subseteq \big\{\{a, b, c\} \,\big|\, a \in A, b \in B, c \in C\big\}$. The goal is to decide whether there is a subset $F \subseteq E$ that perfectly covers A, B and C, that is, for each $x \in A \cup B \cup C$ there is exactly one triplet $e \in F$ with $x \in e$. The set F is called a 3D-matching. We assume that the elements of A, B and C are indexed, that is, $A = \{a_1, a_2, \ldots, a_n\}$, $B = \{b_1, b_2, \ldots, b_n\}$ and $C = \{c_1, c_2, \ldots, c_n\}$. Furthermore, we assume that for each $x \in A \cup B \cup C$ there is at least one $e \in E$ with $x \in E$.

We present a reduction from 3-DM to RAR(3). Given an instance (A, B, C, E) of 3-DM, let $E(x) = \{e \in E \,|\, x \in e\}$ for each $x \in A \cup B \cup C$. Furthermore, we set $\alpha_A = 12$, $\alpha_B = 13$, $\alpha_C = 22$, $\beta_A = 14$, $\beta_B = 15$ and $\beta_C = 18$. Let $\mathcal{R} = \{A, B, C\}$ and $\mathcal{M} = E$. For each machine e, we define the resource capacities as follows. Let $X \in \{A, B, C\}$ and $x_i \in X \cap e$ be the element of x with index i. We set $c_X(e) = i$. Furthermore, for each element $x_i \in X$ with index i in $X \in \{A, B, C\}$, we introduce one element job with size α_X and $|E(x)| - 1$ dummy jobs with size β_X. The resource demand for each of these jobs is given by $d(i)$ with $d_X(i) = i$ and $d_Y(i) = 0$ for $Y \in \{A, B, C\} \setminus \{X\}$.

▷ Claim 26. We have $\alpha_A + \alpha_B + \alpha_C = 47 = \beta_A + \beta_B + \beta_C$; any four numbers taken from $\Gamma = \{\alpha_A, \alpha_B, \alpha_C, \beta_A, \beta_B, \beta_C\} = \{12, 13, 22, 14, 15, 18\}$ sum up to a value bigger than 47; any selection of less than 3 numbers sums up to a value smaller than 47; and for any three numbers $\gamma_1, \gamma_2, \gamma_3 \in \Gamma$ with $\gamma_1 \leq \gamma_2 \leq \gamma_3$ and $\gamma_1 + \gamma_2 + \gamma_3 = 47$, we have either $(\gamma_1, \gamma_2, \gamma_3) = (\alpha_A, \alpha_B, \alpha_C)$ or $(\gamma_1, \gamma_2, \gamma_3) = (\beta_A, \beta_B, \beta_C)$.

Proof. The first three assertions are obvious, and the fourth holds due to a simple case analysis:

- If $\gamma_1 > 15$, we have $\gamma_1 \geq 18$, and hence $47 = \gamma_1 + \gamma_2 + \gamma_3 \geq 3 \cdot \gamma_1 = 54$: a contradiction.
- Note that $\gamma_3 \geq (\gamma_2 + \gamma_3)/2 = (47 - \gamma_1)/2$. Hence, $\gamma_1 \leq 15$ implies $\gamma_3 \geq 16$ and therefore $\gamma_3 \in \{18, 22\}$.
- If we have $\gamma_3 = 22 = \alpha_C$, then $\gamma_1 \leq (\gamma_1+\gamma_2)/2 = (47-\gamma_3)/2 = 12.5$. Hence, $\gamma_1 = 12 = \alpha_A$ and $\gamma_2 = 13 = \alpha_B$.

- If we have $\gamma_3 = 18 = \beta_C$, than $\gamma_2 \geq (\gamma_1+\gamma_2)/2 = (47-\gamma_3)/2 = 14.5$. Hence, $\gamma_2 \in \{15, 18\}$. If $\gamma_2 = 15 = \beta_B$, then $\gamma_1 = 14 = \beta_A$, and if $\gamma_2 = 18$, then $\gamma_1 = 11 \notin \Gamma$.

This concludes the proof of the claim. ◁

By brute force, it can be verified that 47 is the smallest value such that suitable numbers α_A, α_B, α_C, β_A, β_B and β_C exist and the above claim holds.

▷ Claim 27. The summed up size of all the element and dummy jobs is $47|\mathcal{M}|$.

Proof. We have exactly n element jobs with size α_A, α_B and α_C, respectively, yielding an overall load of $47n$. The dummy jobs have an overall load of:

$$\begin{aligned}&\beta_A \sum_{a\in A}(|E(a)|-1) + \beta_B \sum_{b\in B}(|E(b)|-1) + \beta_C \sum_{c\in C}(|E(b)|-1)\\ =&(\beta_A+\beta_B+\beta_C)(|E|-n) = 47(|\mathcal{M}|-n)\end{aligned}$$

In this equation, we used the simple fact that $\{E(x) \mid x \in X\}$ is a partition of E for each $X \in \{A, B, C\}$, and hence $|E| = \sum_{x\in X} |E(x)|$. ◁

These two claims imply:

▷ Claim 28. In any schedule for the constructed instance with makespan 47, each machine receives exactly three jobs with sizes $\gamma_1, \gamma_2, \gamma_3$ such that $(\gamma_1, \gamma_2, \gamma_3) = (\alpha_A, \alpha_B, \alpha_C)$ or $(\gamma_1, \gamma_2, \gamma_3) = (\beta_A, \beta_B, \beta_C)$.

Using these claims, we can show:

▶ **Proposition 29.** *There is a perfect matching for the given* 3-DM *instance, if and only if there is a schedule with makespan* 47 *for the constructed* RAR(3) *instance.*

Proof. Let F be a perfect matching for the 3-DM instance. For each $x \in A \cup B \cup C$ we assign the corresponding element job to the machine e with $x \in e$ and $e \in F$. Furthermore, the dummy jobs corresponding to $x \in X$ with $X \in \{A, B, C\}$, are distributed to the machines e with $x \in e$ and $e \notin F$ such that each machine receives exactly one job in this step. Hence, each machine $e \in E$ receives exactly three eligible jobs either with sizes α_A, α_B and α_C (if $e \in F$) or β_A, β_B and β_C (otherwise).

Next, we assume that there is a schedule with makespan 47 for the scheduling instance. For each $X \in \{A, B, C\}$, there are exactly $|\mathcal{M}|$ many jobs with size α_X or β_X, and due to the above claims, we know that each machine receives exactly one of these jobs. For each $j \in [n]$, let $x_j \in X$ be the element with index j in $X \in \{A, B, C\}$. The machines $\bigcup_{j=i}^{n} E(x_j)$ are the only machines that may process jobs corresponding to $x_i, \ldots, x_n$ for each $i \in [n]$ and we have exactly $\sum_{j=i}^{n} |E(x_j)|$ many such jobs. Hence, the machines from $E(x_i)$ receive exactly the jobs corresponding to x_i. Now, considering this and Claim 28, we get a perfect matching by selecting the machines that process three element jobs. ◀

Two Resources. We are able to refine the result for three resources to work for two resources as well by using another variant of 3-DM as the starting point of the reduction. The problem 3-DM* was introduced by Chen et al. [6] to get an improved lower bound for the approximation ratio of rank four unrelated scheduling.

In this problem, a set of six disjoint sets $\mathcal{E} = \{A, A', B, B', C, C'\}$ is given. For each $X \in \mathcal{E}$, we have $|X| = 3n$ for some $n \in \mathbb{N}$ and the sets are indexed by $[3n]$, e.g., $A = \{a_1, a_2, \ldots, a_{3n}\}$. Furthermore, there are two sets of triplets $E_1 \subseteq \big\{\{a_i, b_j, c_j\}, \{a'_i, b_j, c_j\} \,\big|\, i \in [3n], j \in [3n]\big\}$

Table 6 The resource demands and capacities for the different job (types) and machines.

Jobs	Resources	Machines	Resources
a_i	$(2i, 0)$	$\{a_i, b_j, c_j\}$	$(2i, 3n + j)$
a'_i	$(2i - 1, 0)$	$\{a'_i, b_j, c_j\}$	$(2i - 1, 3n + j)$
b_j	$(0, 3n + j)$	$\{a_i, b'_i, c'_i\}$	$(2i, i)$
c_j	$(0, 3n + j)$	$\{a'_i, b'_i, c'_{\zeta(i)}\}$	$(2i - 1, \zeta(i))$
b'_i	$(2i - 1, 0)$		
c'_i	$(0, i)$		

and $E_2 = \left\{\{a_i, b'_i, c'_i\}, \{a'_i, b'_i, c'_{\zeta(i)}\} \,\middle|\, i \in [3n]\right\}$ with $\zeta(3k + 1) = 3k + 2$, $\zeta(3k + 2) = 3k + 3$ and $\zeta(3k + 3) = 3k + 1$ for each $k \in \{0, \dots, n - 1\}$. Note that the second set of triplets is determined by the element sets in the input. Similar to the classical 3-DM problem, the goal is to decide whether there is a subset $F \subseteq E_1 \cup E_2$ that perfectly covers the element set, that is, for each $x \in \bigcup_{X \in \mathcal{E}} X$ there is exactly one triplet $e \in F$ with $x \in e$. We assume that for each $x \in \bigcup_{X \in \mathcal{E}} X$ there is at least one $e \in E$ with $x \in E$ (otherwise the problem is trivial).

Let $\alpha_A = \alpha_{A'} = 12$, $\alpha_B = \alpha_{B'} = 13$, $\alpha_C = \alpha_{C'} = 22$, $\beta_A = \beta_{A'} = 14$, $\beta_B = \beta_{B'} = 15$ and $\beta_C = \beta_{C'} = 18$. We set $\mathcal{M} = E_1 \cup E_2$ and $\mathcal{R} = [2]$. The corresponding resource capacity vectors are presented in Table 6. Furthermore, for each element $x \in X$ in $X \in \mathcal{E}$, we introduce one element job with size α_X and $|E(x)| - 1$ dummy jobs with size β_X. The vector of resource demands for each such job is given in Table 6. Note that Claim 26-28 hold for this reduction as well and with the same reasoning. A simple case analysis yields:

▷ Claim 30. For each $x \in \bigcup_{X \in \mathcal{E}} X$, a (dummy or element) job corresponding to x is eligible on each machine e with $x \in e$.

Using these claims, we can conclude the proof of Theorem 2:

▶ **Lemma 31.** *There is a perfect matching for the given* 3-DM* *instance, if and only if there is a schedule with makespan* 47 *for the constructed* RAR(2) *instance.*

Proof. Let F be a perfect matching for the 3-DM* instance. For each $x \in \bigcup_{X \in \mathcal{E}} X$, we assign the corresponding element job to the machine e with $x \in e$ and $e \in F$. Furthermore, the dummy jobs corresponding to $x \in X$ with $X \in \mathcal{E}$, are distributed to the machines e with $x \in e$ and $e \notin F$ such that each such machine receives exactly one job. Hence, each machine $e \in E$ receives exactly three eligible jobs either with sizes α_A, α_B and α_C or β_A, β_B and β_C.

Next, we assume that there is a schedule with makespan 47 for the scheduling instance. There are exactly $|\mathcal{M}|$ many jobs with size $\alpha_A = \alpha_{A'}$ or $\beta_A = \beta_{A'}$ corresponding to elements of $A \cup A'$, and due to Claim 28 we know that each machine receives exactly one of these jobs. The machines corresponding to triplets from $E(a_{3n})$ are the only ones that can process the $|E(a_{3n})|$ jobs corresponding to a_{3n}, and hence each of these machines receives exactly one of these jobs. Now, the machines corresponding to triplets from $E(a'_{3n})$ are the only *remaining* ones that can process the $|E(a'_{3n})|$ jobs corresponding to a'_{3n}. Iterating this argument, we get that each machine e receives exactly one job corresponding to some $x \in A \cup A'$ with $x \in e$. Note that the above argument was based on the first resource value. Considering the second resource value yields the same result for each $x \in C \cup C'$. For the elements $x \in B \cup B'$ both resource values have to be considered, namely the second for $b \in B$ and the first for $b' \in B'$, but the argument stays the same. Summing up, each machine $e = \{x, y, z\}$ receives exactly three jobs corresponding to x, y and z. Now, considering this and Claim 28, we get a perfect matching by selecting the triplets e that processes three element jobs. ◀

4 Conclusion

In this paper we provided hardness of approximation results for scheduling with interval and resource restrictions. We list some possible future research directions:

From the perspective of complexity, tighter hardness results seem plausible. In particular, we have the same inapproximability results for RAR(2) and RAR(3) and it would be interesting to find a better result for RAR(3).

From the algorithmic perspective, it remains open whether any of the studied problems and RAI in particular admits an approximation algorithm with a rate smaller than 2. There have been some results [32, 27] for RAI using promising linear programming relaxations that may be useful in this context. Another possibility is the application of the local search techniques originally used by Svensson [30] for the restricted assignment problem. This approach recently yielded a breakthrough for the graph balancing problem [14].

Finally, while a PTAS for RAR(1) is known [24], it is unclear whether the problem admits a so called *efficient* PTAS with a running time of the form $f(1/\varepsilon)\text{poly}(|I|)$ for some computable function f.

References

1 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, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 937–947, 2013. doi:10.1137/1.9781611973105.67.

2 Andreas Brandstädt and Vadim V. Lozin. On the linear structure and clique-width of bipartite permutation graphs. *Ars Comb.*, 67, 2003.

3 Deeparnab Chakrabarty and Kirankumar Shiragur. Graph balancing with two edge types. *CoRR*, abs/1604.06918, 2016. arXiv:1604.06918.

4 Lin Chen, Klaus Jansen, and Guochuan Zhang. On the optimality of exact and approximation algorithms for scheduling problems. *J. Comput. Syst. Sci.*, 96:1–32, 2018. doi:10.1016/j.jcss.2018.03.005.

5 Lin Chen, Dániel Marx, Deshi Ye, and Guochuan Zhang. Parameterized and approximation results for scheduling with a low rank processing time matrix. In *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, pages 22:1–22:14, 2017. doi:10.4230/LIPIcs.STACS.2017.22.

6 Lin Chen, Deshi Ye, and Guochuan Zhang. An improved lower bound for rank four scheduling. *Oper. Res. Lett.*, 42(5):348–350, 2014. doi:10.1016/j.orl.2014.06.003.

7 Tomás Ebenlendr, Marek Krcál, and Jirí Sgall. Graph balancing: A special case of scheduling unrelated parallel machines. *Algorithmica*, 68(1):62–80, 2014. doi:10.1007/s00453-012-9668-9.

8 Leah Epstein and Asaf Levin. Scheduling with processing set restrictions: Ptas results for several variants. *International Journal of Production Economics*, 133(2):586–595, 2011. doi:10.1016/j.ijpe.2011.04.024.

9 Pinar Heggernes and Dieter Kratsch. Linear-time certifying recognition algorithms and forbidden induced subgraphs. *Nord. J. Comput.*, 14(1-2):87–108, 2007.

10 Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *J. ACM*, 34(1):144–162, 1987. doi:10.1145/7531.7535.

11 Ellis Horowitz and Sartaj Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *J. ACM*, 23(2):317–327, 1976. doi:10.1145/321941.321951.

12 Chien-Chung Huang and Sebastian Ott. A combinatorial approximation algorithm for graph balancing with light hyper edges. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 49:1–49:15, 2016. doi:10.4230/LIPIcs.ESA.2016.49.

13 Klaus Jansen, Marten Maack, and Roberto Solis-Oba. Structural parameters for scheduling with assignment restrictions. In *Algorithms and Complexity - 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*, pages 357–368, 2017. doi:10.1007/978-3-319-57586-5_30.

14 Klaus Jansen and Lars Rohwedder. Local search breaks 1.75 for graph balancing. *CoRR*, abs/1811.00955, 2018. arXiv:1811.00955.

15 Kamyar Khodamoradi. *Algorithms for Scheduling and Routing Problems*. PhD thesis, Simon Fraser University, 2016.

16 Kamyar Khodamoradi, Ramesh Krishnamurti, Arash Rafiey, and Georgios Stamoulis. PTAS for ordered instances of resource allocation problems. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, December 12-14, 2013, Guwahati, India*, pages 461–473, 2013. doi:10.4230/LIPIcs.FSTTCS.2013.461.

17 Kamyar Khodamoradi, Ramesh Krishnamurti, Arash Rafiey, and Georgios Stamoulis. PTAS for ordered instances of resource allocation problems with restrictions on inclusions. *CoRR*, abs/1610.00082, 2016. arXiv:1610.00082.

18 Kangbok Lee, Joseph Y.-T. Leung, and Michael L. Pinedo. Makespan minimization in online scheduling with machine eligibility. *Annals OR*, 204(1):189–222, 2013. doi:10.1007/s10479-012-1271-6.

19 Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46:259–271, 1990. doi:10.1007/BF01585745.

20 Joseph Y-T Leung and Chung-Lun Li. Scheduling with processing set restrictions: A survey. *International Journal of Production Economics*, 116(2):251–262, 2008. doi:10.1016/j.ijpe.2008.09.003.

21 Joseph Y-T Leung and Chung-Lun Li. Scheduling with processing set restrictions: A literature update. *International Journal of Production Economics*, 175:1–11, 2016. doi:10.1016/j.ijpe.2014.09.038.

22 Marten Maack and Klaus Jansen. Inapproximability results for scheduling with interval and resource restrictions. *CoRR*, abs/1907.03526, 2019. URL: http://arxiv.org/abs/1907.03526.

23 Gabriella Muratore, Ulrich M. Schwarz, and Gerhard J. Woeginger. Parallel machine scheduling with nested job assignment restrictions. *Oper. Res. Lett.*, 38(1):47–50, 2010. doi:10.1016/j.orl.2009.09.010.

24 Jinwen Ou, Joseph Y-T Leung, and Chung-Lun Li. Scheduling parallel machines with inclusive processing set restrictions. *Naval Research Logistics (NRL)*, 55(4):328–338, 2008. doi:10.1002/nav.20286.

25 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 216–226, 1978. doi:10.1145/800133.804350.

26 Petra Schuurman and Gerhard J Woeginger. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *Journal of Scheduling*, 2(5):203–213, 1999. doi:10.1002/(SICI)1099-1425(199909/10)2:5<203::AID-JOS26>3.0.CO;2-5.

27 Ulrich M. Schwarz. *Approximation algorithms for scheduling and two-dimensional packing problems*. PhD thesis, University of Kiel, 2010. URL: http://eldiss.uni-kiel.de/macau/receive/dissertation_diss_00005147.

28 Ulrich M. Schwarz. A PTAS for scheduling with tree assignment restrictions. *CoRR*, abs/1009.4529, 2010. arXiv:1009.4529.

29 Georgios Stamoulis. Private communication, 2019.

30 Ola Svensson. Santa claus schedules jobs on unrelated machines. *SIAM J. Comput.*, 41(5):1318–1341, 2012. doi:10.1137/110851201.

31 Craig A. Tovey. A simplified np-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984. doi:10.1016/0166-218X(84)90081-7.

32 Chao Wang and René Sitters. On some special cases of the restricted assignment problem. *Inf. Process. Lett.*, 116(11):723–728, 2016. doi:10.1016/j.ipl.2016.06.007.

33 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. URL: http://www.cambridge.org/de/knowledge/isbn/item5759340/?site_locale=de_DE.

An Automaton Group with PSPACE-Complete Word Problem

Jan Philipp Wächter
Universität Stuttgart, Institut für Formale Methoden der Informatik (FMI), Universitätsstraße 38, 70569 Stuttgart, Germany
jan-philipp.waechter@fmi.uni-stuttgart.de

Armin Weiß
Universität Stuttgart, Institut für Formale Methoden der Informatik (FMI), Universitätsstraße 38, 70569 Stuttgart, Germany
armin.weiss@fmi.uni-stuttgart.de

Abstract

We construct an automaton group with a PSPACE-complete word problem, proving a conjecture due to Steinberg. Additionally, the constructed group has a provably more difficult, namely EXPSPACE-complete, compressed word problem. Our construction directly simulates the computation of a Turing machine in an automaton group and, therefore, seems to be quite versatile. It combines two ideas: the first one is a construction used by D'Angeli, Rodaro and the first author to obtain an inverse automaton semigroup with a PSPACE-complete word problem and the second one is to utilize a construction used by Barrington to simulate circuits of bounded degree and logarithmic depth in the group of even permutations over five elements.

2012 ACM Subject Classification Theory of computation → Transducers

Keywords and phrases automaton group, word problem, PSPACE, compressed word problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.6

Funding *Armin Weiß*: Funded by DFG project DI 435/7-1.

1 Introduction

The word problem is one of Dehn's fundamental algorithmic problems in group theory [10]: given a word over a finite set of generators for a group, decide whether the word represents the identity in the group. While, in general, the word problem is undecidable [18, 7], many classes of groups have a decidable word problem. Among them is the class of automaton groups. In this context, the term *automaton* refers to finite state, letter-to-letter transducers. In such automata, every state q induces a length-preserving, prefix-compatible action on the set of words, where an *input word* u is mapped to the *output word* obtained by reading u starting in q. The group or semigroup generated by the automaton is the closure under composition of the actions of the different states and a (semi)group arising in this way is called an *automaton (semi)group*.

The interest in automaton groups was stirred by the observation that many groups with interesting properties are automaton groups. Most prominently, the class contains the famous Grigorchuk group (which is the first example of a group with sub-exponential but super-polynomial growth and admits other peculiar properties, see [14] for an accessible introduction). There is also a quite extensive study of algorithmic problems in automaton (semi)groups: the conjugacy problem and the isomorphism problem (here the automaton is part of the input) – the other two of Dehn's fundamental problems – are undecidable for automaton groups [23] . For automaton semigroups, the order problem could be proved to be undecidable [12, Corollary 3.14]. Recently, this could be extended to automaton groups

[13] (see also [4]). On the other hand, the undecidability result for the finiteness problem for automaton semigroups [12, Theorem 3.13] could not be lifted to automaton groups so far.

The undecidability results show that the presentation of groups using automata is still quite powerful. Nevertheless, it is not very difficult to see that the word problem for automaton groups is decidable. One possible way is to show an upper bound on the length of an input word on which a state sequence[1] not representing the identity of the group acts non-trivially. In the most general setting, this bound is $|Q|^n$ where Q is the state set of the automaton and n is the length of the state sequence. Another viewpoint is that one can use a non-deterministic guess and check algorithm to solve the word problem. This algorithm uses linear space proving that the word problem for automaton (semi)groups is in PSPACE. This approach seems to be mentioned first by Steinberg [22, Section 3] (see also [9, Proposition 2 and 3]). In some special cases, better algorithms or upper bounds are known: for example, for contracting automaton groups (and this includes the Grigorchuk group), the witness length is bounded logarithmically [17] and the problem, thus, is in LOGSPACE; other examples of classes with better upper bounds or algorithms include automata with polynomial activity [5] or Hanoi Tower groups [6]. On the other hand, Steinberg conjectured that there is an automaton group with a PSPACE-complete word problem [22, Question 5]. As a partial solution to his problem, an inverse automaton semigroup with a PSPACE-complete word problem has been constructed in [9, Proposition 6][2]. In this paper, our aim is to finally prove the conjecture for groups.

In order to do so, we adopt the construction used by D'Angeli, Rodaro and the first author from [9, Proposition 6]. This construction uses a master reduction and directly encodes a Turing machine into an automaton. Already in [9, Proposition 6], it was also used to show that there is an automaton group whose word problem with a rational constraint (which depends on the input) is PSPACE-complete. To get rid of this rational constraint, we apply an idea used by Barrington [3] to transform NC^1-circuits (circuits of bounded fan-in and logarithmic depth) into bounded-width polynomial-size branching programs. Similar ideas predating Barrington have been attributed to Gurevich (see [15]) and given by Mal'cev [16]. Nevertheless, this paper is fully self-contained and no previous knowledge of either [9] or [3] is needed.

In addition, we also investigate the compressed word problem for automaton groups. Here, the (input) state sequence is given as a so-called *straight-line program* (a context-free grammar which generates exactly one word). By uncompressing the input sequence and applying the above mentioned non-deterministic linear-space algorithm, one can see that the compressed word problem can be solved in EXPSPACE. Thus, the more interesting part is to prove that this algorithm cannot be improved significantly: we show that there is an automaton group with an EXPSPACE-hard compressed word problem. This result is interesting because, by taking the direct product, we obtain a group whose (ordinary) word problem is PSPACE-complete and whose compressed word problem is EXPSPACE-complete and, thus, provably more difficult by the space hierarchy theorem [21, Theorem 6] (or e. g. [2, Theorem 4.8]). To the best of our knowledge, this is the first example of a group for which this is possible.

---

[1] In order to avoid ambiguities, we call a word over the states of the automaton a *state sequence*. So, in our case the input for the word problem is a state sequence.

[2] In fact, the semigroup is generated by a partial, invertible automaton. A priori, this seems to be a stronger statement than that the semigroup is inverse and also an automaton semigroup. That is why the cited paper uses the term 'automaton-inverse semigroup'. Only later, it was shown that the two concepts actually coincide [8, Theorem 25].

Explicit previous results on the compressed word problem for automaton groups do not seem to exist. However, it was observed by Gillibert [11] that the proof of [9, Proposition 6] also yields an automaton semigroup with an EXPSPACE-complete compressed word problem in a rather straightforward manner. For the case of groups, it is possible to adapt the construction used by Gillibert to prove the existence of an automaton group with an undecidable order problem [13] slightly to obtain an automaton group with a PSPACE-hard compressed word problem [11].

2 Preliminaries

Words and Alphabets with Involution. We use common notations from formal language theory. In particular, we use Σ^* to denote the set of words over an alphabet Σ including the empty word. If we want to exclude the empty word, we write Σ^+. For any alphabet Q, we define a natural involution between Q and a disjoint copy $Q^{-1} = \{q^{-1} \mid q \in Q\}$ of Q: it maps $q \in Q$ to $q^{-1} \in Q^{-1}$ and vice versa. In particular, we have $(q^{-1})^{-1} = q$. The involution extends naturally to words over $Q \cup Q^{-1}$: for $q_1, \dots, q_n \in Q \cup Q^{-1}$, we set $(q_n \dots q_1)^{-1} = q_1^{-1} \dots q_n^{-1}$. This way, the involution is equivalent to taking the group inverse if Q is a generating set of a group.

Turing Machines and Complexity. We assume the reader to be familiar with basic notions of complexity theory such as configurations for Turing machines, computations and reductions in logarithmic space as well as complete and hard problems for PSPACE and the class EXPSPACE. See [19] or [2] for standard text books on complexity theory. We only consider deterministic, single-tape machines and write their configurations as word $c_1 \dots c_{i-1} p c_i \dots c_n$ where the c_j are symbols from the tape alphabet and p is a state. In this configuration, the machine is in state p and its head is over the symbol c_i.

▶ **Fact 1** (Folklore). *Consider a deterministic Turing machine with state set P and tape alphabet Δ. After a straightforward transformation of the transition function and states, we can assume that the symbol $\gamma_i^{(t+1)}$ at position i of the configuration at time step $t+1$ only depends on the symbols $\gamma_{i-1}^{(t)}, \gamma_i^{(t)}, \gamma_{i+1}^{(t)} \in \Gamma$ at position $i-1$, i and $i+1$ at time step t. Thus, we may always assume that there is a function $\tau : \Gamma^3 \to \Gamma$ with $\Gamma = P \uplus \Delta$ mapping the symbols $\gamma_{i-1}^{(t)}, \gamma_i^{(t)}, \gamma_{i+1}^{(t)} \in \Gamma$ to the uniquely determined symbol $\gamma_i^{(t+1)}$ for all i and t.*

Group Theory and A_5. For elements h and g of a group G, we write g^h for the conjugation $h^{-1}gh$ of g with h and $[h, g]$ for the commutator $h^{-1}g^{-1}hg$. For the neutral element of a group, we write $\mathbb{1}$. We write $\boldsymbol{p} =_G \boldsymbol{q}$ or $\boldsymbol{p} = \boldsymbol{q}$ in G if two words $\boldsymbol{p}$ and $\boldsymbol{q}$ over the generators (and their inverses) of a group evaluate to the same group element.

With A_5 we denote the alternating group of degree five, i.e. the group of even permutations of five elements. It was used by Barrington [3] to convert logical circuits of bounded fan-in and logarithmic depth (so-called NC^1*-circuits*) to bounded-width, polynomial-size branching programs. We will not require this actual result (or knowledge of the involved concepts) in the following, but we will make heavy use of the next lemma and the idea to use iterated commutators, which we will outline below.

▶ **Lemma 2** (see Lemma 1 and 3 of [3]). *There are $\sigma, \alpha, \beta \in A_5$ such that $\sigma = [\sigma^\beta, \sigma^\alpha]$.*

From now on, σ, α and β will refer to those mentioned in Lemma 2 (unless stated otherwise).

Word Problem. The *word problem* of a group G generated by a finite set Q is the problem:

Constant: the group G
Input: $\boldsymbol{q} \in (Q \cup Q^{-1})^*$
Question: is $\boldsymbol{q} = \mathbb{1}$ in G?

In addition, if $\mathcal{C}$ is a class of groups, we also consider the *uniform* word problem for $\mathcal{C}$. Here, the group $G \in \mathcal{C}$ is part of the input (in a suitable representation).

Automata. We use the word *automaton* to denote what is more precisely called a letter-to-letter, finite state transducer. Formally, an automaton is a triple $\mathcal{T} = (Q, \Sigma, \delta)$ consisting of a finite set of *states* Q, an input and output alphabet Σ and a set $\delta \subseteq Q \times \Sigma \times \Sigma \times Q$ of *transitions*. For a transition $(p, a, b, q) \in Q \times \Sigma \times \Sigma \times Q$, we usually use the more graphical notation $p \xrightarrow{a/b} q$ where a is the *input* and b is the *output*. Additionally, we use the common way of depicting automata (see e. g. in Example 3). We will usually work with *deterministic* and *complete* automata, i. e. automata where we have $d_{p,a} = \left|\{p \xrightarrow{a/b} q \in \delta \mid b \in \Sigma, q \in Q\}\right| = 1$ for all $p \in Q$ and $a \in \Sigma$. In other words, for every $a \in \Sigma$, every state has exactly one transition with input a.

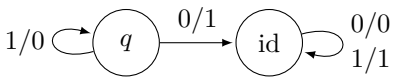
A *run* of an automaton $\mathcal{T} = (Q, \Sigma, \delta)$ is a sequence

$$q_0 \xrightarrow{a_1/b_1} q_1 \xrightarrow{a_2/b_2} \cdots \xrightarrow{a_n/b_n} q_n$$

of transitions from δ. It *starts* in q_0 and *ends* in q_n. Its *input* is $a_1 \dots a_n$ and its *output* is $b_1 \dots b_n$. If $\mathcal{T}$ is complete and deterministic, then, for every state $q \in Q$ and every word $u \in \Sigma^*$, there is exactly one run starting in q with input u. We write $q \circ u$ for its output and $q \cdot u$ for the state in which it ends. This notation can be extended to multiple states. To avoid confusion, we usually use the term *state sequence* instead of 'word' (which we reserve for input or output words) for elements $\boldsymbol{q} \in Q^*$. Now, for states $q_1, q_2, \dots, q_\ell \in Q$, we set $q_\ell \dots q_2 q_1 \circ u = q_\ell \dots q_2 \circ (q_1 \circ u)$ inductively. If the state sequence $\boldsymbol{q} \in Q^*$ is empty, then $\boldsymbol{q} \circ u$ is simply u.

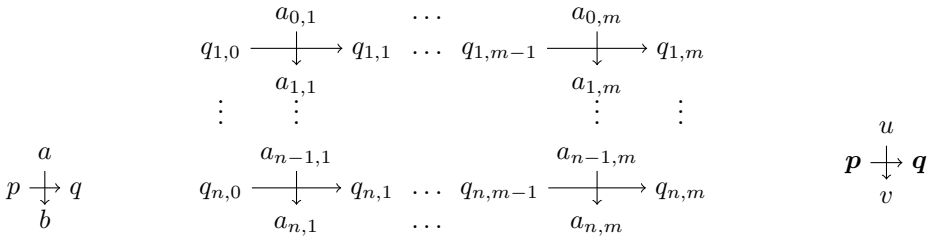
This way, every state $q \in Q$ (and even every state sequence $\boldsymbol{q} \in Q^*$) induces a map $\Sigma^* \to \Sigma^*$ and every word $u \in \Sigma^*$ induces a map $Q \to Q$. If all states of an automaton induce bijective functions, we say it is *invertible* and call it a *$\mathscr{G}$-automaton*. For a $\mathscr{G}$-automaton $\mathcal{T}$, all bijections induced by the states generate a group (with composition as operation), which we denote by $\mathscr{G}(\mathcal{T})$. A group is called an *automaton group* if it arises in this way. Clearly, $\mathscr{G}(\mathcal{T})$ is generated by the maps induced by the states of $\mathcal{T}$ and, thus, finitely generated.

▶ **Example 3.** The typical first example of an automaton generating a group is the *adding machine* $\mathcal{T} = (\{q, \mathrm{id}\}, \{0, 1\}, \delta)$:



It obviously is deterministic and complete and, therefore, we can consider the map induced by state q. We have $q^3 \circ 000 = q^2 \circ 100 = q \circ 010 = 110$. From this example, it is easy to see that the action of q is to increment the input word (which is interpreted as a reverse/least significant bit first binary representation $\overleftarrow{\mathrm{bin}}(n)$ of a number n). The inverse is accordingly to decrement the value. As the other state id acts like the identity, we obtain that the group $\mathscr{G}(\mathcal{T})$ generated by $\mathcal{T}$ is isomorphic to the infinite cyclic group.

Similar to extending the notation $\boldsymbol{q} \circ u$ to state sequences, we can also extend the notation $q \cdot u$. For this, it is useful to introduce *cross diagrams*, another notation for transitions of



(a) Cross diagrams. **(b)** Multiple combined cross diagrams. **(c)** Abbreviated cross diagram.

Figure 1 Combined and abbreviated cross diagrams.

automata. For a transition $p \xrightarrow{a/b} q$ of an automaton, we write the cross diagram given in Figure 1a. Multiple cross diagrams can be combined into a larger one. For example, the cross diagram in Figure 1b indicates that there is a transition $q_{i,j-1} \xrightarrow{a_{i-1,j}/a_{i,j}} q_{i,j}$ for all $1 \le i \le n$ and $1 \le j \le m$. Typically, we omit unneeded names for states and abbreviate cross diagrams. Such an abbreviated cross diagram is depicted in Figure 1c. If we set $q_{n,0} \dots q_{1,0} = \boldsymbol{p}$, $u = a_{0,1} \dots a_{0,m}$, $v = a_{n,1} \dots a_{n,m}$ and $\boldsymbol{q} = q_{n,m} \dots q_{1,m}$, then it indicates the same transitions as the one in Figure 1b. It is important to note here, that the right-most state in $\boldsymbol{p}$ is actually the one to act first.

If we have the cross diagram from Figure 1c, we set $\boldsymbol{p} \cdot u = \boldsymbol{q}$. This is the same, as setting $q_n \dots q_1 \cdot u = [q_n \dots q_2 \cdot (q_1 \circ u)](q_1 \cdot u)$ inductively and, with the definition from above, we already have $\boldsymbol{p} \circ u = v$.

Normally, we cannot simply re-order the rows of a cross diagram as the output interferes and we could get into different states. However, we can clearly re-order rows if they act like the identity:

▶ **Fact 4.** *Let $\mathcal{T} = (Q, \Sigma, \delta)$ be a deterministic automaton, $w \in \Sigma^*$ and $\boldsymbol{q_1}, \dots, \boldsymbol{q_\ell} \in Q^*$ such that $\boldsymbol{q}_1 \circ w = \dots = \boldsymbol{q}_\ell \circ w = w$.*

Then, for any $\boldsymbol{p} = \boldsymbol{p}_k \cdots \boldsymbol{p}_1 \in \{\boldsymbol{q}_1, \dots, \boldsymbol{q}_\ell\}^*$, we have $\boldsymbol{p} \cdot w = (\boldsymbol{p}_k \cdot w) \cdots (\boldsymbol{p}_1 \cdot w)$.

Balanced Iterated Commutators. We lift the notation g^h for conjugation and $[h, g]$ for the commutator from groups to words over the generators: for an alphabet Q and words $\boldsymbol{p}, \boldsymbol{q} \in Q^*$, we write $\boldsymbol{p}^{\boldsymbol{q}} = \boldsymbol{q}^{-1}\boldsymbol{p}\boldsymbol{q}$ and $[\boldsymbol{q}, \boldsymbol{p}] = \boldsymbol{q}^{-1}\boldsymbol{p}^{-1}\boldsymbol{q}\boldsymbol{p}$ using the natural involution for $Q \cup Q^{-1}$. We also need a balanced version of an iterated commutator; in fact, it will be crucial to our constructions.[3]

▶ **Definition 5.** *Let Q be an alphabet and $\alpha, \beta \in (Q \cup Q^{-1})^*$. For $g_t, \dots, g_1 \in (Q \cup Q^{-1})^*$, we inductively define the word $B_{\beta,\alpha}[g_t, \dots, g_1]$ by*

$$B_{\beta,\alpha}[g_1] = g_1$$
$$B_{\beta,\alpha}[g_t, \dots, g_1] = \left[B_{\beta,\alpha}[g_t, \dots, g_{\lfloor \frac{t}{2} \rfloor + 1}]^\beta,\ B_{\beta,\alpha}[g_{\lfloor \frac{t}{2} \rfloor}, \dots, g_1]^\alpha\right].$$

▶ **Lemma 6.** *On input of $g_t, \dots, g_1$, one can compute $B_{\beta,\alpha}[g_t, \dots, g_1]$ in logarithmic space.*

[3] Here, we make an exception as α and β do not refer to the corresponding permutations from Lemma 2 but are arbitrary words. Later on, however, we will apply the definition usually in such a way that α and β indeed are related to the ones from Lemma 2.

▶ Remark 7. For readers familiar with the notions: using a more careful but tedious analysis (and appropriate padding symbols), one can see that the reduction from $g_1, \dots, g_t$ to $B_{\beta,\alpha}[g_t, \dots, g_1]$ can not only be done in logarithmic space but actually it is a DLOGTIME-uniform projection reduction (compare to the proof of Barrington's result in [24, Theorem 4.52]).

If we substitute σ, α and β by the actual elements from A_5, we can see (using a simple induction) that $B_{\beta,\alpha}[g_t, \dots, g_1]$ works as a t-ary logical conjunction:

▶ **Lemma 8.** *For all $g_t, \dots, g_1 \in \{\mathrm{id}, \sigma\} \subseteq A_5$, we have*

$$B_{\beta,\alpha}[g_t, \dots, g_1] =_{A_5} \begin{cases} \sigma & \text{if } g_1 = \cdots = g_t = \sigma \\ \mathbb{1} & \text{otherwise.} \end{cases}$$

▶ Remark 9. Something similar can be done with the free group of rank two (instead of A_5) (see [20]). This is interesting because A_5 cannot be realized as an automaton group over an alphabet with less than five elements but the free group of rank three can be generated by an automaton with binary alphabet [1, 25].

3 Word Problem

In this section, we will show our main result:

▶ **Theorem 10.** *There is an automaton group with a* PSPACE*-complete word problem:*

Constant: *a $\mathscr{G}$-automaton $\mathcal{T} = (Q, \Sigma, \delta)$*
Input: *$\boldsymbol{q} \in Q^*$*
Question: *is $\boldsymbol{q} = \mathbb{1}$ in $\mathscr{G}(\mathcal{T})$?*

In order to prove this theorem, we are going to adapt the construction used in [9, Proposition 6] to show that there is an inverse automaton semigroup with a PSPACE-complete word problem and that there is an automaton group whose word problem with a single rational constraint is PSPACE-complete. The main idea is to use a master reduction. Our automaton operates in two modes. In the first mode, which we will call 'TM-mode', it will interpret its input word as a sequence of configurations of a (suitable) PSPACE-machine and verifies that the configuration sequence constitutes a valid computation of the Turing machine. This verification is done by multiple states (where each state is responsible for a different verification part) and the information whether the verification was successful is stored in the state, **not** by manipulating the input word. So we have *successful states* and *fail states*. Upon reading a special input symbol, the automaton will switch into a second mode, the 'A_5-mode'. More precisely, successful states go into a state which acts like σ from Lemma 2 and the fail states go into an identity state id. Finally, to extract the information from the states, we use the iterated commutator from Definition 5.

The idea for the TM-mode is similar to the approach taken by Kozen to show PSPACE-completeness of the DFA Intersection Problem where the input word is interpreted as a sequence of configurations of a PSPACE Turing machine where each configuration is of length $s(n)$:

$$\gamma_1^{(0)}\gamma_2^{(0)}\gamma_3^{(0)} \cdots \gamma_{s(n)}^{(0)} \# \gamma_1^{(1)}\gamma_2^{(1)}\gamma_3^{(1)} \cdots \gamma_{s(n)}^{(1)} \# \cdots$$

In Kozen's proof, there is an acceptor for each position i of the configurations with $1 \leq i \leq s(n)$ which checks for all t whether the transition from $\gamma_i^{(t)}$ to $\gamma_i^{(t+1)}$ is valid. In our case, however,

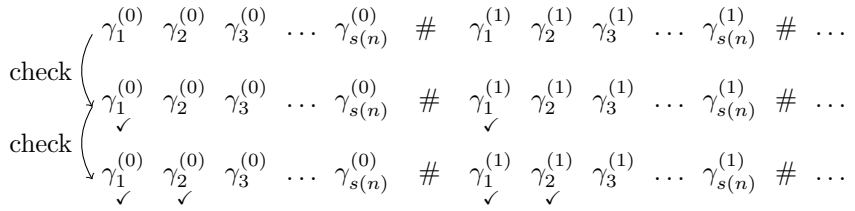


Figure 2 Illustration of the checkmark approach.

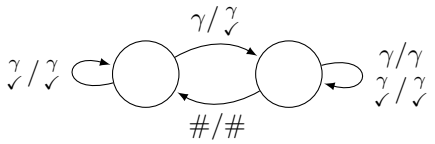


Figure 3 Adding a check-mark yields a non-invertible automaton.

the automaton must not depend on the input (or its length n) and we have to handle this a bit differently. The first idea is to use a 'check-mark approach'. First, we check all first positions for valid transitions. Then, we put a check-mark on all first positions, which tells us that we now have to check all second positions (i. e. the first ones without a check-mark). Again, we put a check-mark on all these, continue with checking all third positions and so on (see Figure 2).

The problem with this approach is that the check-marking leads to an intrinsically non-invertible automaton (see Figure 3). To circumvent this, we generalize the check-mark approach: before each symbol $\gamma_i^{(t)}$ of a configuration, we add a 0^k block (of sufficient length k). In the spirit of Example 3, we interpret this block as representing a binary number. We consider the symbol following the block as 'unchecked' if the number is zero; for all other numbers, it is considered as 'checked'. Now, checking the next symbol boils down to incrementing each block until we have encountered a block whose value was previously zero (and this can be detected while doing the addition). This idea is depicted in Figure 4. It would also be possible to have the check-mark block after each symbol instead of before (which might be more intuitive) but it turns out that our ordering has some technical advantages.

Proof of Theorem 10. Since the uniform word problem for automaton groups is in PSPACE [22] (see also [9, Proposition 2 and 3]), so is the word problem of any (fixed) automaton group. Therefore, we only have to show the hardness part of the result.

Consider an arbitrary PSPACE-complete problem and let M be a deterministic, polyno-

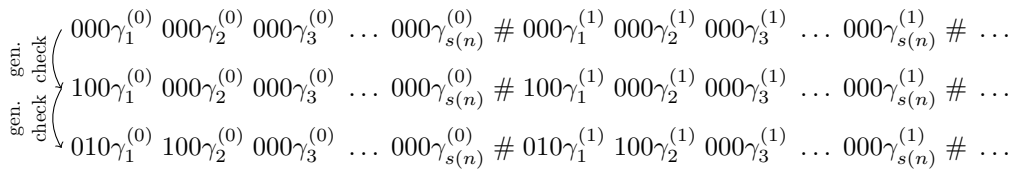


Figure 4 The idea of our generalized check-marking approach.

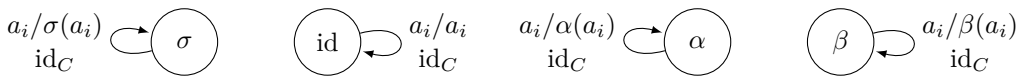
mially space-bounded Turing machine deciding it[4] with input alphabet Λ, tape alphabet Δ, blank symbol $\square$, state set P, initial state p_0 and accepting states $F \subseteq P$. Thus, for any input word of length n, all configurations of M are of the form $\square\Delta^\ell P\Delta^m\square$ with $\ell + 1 + m = s(n)$ for some polynomial s. This makes the problem

Constant:	the PSPACE machine M
Input:	$w \in \Lambda^*$
Question:	does M reach a configuration with a state from F from the initial configuration $\square p_0 w \square^{s(n)-n-1}\square$?

PSPACE-complete. From the machine M, we construct the $\mathscr{G}$-automaton $\mathcal{T}$ and, from the input w, we construct the state sequence $\boldsymbol{q}$. The input words for the automaton will be interpreted to have two parts separated by a special symbol \$. The first part will represent a computation of the machine M and the automaton will be in the TM-mode (mentioned above) while reading it. At the separation symbol, the automaton will switch into the A_5-mode and will operate letter-wise as σ, α, β (from Lemma 2) or the identity on the second part of the word. For the state sequence, we define $\boldsymbol{q} = B_0[f, \boldsymbol{q}_{s(n)}, \dots, \boldsymbol{q}_1, \boldsymbol{c}', \boldsymbol{c}_{s(n)}, \dots, \boldsymbol{c}_1, r]$ where we use the short-hand notation B_0 for the balanced commutator B_{β_0,α_0} from Definition 5. We will define the individual entries $f, \boldsymbol{q}_{s(n)}, \dots, \boldsymbol{q}_1, \boldsymbol{c}', \boldsymbol{c}_{s(n)}, \dots, \boldsymbol{c}_1, r$ of the balanced commutator in detail below. The general idea is that they are used to check different parts of the computation. For example, r is responsible for checking that the first part of the input word is in the correct form for a computation and $\boldsymbol{q}_i$ is responsible for checking that all transitions at position i are valid transitions of M. The individual entries do not change the first part of the word (or – more precisely – any changes are reverted) and operate as the identity if the check failed or as σ if the check succeeded on the second word part. This way, if a single check fails, $\boldsymbol{q}$ as a whole will operate as the identity and, if all checks succeed, it will operate as σ on the second word part by Lemma 8. The commutator acts as a logical conjunction where σ is interpreted as `true` and the identity is interpreted as `false`.

Let $\Gamma = \Delta \uplus P$. From now on, we will not work with M anymore but rather only with its corresponding $\tau : \Gamma^3 \to \Gamma$ from Fact 1.

Construction of the Automaton. The automaton $\mathcal{T}$ works in the way described above and is the union of several simpler automata. For the alphabet, we use $\Sigma = \Gamma \uplus \{0, 1\} \uplus \{\#, \$\}$ with new letters 0, 1, # and \$. The letters 0 and 1 will be used for the generalized check-mark approach described above, the letter # is used to separate individual configurations and \$ acts as an 'end-of-computation' symbol switching the automaton from the TM-mode to the A_5-mode. For the A_5-mode, we choose $a_1, \dots, a_5 \in \Sigma$ arbitrarily (but distinct) and assume that σ, α and β (from Lemma 2) operate on $\{a_1, \dots, a_5\}$ such that $\sigma(a_1) \neq a_1$. Furthermore, we set $C = \Sigma \setminus \{a_1, \dots, a_5\}$. With this, the first part of the automaton $\mathcal{T}$ used for the A_5-mode is



where the a_i-transitions exist for all $i \in \{1, \dots, 5\}$ and we use the convention that id_X indicates x/x-transitions for all $x \in X \subseteq \Sigma$. Obviously, the state id acts as the identity and the action of the state σ on a word is to apply the permutation σ letter-wise (and to ignore letters from C), which justifies the dual use in notation as we can identify σ in the

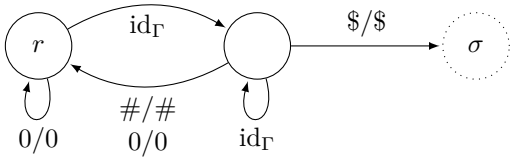
[4] Alternatively, we could also use a PSPACE-universal Turing machine for our construction.

automaton group with σ in A_5. For the intuition, it helps to see id as a 'fail' state and σ as an 'okay' state in the following. In the end, we will implement this intuition basically using the iterated commutator from Definition 5. For this commutator, we also need the conjugating elements α and β, which work in the same way as σ. However, they do not have an intuitive semantic and are mostly there for technical reasons.

Now, let us describe the part of the automaton used for the TM-mode. First, we need two states which ignore everything in the TM-mode and then go to α or β:

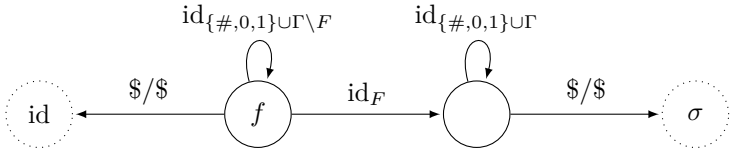
where dotted states refer to the states defined above.

The next part of our automaton is used to check that the input word (for the TM-mode) is of the form $(0^*\Gamma)^+ (\#(0^*\Gamma)^+)^*$:



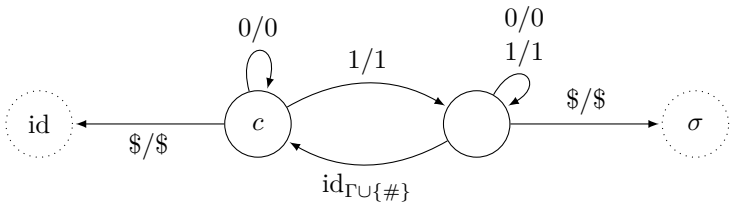
Here, we use the convention that, whenever a transition is missing for some $x \in \Sigma$, there is an implicit x/x-transition to the state id (as defined above). Note that we do not check that the factors in $(0^*\Gamma)^+$ correspond to well-formed configurations for the Turing machine. This will be done implicitly by checking that the input word belongs to a valid computation of the Turing machine, which we describe below.

Next, we need a part which checks whether the input word contains a final state (if this is not the case, we want to 'reject' the word):



Finally, we come to the more complicated parts of $\mathcal{T}$. The first one is for the generalized check-marking as described above and is depicted in Figure 5. In fact, we need this part *twice*: once for $g = \sigma$ and once for $g = \mathrm{id}$. Notice that, during the TM-mode phase (i. e. before the first \$), the two versions behave exactly the same way; the only difference is after switching to the A_5-mode: while $\checkmark_{\mathrm{id}}$ still always acts like the identity, $\checkmark_\sigma$ acts non-trivially on suitable input words.

Additionally, we also need an automaton part verifying that every configuration symbol has been check-marked (in the generalized sense):



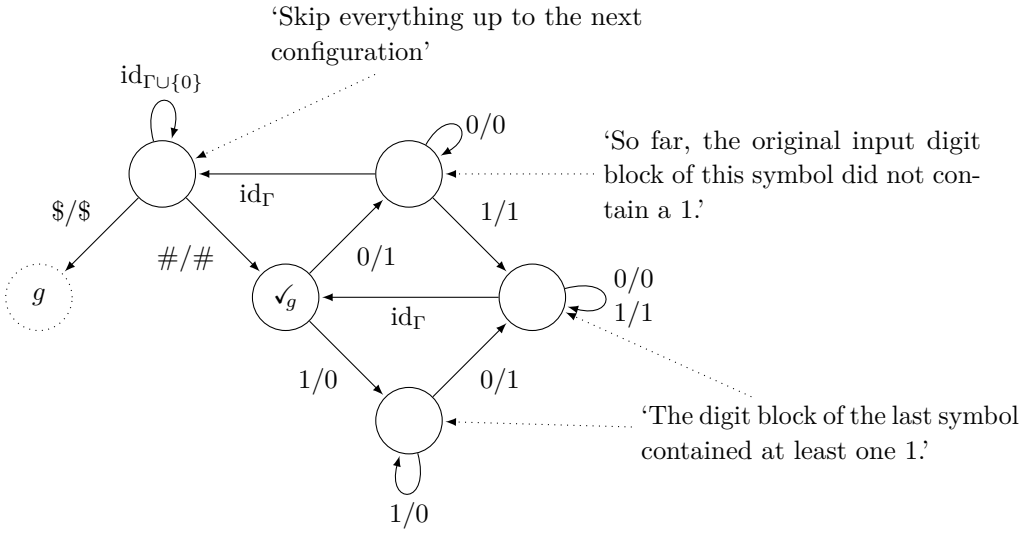
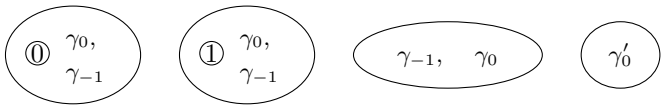


Figure 5 The automaton part used for generalized check-marking.

The last part is for checking the validity of the transition at all first so-far unchecked positions. While it is not really difficult, this part is a bit technical. Intuitively, for checking the transition from time step $t-1$ to time step t at position i, we need to compute $\gamma_i^{(t)} = \tau(\gamma_{i-1}^{(t-1)}, \gamma_i^{(t-1)}, \gamma_{i+1}^{(t-1)})$ from the configuration symbol at positions $i-1$, i and $i+1$ for time step $t-1$. We store $\gamma_i^{(t)}$ in the state (to compare it to the actual value). Additionally, we need to store the last two symbols of configuration t we have encountered so far (for computing what we expect in the next time step later on) and whether we have seen a 1 or only 0s in the check-mark digit block. For all this, we use the states



with $\gamma_{-1}, \gamma_0, \gamma_0' \in \Gamma$. The idea is the following. In the ⓪ and ① states, we store the value we expect for the first unchecked symbol (γ_0) and the last symbol we have seen in the current configuration (γ_{-1}). We are in the ⓪-state if we have not seen any 1 in the digit block yet and in the ① if we did. The latter two are used to skip the rest of the current configuration and to compute the symbol we expect for the first unchecked position in the next configuration (γ_0').

We use these states in the transitions schematically depicted in Figure 6. Here, the dashed transitions exist for all γ_{-1}' and γ_1 in Γ but go to different states, respectively, and the dotted states correspond to the respective non-dotted states with different values for γ_0 and γ_{-1} (with the exception of σ, which corresponds to the state defined above). We also define $q_{\gamma'}$ as the state on the bottom right (for every $\gamma' \in \Gamma$, respectively).

The automaton parts depicted in Figure 5 and Figure 6 are best understood with an example. Consider the input word

$$100\gamma_1^{(0)}\ 000\gamma_2^{(0)}\ 000\gamma_3^{(0)}\ \#\ 100\gamma_1^{(1)}\ 000\gamma_2^{(1)}\ 000\gamma_3^{(1)}\ \$$$

where we consider the $\gamma_i^{(t)}$ to form a valid computation. If we start in state $q_{\gamma_2^{(0)}}$ and read the above word, we immediately take the 1/1-transition and go into the corresponding ① state where we skip the rest of the digit block. Using the dashed transition, the next symbol

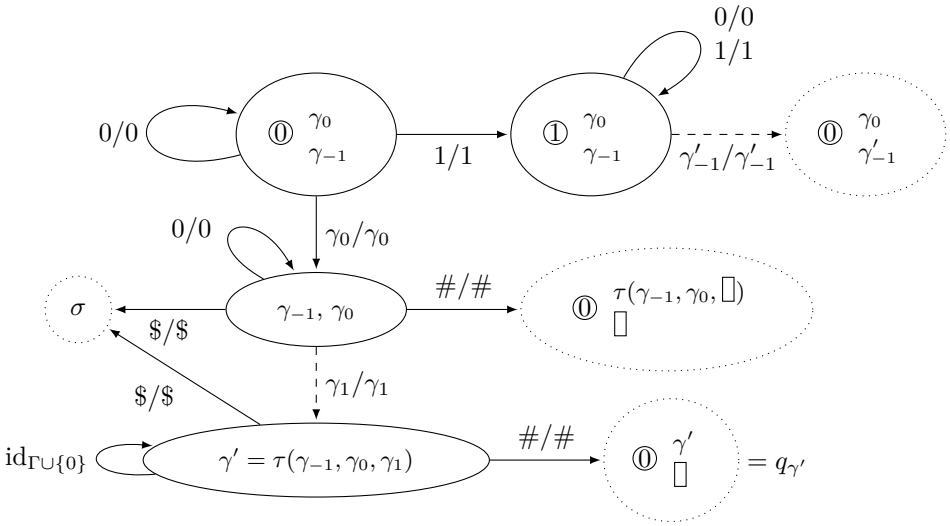


Figure 6 Schematic representation of the transitions used for checking Turing machine transitions and definition of q'_γ; the dashed transitions exist for all γ'_{-1} and γ_1 in Γ but go to different states, respectively

$\gamma_1^{(0)}$ takes us back into a ⓪-state where the upper entry is still $\gamma_2^{(0)}$ but the lower entry is now $\gamma_1^{(0)}$ (i. e. the last configuration symbol we just read). We loop at this state while reading the next three 0s and, since the next symbol $\gamma_2^{(0)}$ matches with the one stored in the state, we get into the states with entries $\gamma_1^{(0)}, \gamma_2^{(0)}$ where we skip the next three 0s again. Reading $\gamma_3^{(0)}$ now gets us into the state with entry $\gamma_2^{(1)}$ since we have $\tau(\gamma_1^{(0)}, \gamma_2^{(0)}, \gamma_3^{(0)}) = \gamma_2^{(1)}$ by assumption that the $\gamma_i^{(t)}$ form a valid computation. Here, we read #/# and the process repeats for the second configuration, this time starting in $q_{\gamma_2^{(2)}}$. When reading the final \$, we are in the state with entry $\tau(\gamma_1^{(1)}, \gamma_2^{(1)}, \gamma_3^{(1)})$ and finally go to σ. Notice that during the whole process, we have not changed the input word at all!

If we now start reading the input word again in state $\checkmark_\sigma$ (see Figure 5 and also refer to Figure 4), we turn the first 1 into a 0, go to the state at the bottom, turn the next 0 into a 1 and go to the state on the right, where we ignore the next 0. When reading $\gamma_1^{(0)}$, we go back to $\checkmark_\sigma$. Next, we take the upper exit and turn the next 0 into a 1. The remaining 0s are ignored and we remain in the state at the top right until we read $\gamma_2^{(0)}$ and go to the state at the top left. Here, we ignore everything up to #, which gets us back into $\checkmark_\sigma$. The second part works in the same way with the difference that we go to σ at the end since we encounter the \$ instead of #. The output word, thus, is

$$010\gamma_1^{(0)}\ 100\gamma_2^{(0)}\ 000\gamma_3^{(0)}\ \#\ 010\gamma_1^{(1)}\ 100\gamma_2^{(1)}\ 000\gamma_3^{(1)}\ \$$$

and we have check-marked the next position in both configurations.

This concludes the definition of the automaton and the reader may verify that $\mathcal{T}$ is indeed a $\mathscr{G}$-automaton since all individual parts are $\mathscr{G}$-automata. Furthermore, apart from the check-marking, all states except σ, α and β (which belong to the A_5-mode and are only entered when reading \$) act like the identity.

Definition of the State Sequence. To describe the actual reduction, we have to define the state sequence $\boldsymbol{q}$ such that $\boldsymbol{q}$ depends only on the input word w for the Turing machine. Similar to the automaton $\mathcal{T}$, this sequence consists of multiple parts. Each part will verify a certain aspect of the input word and the general idea is that, after reading a word $u\$$, we

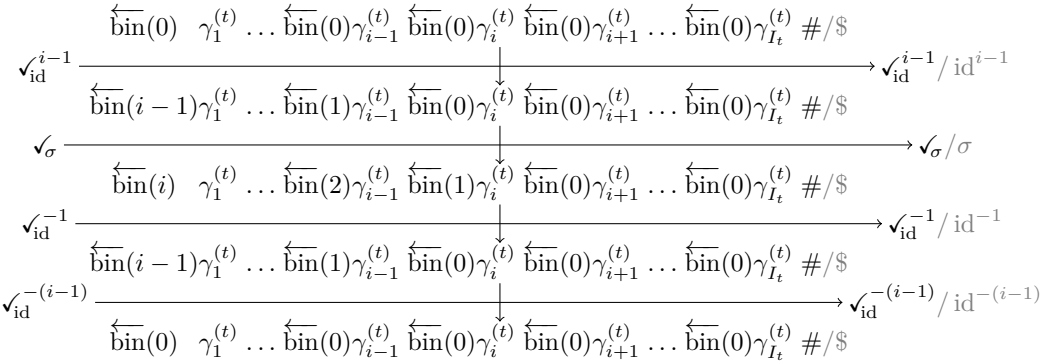
are either in σ (if u satisfies the criterion we are currently checking) or in id (if it does not). Finally, using the balanced commutator from Definition 5, we can find whether any of the criteria was not satisfied. For this to work easily, we define the individual parts in such a way that the output will be $u\$$ again, which will allow us to apply Fact 4.

First, we simply use the state r to verify that u is from $(0^*\Gamma)^+ (\#(0^*\Gamma)^+)^*$. Thus, we only need to consider the case that u is of the form

$$0^{\ell_1^{(0)}}\gamma_1^{(0)}\ 0^{\ell_2^{(0)}}\gamma_2^{(0)}\ \dots\ 0^{\ell_{I_0}^{(0)}}\gamma_{I_0}^{(0)}\ \#\dots\#\ 0^{\ell_1^{(T)}}\gamma_1^{(T)}\ 0^{\ell_2^{(T)}}\gamma_2^{(T)}\ \dots\ 0^{\ell_{I_T}^{(T)}}\gamma_{I_T}^{(T)} \tag{$\dagger$}$$

with $\gamma_i^{(t)} \in \Gamma$ any further.

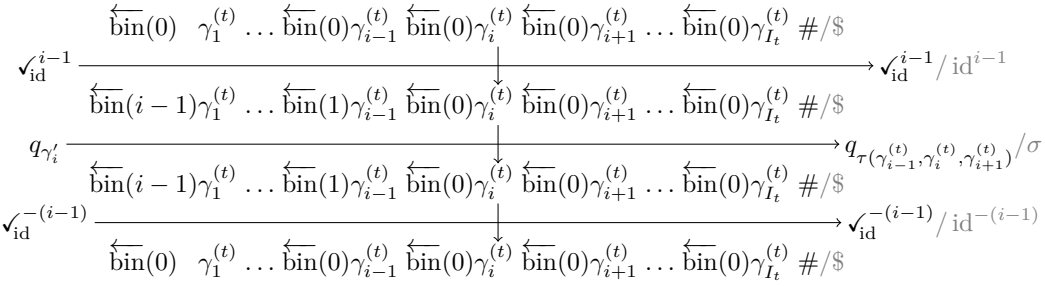
Next, we need to verify that, for every $1 \le i \le s(n)$, we can check-mark the first i positions. For this, we use $\boldsymbol{c}_i = \checkmark_{\mathrm{id}}^{-i}\checkmark_\sigma\checkmark_{\mathrm{id}}^{i-1}$ as we have the cross diagram



where $\overleftarrow{\mathrm{bin}}(z)$ denotes the reverse/least significant bit first binary representation of z (of sufficient length). Here, it is useful to observe that, if the j^{th} 0 block with $j \le i$ is not long enough to count to its required value, then we will always end up in id after reading a \$. The same happens if $I_t < i$ (i. e. if one of the configurations is 'too short'). So this guarantees, $I_t \ge s(n)$ for all t.

On the other hand, we use $\boldsymbol{c}' = \checkmark_{\mathrm{id}}^{-s(n)} c \checkmark_{\mathrm{id}}^{s(n)}$ to ensure that, after check-marking the first $s(n)$ positions in every configurations, all symbols have been check-marked (i. e. that no configuration is 'too long'), which guarantees $I_t = s(n)$ for all t.

Now that we have ensured that the word is of the correct form and we can count high enough for our check-marking, we need to actually verify that the $\gamma_i^{(t)}$ constitute a valid computation of the Turing machine with the initial configuration $\gamma'_1 \dots \gamma'_{s(n)} = p_0 w \square^{s(n)-n-1}$ for the input word w. To do this, we define $\boldsymbol{q}_i = \checkmark_{\mathrm{id}}^{-(i-1)} q_{\gamma'_i} \checkmark_{\mathrm{id}}^{i-1}$ for every $1 \le i \le s(n)$ as we have the cross diagram



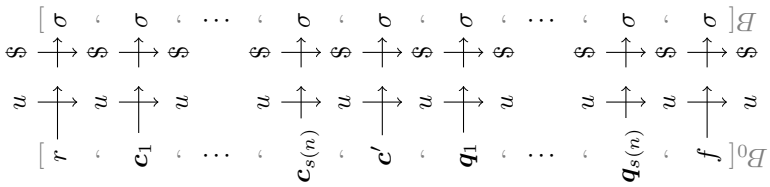


Figure 7 Cross diagram for $\boldsymbol{q}$.

if $\gamma_i^{(t)}$ is the expected γ_i'. Otherwise (if $\gamma_i^{(t)} \neq \gamma_i'$), we always end in state id after reading the \$. Finally, to ensure that the computation is not only valid but also accepting, we use the state f.

Summing this up, we define $\boldsymbol{q} = B_0[f, \boldsymbol{q}_{s(n)}, \ldots, \boldsymbol{q}_1, \boldsymbol{c}', \boldsymbol{c}_{s(n)}, \ldots, \boldsymbol{c}_1, r]$ as mentioned at the begnning of the proof. Remember that we use the short-hand notation B_0 for the balanced commutator B_{β_0,α_0} from Definition 5 and observe that the individual parts of $\boldsymbol{q}$ can indeed be computed in logarithmic space and that, thus, this is also true for $\boldsymbol{q}$ itself by Lemma 6.

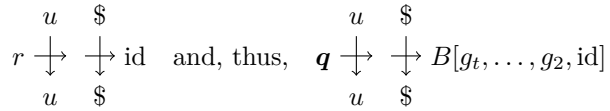
Correctness. We need to prove that the action of $\boldsymbol{q}$ is equal to the identity if and only if the Turing machine does **not** accept the input word w. The easier direction is to assume that the Turing machine accepts on the initial configuration $\square p_0 w \square^{s(n)-n-1}\square$. Let $\gamma_1^{(0)} \ldots \gamma_{s(n)}^{(0)} \vdash \gamma_1^{(1)} \ldots \gamma_{s(n)}^{(1)} \vdash \cdots \vdash \gamma_1^{(T)} \ldots \gamma_{s(n)}^{(T)}$ be the corresponding computation with $\gamma_1^{(0)} = p_0$, $\gamma_2^{(0)} \ldots \gamma_{n+1}^{(0)} = w$ and $\gamma_i^{(T)} \in F$ for some $1 \leq i \leq s(n)$. We choose $\ell = \lceil \log(s(n)) \rceil + 1$ and define

$$u = 0^\ell \gamma_1^{(0)} \ldots 0^\ell \gamma_{s(n)}^{(0)} \# 0^\ell \gamma_1^{(1)} \ldots 0^\ell \gamma_{s(n)}^{(1)} \# \ldots \# 0^\ell \gamma_1^{(T)} \ldots 0^\ell \gamma_{s(n)}^{(T)}.$$

We now let $\boldsymbol{q}$ act on the word $u\$a_1$. Recall that we assume σ to operate non-trivially on a_1. The reader may verify that we have the black part of the cross diagram depicted in Figure 7. From Fact 4, we immediately also obtain the gray additions to the cross diagram where we use B instead of $B_{\beta,\alpha}$ for the balanced commutator from Definition 5. By Lemma 8, we obtain $B[\sigma, \ldots, \sigma] = \sigma$ in A_5 and, thus, in $\mathscr{G}(\mathcal{T})$. Therefore, $\boldsymbol{q}$ acts non-trivially on $u\$a_1$.

For the other direction, assume that no valid computation of M on the initial configuration $\square p_0 w \square^{s(n)-n-1}\square$ contains an accepting state from F. We have to show that $\boldsymbol{q}$ acts like the identity on all words from Σ^*. If the word does not contain a \$, then all individual parts of $\boldsymbol{q}$ act on it like the identity by construction. This is clearly the case for r, $\boldsymbol{c}'$, the $\boldsymbol{q}_i$ and f. For the $\boldsymbol{c}_i$, the only point to note is that $\checkmark_\sigma$ acts in the same way as $\checkmark_{\mathrm{id}}$ on such words.

Thus, we may assume that the word is of the form $u\$v$. If u is not of the form $(0^*\Gamma)^+ (\#(0^*\Gamma)^+)^*$, we have the cross diagram



with $g_2, \ldots, g_t \in \{\sigma, \mathrm{id}\}$. As we have, $B[g_t, \ldots, g_2, \mathrm{id}] =_{A_5} \mathbb{1}$ by Lemma 8, we obtain that $\boldsymbol{q}$ acts like the identity on $u\$v$.

Therefore, we assume u to be of the form mentioned in Equation † and use a similar argumentation for the remaining cases. If u does not contain a state from F, then we end up in state id after reading \$ for f. As w is not accepted by the machine, this includes in particular all valid computations on the initial configuration $\square p_0 w \square^{s(n)-n-1}\square$. If one of the

0 blocks in u is too short to count to a value required for the check-marking (i. e. one $\ell_i^{(t)}$ is too small), then the corresponding $\boldsymbol{c}_i$ will go to (a state sequence equivalent to) id. This is also true if one configuration is too short (i. e. $I_t < s(n)$ for some t). If one configuration is too long (i. e. $I_t > s(n)$), then this will be detected by $\boldsymbol{c}'$ as not all positions will be check-marked after check-marking all first $s(n)$ positions in every configuration. Finally, $\boldsymbol{q}_i$ yields an id if $\gamma_i^{(0)}$ is not the correct symbol from the initial configuration or if we have $\gamma_i^{(t+1)} \neq \tau(\gamma_{i-1}^{(t)}, \gamma_i^{(t)}, \gamma_{i+1}^{(t)})$ for some t (where we let $\gamma_{-1}^{(t)} = \square = \gamma_{s(n)+1}^{(t)}$). ◀

▶ Remark 11. The constructed automaton has $3\Gamma^2 + \Gamma + 22$ states where Γ is the sum of the number of states and the number of tape symbols for a Turing machine for a PSPACE-hard problem.

▶ Remark 12. Using Remark 9, we can reduce the alphabet of the automaton to a binary one. This, however, involves some technical difficulties mainly for two reasons. First, the original alphabet needs to be compressed into blocks over the binary alphabet and, second, the element to be used in the balanced iterated commutator depends on the position.

4 Compressed Word Problem

In this section, we re-apply our previous construction to show that there is an automaton group with an EXPSPACE-complete compressed word problem. The *compressed word problem* of a group is similar to the normal word problem. However, the input element (to be compared to the neutral element) is not given directly but as a straight-line program. A *straight-line program* is a context-free grammar which generates exactly one word.

▶ **Theorem 13.** *There is an automaton group with an* EXPSPACE*-complete compressed word problem:*

Constant:	*a $\mathscr{G}$-automaton $\mathcal{T} = (Q, \Sigma, \delta)$*
Input:	*a straight-line program generating a state sequence $\boldsymbol{q} \in Q^*$*
Question:	*is $\boldsymbol{q} = \mathbb{1}$ in $\mathscr{G}(\mathcal{T})$?*

Proof. Before starting, we observe that, whenever we have a variable X in a straight-line program generating a word representing a group element g, we can easily obtain a variable X^{-1} generating a word representing g^{-1} by mirroring all rules for X, replacing all letters a by a^{-1} and all variables A by A^{-1} (where we have to continue recursively). Hence, we will always assume that we also have A^{-1} if we have described A in the following.

For the actual proof, we use the same construction as in the proof of Theorem 10, but we start with a Turing machine M for an EXPSPACE-complete problem. Now, all configurations on input of a word w of length n are of the form $\square\Delta^\ell P \Delta^m\square$ with $\ell + 1 + m = s(n)$ where $s(n)$ is of the form 2^{n^e} for some constant $e \in \mathbb{N}$.

Recall that, for the (normal) word problem, we used

$$\boldsymbol{q} = B_0[f, \boldsymbol{q}_{s(n)}, \ldots, \boldsymbol{q}_1, \boldsymbol{c}', \boldsymbol{c}_{s(n)}, \ldots, \boldsymbol{c}_1, r] \tag{‡}$$

for the reduction where

$$\boldsymbol{c}_i = \checkmark_{\mathrm{id}}^{-i} \checkmark_\sigma \checkmark_{\mathrm{id}}^{i-1}, \quad \boldsymbol{c}' = \checkmark_{\mathrm{id}}^{-s(n)} c \checkmark_{\mathrm{id}}^{s(n)} \quad \text{and} \quad \boldsymbol{q}_i = \checkmark_{\mathrm{id}}^{-(i-1)} q_{\gamma_i'} \checkmark_{\mathrm{id}}^{i-1}$$

for $1 \leq i \leq s(n)$. The problem now is that we have exponentially many $\boldsymbol{c}_i$ and $\boldsymbol{q}_i$. Thus, we cannot output them in logarithmic space (or polynomial time), not even if we use a straight-line program. So, we will have to accommodate for this. The good news is that, for

$\boldsymbol{c}'$, this is not a problem: we can output a straight-line program with starting variable C' generating $\boldsymbol{c}'$:

$$C' \to M_{2^{n^e}} c M_{2^{n^e}}^{-1}, \quad M_{2^{n^e}} \to M_{2^{n^e-1}} M_{2^{n^e-1}}, \quad \ldots, \quad M_{2^1} \to M_{2^0} M_{2^0}, \quad M_{2^0} \to \checkmark_{\mathrm{id}}$$

Notice that this program is of polynomial length in n and, clearly, C' evaluates to $\boldsymbol{c}'$.

To circumvent the problem with the $\boldsymbol{c}_i$, we use the fact that the behavior of $\boldsymbol{c}_i$ is structurally the same as the behavior of $\boldsymbol{c}_j$. This can be exploited by defining a slightly modified version of the balanced commutator B_0 (where B_0 is as in the proof of Theorem 10): let $B_c[1] = \checkmark_{\mathrm{id}}^{-1} \checkmark_\sigma$ and

$$\begin{aligned} B_c[k] &= \left[B_c[\lceil \tfrac{k}{2} \rceil]^{\beta_0 \checkmark_{\mathrm{id}}^{\lfloor \frac{k}{2} \rfloor}},\ B_c[\lfloor \tfrac{k}{2} \rfloor]^{\alpha_0} \right] \\ &= \checkmark_{\mathrm{id}}^{-\lfloor \frac{k}{2} \rfloor}\ \beta_0^{-1} B_c[\lceil \tfrac{k}{2} \rceil]^{-1} \beta_0\ \checkmark_{\mathrm{id}}^{\lfloor \frac{k}{2} \rfloor}\ \alpha_0^{-1} B_c[\lfloor \tfrac{k}{2} \rfloor]^{-1} \alpha_0 \\ &\quad\ \checkmark_{\mathrm{id}}^{-\lfloor \frac{k}{2} \rfloor}\ \beta_0^{-1} B_c[\lceil \tfrac{k}{2} \rceil] \beta_0\ \checkmark_{\mathrm{id}}^{\lfloor \frac{k}{2} \rfloor}\ \alpha_0^{-1} B_c[\lfloor \tfrac{k}{2} \rfloor] \alpha_0. \end{aligned}$$

The intuitive idea behind this definition is that we start with all check-mark counters at value zero ('nothing is check-marked'). Then the bottom right part checks that we can check the first $\lfloor \frac{k}{2} \rfloor$ positions. However, the counters are reset to zero during this checking (by induction), so we check-mark the positions again (without modifying the value in the A_5-mode) using a suitable power of $\checkmark_{\mathrm{id}}$. Then the part on the bottom left checks that we can check-mark the next $\lceil \frac{k}{2} \rceil$ positions. After this, the counters are reset to zero. Then the same happens again only using the inverse group elements this time to realize the commutator. Here, it is important to note that the top left part sees the same situation as the bottom left part, so it behaves in the same way except for going to σ^{-1} instead of σ in the A_5-mode.

Formally, we can show $B_c[k] =_{\mathscr{G}(\mathcal{T})} B_0[\boldsymbol{c}_k, \ldots, \boldsymbol{c}_1]$ by induction. The base case holds by definition and, for the induction step, we observe that $\checkmark_{\mathrm{id}}$ commutes with α_0 and β_0 and that we have $\boldsymbol{c}_{i+d} = \checkmark_{\mathrm{id}}^{-d} \boldsymbol{c}_i \checkmark_{\mathrm{id}}^{d}$ in $\mathscr{G}(\mathcal{T})$. Since we have $[y,x]^z = [y^z, x^z]$ in any group, we obtain in $\mathscr{G}(\mathcal{T})$

$$\begin{aligned} B_c[k] &= \left[B_0[\boldsymbol{c}_{\lceil \frac{k}{2} \rceil}, \ldots, \boldsymbol{c}_1]^{\beta_0 \checkmark_{\mathrm{id}}^{\lfloor \frac{k}{2} \rfloor}},\ B_0[\boldsymbol{c}_{\lfloor \frac{k}{2} \rfloor}, \ldots, \boldsymbol{c}_1]^{\alpha_0} \right] && \text{(by induction)} \\ &= \left[B_0\left[\boldsymbol{c}_{\lceil \frac{k}{2} \rceil}^{\checkmark_{\mathrm{id}}^{\lfloor \frac{k}{2} \rfloor}}, \ldots, \boldsymbol{c}_1^{\checkmark_{\mathrm{id}}^{\lfloor \frac{k}{2} \rfloor}}\right]^{\beta_0},\ B_0[\boldsymbol{c}_{\lfloor \frac{k}{2} \rfloor}, \ldots, \boldsymbol{c}_1]^{\alpha_0} \right] \\ &= \left[B_0[\boldsymbol{c}_k, \ldots, \boldsymbol{c}_{\lfloor \frac{k}{2} \rfloor + 1}]^{\beta_0},\ B_0[\boldsymbol{c}_{\lfloor \frac{k}{2} \rfloor}, \ldots, \boldsymbol{c}_1]^{\alpha_0} \right] \\ &= B_0[\boldsymbol{c}_k, \ldots, \boldsymbol{c}_1]. \end{aligned}$$

In particular, this shows that no $B_c[k]$ modifies a word not containing \$ (i. e. all $B_c[k]$ act like the identity in the TM-mode). This is important as we will be using $B_c[s(n)] = B_c[2^{n^e}]$ for checking that we can check-mark all positions in the end.

The straight-line program for $B_c[s(n)]$ is similar to the one of $\boldsymbol{c}'$ (and uses the variables M_{2^i} defined above). We let

$$\begin{aligned} B_{c,2^\ell} \to\ & M_{2^{\ell-1}}^{-1}\ \beta_0^{-1} B_{c,2^{\ell-1}}^{-1} \beta_0\ M_{2^{\ell-1}}\ \alpha_0^{-1} B_{c,2^{\ell-1}}^{-1} \alpha_0 \\ & M_{2^{\ell-1}}^{-1}\ \beta_0^{-1} B_{c,2^{\ell-1}} \beta_0\ M_{2^{\ell-1}}\ \alpha_0^{-1} B_{c,2^{\ell-1}} \alpha_0 \end{aligned}$$

for $\ell > 0$ and $B_{c,2^0} \to \checkmark_{\mathrm{id}}^{-1} \checkmark_\sigma$. This way, we can compute $B_{c,2^{n^e}}$ as a variable for $B_c[s(n)]$ in logarithmic space.

At first, it seems that we cannot use a similar idea for the $\boldsymbol{q}_i$ as they are responsible for different symbols of the configuration. However, as the initial configuration is $\square p_0 w \square^{2^{n^e}-n}\square$, all $\boldsymbol{q}_i$ except of the first $n+1$ ones are responsible for the same tape symbol (the blank symbol $\square$). We define $B_q[1] = q_\square$; the inductive step is the same as in the definition of $B_c[k]$ and the same ideas apply. This time, however, we want to start with the first $n+1$ positions check-marked (this is the $p_0 w$ part) and, thus, we will be using $\checkmark_{\mathrm{id}}^{-n-1} B_q[s(n)-n-1]\checkmark_{\mathrm{id}}^{n+1}$ and the original $\boldsymbol{q}_i$ for the first $n+1$ positions. With the same arguments as for $B_c[s(n)]$, one can show

$$\checkmark_{\mathrm{id}}^{-n-1} B_q[s(n)-n-1]\checkmark_{\mathrm{id}}^{n+1} =_{\mathscr{G}(\mathcal{T})} B_0[\boldsymbol{q}_{s(n)}, \ldots, \boldsymbol{q}_{n+2}].$$

A straight-line program for $B_q[s(n)-n-1]$ can be obtained in a similar way to the one for $B_c[s(n)]$[5] and the $\checkmark_{\mathrm{id}}$ blocks are short enough to output them directly.

Summing this up, we set

$$Q \to B_0[f,\ \checkmark_{\mathrm{id}}^{-n-1} B_q \checkmark_{\mathrm{id}}^{n+1},\ \boldsymbol{q}_{n+1}, \ldots, \boldsymbol{q}_1,\ C',\ B_{c,s(n)},\ r]$$

where B_q is the starting variable of the straight-line program for $B_q[s(n)-n-1]$. By Lemma 6, we can compute the right-hand side of this rule in logarithmic space. By the construction of the straight-line programs for B_q, $B_{c,s(n)}$ and C' (and the proved equalities), it follows that Q evaluates to

$$B_0[f,\ B_0[\boldsymbol{q}_{s(n)}, \ldots, \boldsymbol{q}_{n+2}],\ \boldsymbol{q}_{n+1}, \ldots, \boldsymbol{q}_1,\ \boldsymbol{c}',\ B_0[\boldsymbol{c}_{s(n)}, \ldots, \boldsymbol{c}_1],\ r].$$

Now, using Lemma 8, it is easy to see that this is equal to $\boldsymbol{q}$ from (‡) in $\mathscr{G}(\mathcal{T})$: before reading the first letter \$, it operates as the identity and, after the first letter \$, it operates either as the identity or as σ depending on whether all components of the commutator act like σ. ◀

▶ **Corollary 14.** *There is an automaton group with a* PSPACE*-complete word problem and an* EXPSPACE*-complete compressed word problem.*

Proof. The result follows from the closure of the class of automaton groups under direct product (which is well-known and easy to see) in combination with Theorem 10 and Theorem 13. ◀

References

1 Stanislav V. Aleshin. A free group of finite automata. *Vestnik Moskovskogo Universiteta. Seriya I. Matematika, Mekhanika*, 4:12–14, 1983.

2 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

3 David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989. doi:10.1016/0022-0000(89)90037-8.

4 Laurent Bartholdi and Ivan Mitrofanov. The word and order problems for self-similar and automata groups. *arXiv preprint*, 2017. arXiv:1710.10109.

5 Ievgen V. Bondarenko. Growth of Schreier graphs of automaton groups. *Mathematische Annalen*, 354(2):765–785, 2012. doi:10.1007/s00208-011-0757-x.

[5] Be aware that this is a bit more technical, since we are not only working with powers of two anymore and need to take the rounding into consideration.

6 Ievgen V. Bondarenko. The word problem in Hanoi Towers groups. *Algebra and Discrete Mathematics*, 17(2):248–255, 2014.

7 William W. Boone. The Word Problem. *Annals of Mathematics*, 70(2):207–265, 1959.

8 Daniele D'Angeli, Emanuele Rodaro, and Jan Philipp Wächter. The structure theory of partial automaton semigroups. *arXiv preprint*, 2018. `arXiv:1811.09420`.

9 Daniele D'Angeli, Emanuele Rodaro, and Jan Philipp Wächter. On the complexity of the word problem for automaton semigroups and automaton groups. *Advances in Applied Mathematics*, 90:160–187, 2017. `doi:10.1016/j.aam.2017.05.008`.

10 Max Dehn. Über unendliche diskontinuierliche Gruppen. *Math. Ann.*, 71:116–144, 1911.

11 Pierre Gillibert. Personal Communication.

12 Pierre Gillibert. The finiteness problem for automaton semigroups is undecidable. *International Journal of Algebra and Computation*, 24(01):1–9, 2014. `doi:10.1142/S0218196714500015`.

13 Pierre Gillibert. An automaton group with undecidable order and Engel problems. *Journal of Algebra*, 497:363–392, 2018. `doi:10.1016/j.jalgebra.2017.11.049`.

14 Rostislav Grigorchuk and Igor Pak. Groups of intermediate growth: an introduction. *L'Enseignement Mathématique*, 54:251–272, 2008.

15 Gennadiĭ S. Makanin. Decidability of the universal and positive theories of a free group. *Izv. Akad. Nauk SSSR*, Ser. Mat. 48:735–749, 1984. In Russian; English translation in: *Math. USSR Izvestija, 25*, 75–88, 1985.

16 Anatolij I. Mal'cev. On the equation $zxyx^{-1}y^{-1}z^{-1} = aba^{-1}b^{-1}$ in a free group. *Akademiya Nauk SSSR. Sibirskoe Otdelenie. Institut Matematiki. Algebra i Logika*, 1(5):45–50, 1962.

17 Volodymyr V. Nekrashevych. *Self-similar groups*, volume 117 of *Mathematical Surveys and Monographs*. American Mathematical Society, Providence, RI, 2005. `doi:10.1090/surv/117`.

18 Petr S. Novikov. On the algorithmic unsolvability of the word problem in group theory. *Trudy Mat. Inst. Steklov*, pages 1–143, 1955. In Russian.

19 Christos M. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

20 David Robinson. *Parallel Algorithms for Group Word Problems*. PhD thesis, University of California, San Diego, 1993.

21 Richard Edwin Stearns, Juris Hartmanis, and Philip M. Lewis. Hierarchies of memory limited computations. In *6th Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1965)*, pages 179–190. IEEE, 1965.

22 Benjamin Steinberg. *On some algorithmic properties of finite state automorphisms of rooted trees*, volume 633 of *Contemporary Mathematics*, pages 115–123. American Mathematical Society, 2015.

23 Zoran Šunić and Enric Ventura. The conjugacy problem in automaton groups is not solvable. *Journal of Algebra*, 364:148–154, 2012. `doi:10.1016/j.jalgebra.2012.04.014`.

24 Heribert Vollmer. *Introduction to Circuit Complexity*. Springer, Berlin, 1999.

25 Mariya Vorobets and Yaroslav Vorobets. On a free group of transformations defined by an automaton. *Geometriae Dedicata*, 124:237–249, 2007. `doi:10.1007/s10711-006-9060-5`.

A Trichotomy for Regular Trail Queries

Wim Martens
University of Bayreuth, Germany
wim.martens@uni-bayreuth.de

Matthias Niewerth
University of Bayreuth, Germany
matthias.niewerth@uni-bayreuth.de

Tina Trautner
University of Bayreuth, Germany
tina.trautner@uni-bayreuth.de

Abstract

Regular path queries (RPQs) are an essential component of graph query languages. Such queries consider a regular expression r and a directed edge-labeled graph G and search for paths in G for which the sequence of labels is in the language of r. In order to avoid having to consider infinitely many paths, some database engines restrict such paths to be *trails*, that is, they only consider paths without repeated edges. In this paper we consider the evaluation problem for RPQs under trail semantics, in the case where the expression is fixed. We show that, in this setting, there exists a trichotomy. More precisely, the complexity of RPQ evaluation divides the regular languages into the finite languages, the class $\mathsf{T}_{\mathsf{tract}}$ (for which the problem is tractable), and the rest. Interestingly, the tractable class in the trichotomy is larger than for the trichotomy for *simple paths*, discovered by Bagan et al. [5]. In addition to this trichotomy result, we also study characterizations of the tractable class, its expressivity, the recognition problem, closure properties, and show how the decision problem can be extended to the enumeration problem, which is relevant to practice.

2012 ACM Subject Classification Information systems → Query languages for non-relational engines; Information systems → Information retrieval query processing; Theory of computation → Problems, reductions and completeness; Theory of computation → Regular languages

Keywords and phrases Regular languages, query languages, path queries, graph databases, databases, complexity, trails, simple paths

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.7

Related Version A full version of the paper is available at [21], `https://arxiv.org/abs/1903.00226`.

Funding Deutsche Forschungsgemeinschaft (DFG), grant MA 4938/4-1.

Acknowledgements We thank the participants of Shonan meeting No. 138 (and Hassan Chafi in particular), who provided significant inspiration for the first paragraph in the Introduction, Jean-Éric Pin for pointing us to positive C_{ne}-varieties of languages, and Jean-Éric Pin and Luc Segoufin for their help with the proof of Proposition 3.13(b). Furthermore, we want to thank the anonymous reviewers of STACS 2020 for useful comments.

1 Introduction

Graph databases are a popular tool to model, store, and analyze data [25, 33, 27, 35, 12]. They are engineered to make the *connectedness of data* easier to analyze. This is indeed a desirable feature, since some of today's largest companies have become so successful because they understood how to use the connectedness of the data in their specific domain (e.g., Web search and social media). One aspect of graph databases is to bring tools for analyzing connectedness to the masses.

Regular path queries (RPQs) are a crucial component of graph databases, because they allow reasoning about arbitrarily long paths in the graph and, in particular, paths that are longer than the size of the query. A regular path query essentially consists of a regular expression r and is evaluated on a graph database which, for the purpose of this paper, we view as an edge-labeled directed graph G. When evaluated, the RPQ r searches for paths in G for which the sequence of labels is in the language of r. The return type of the query varies: whereas most academic research on RPQs [23, 6, 7, 20, 3] and SPARQL [34] focus on the first and last node of matching paths, Cypher [26] returns the entire paths. G-Core, a recent proposal by partners from industry and academia, sees paths as "first-class citizens" in graph databases [2].

In addition, there is a large variation on which types of paths are considered. Popular options are *all paths*, *simple paths*, *trails*, and *shortest paths*. Here, *simple paths* are paths without repeated nodes and *trails* are paths without repeated edges. Academic research has focused mostly on *all paths*, but Cypher 9 [26, 14], which is perhaps the most widespread graph database query language at the moment, uses *trails*. Since the trail semantics in graph databases has received virtually no attention from the research community yet, it is crucial that we improve our understanding.

In this paper, we study the *data complexity* of RPQ evaluation under trail semantics. That is, we study variants of RPQ evaluation in which the RPQ r is considered to be fixed. As such, the input of the problem only consists of an edge-labeled graph G and a pair (s, t) of nodes and we are asked if there exists a trail from s to t on which the sequence of labels matches r. One of our main results is a trichotomy on the RPQs for which this problem is in AC^0, NL-complete, or NP-complete, respectively. By $\mathsf{T}_{\mathsf{tract}}$, we refer to the class of tractable languages (assuming NP $\neq$ NL).

In order to increase our understanding of $\mathsf{T}_{\mathsf{tract}}$, we study several important aspects of this class of languages. A first set of results is on characterizations of $\mathsf{T}_{\mathsf{tract}}$ in terms of closure properties and syntactic and semantic conditions on their finite automata. In a second set of results, we therefore compare the expressiveness of $\mathsf{T}_{\mathsf{tract}}$ with yardstick languages such as $\mathbf{FO}^2[<]$, $\mathbf{FO}^2[<,+]$, $\mathbf{FO}[<]$ (or *aperiodic languages*), and $\mathsf{SP}_{\mathsf{tract}}$. The latter class, $\mathsf{SP}_{\mathsf{tract}}$, is the closely related class of languages for which the data complexity of RPQ evaluation under *simple path* semantics is tractable.[1] Interestingly, $\mathsf{T}_{\mathsf{tract}}$ is strictly larger than $\mathsf{SP}_{\mathsf{tract}}$ and includes languages outside $\mathsf{SP}_{\mathsf{tract}}$ such as a^*bc^* and $(ab)^*$ that are relevant in application scenarios in network problems, genomic datasets, and tracking provenance information of food products [29] and were recently discovered to appear in public query logs [10, 9]. Furthermore, every *single-occurrence regular expression* [8] is in $\mathsf{T}_{\mathsf{tract}}$, which can be a convenient guideline for users of graph databases, since *single-occurrence* (every alphabet symbol occurs at most once) is a very simple syntactical property. It is also popular in practice: we analyzed the 50 million RPQs found in the logs of [11] and discovered that over 99.8% of the RPQs are single-occurrence regular expressions.

We then study the recognition problem for $\mathsf{T}_{\mathsf{tract}}$, that is: given an automaton, does its language belong to $\mathsf{T}_{\mathsf{tract}}$? This problem is NL-complete (resp., PSPACE-complete) if the input automaton is a DFA (resp., NFA). We also treat closure under common operations such as union, intersection, reversal, quotients and morphisms.

We conclude by showing that also the *enumeration problem* is tractable for $\mathsf{T}_{\mathsf{tract}}$. By tractable, we mean that the paths that match the RPQ can be enumerated with only *polynomial delay* between answers. Technically, this means that we have to prove that we

[1] Bagan et al. [5] called the class $\mathsf{C}_{\mathrm{tract}}$, which stands for "tractable class". We distinguish between $\mathsf{SP}_{\mathsf{tract}}$ and $\mathsf{T}_{\mathsf{tract}}$ here to avoid confusion between simple paths and trails.

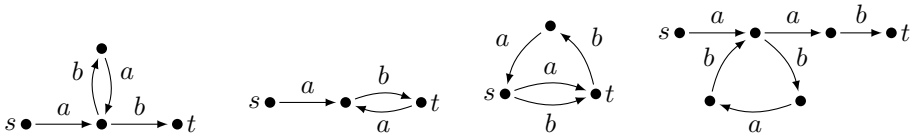


Figure 1 Directed, edge-labeled graphs that have a trail from s to t.

cannot only solve a decision variant of the RPQ evaluation problem, but we also need to find witnessing paths. We prove that the algorithms for the decision problems can be extended to return *shortest paths*. This insight can be combined with Yen's Algorithm [36] to give a polynomial delay enumeration algorithm.

Related Work. RPQs on graph databases have been studied since the end of the 80's and are now finding their way into commercial products. The literature usually considers the variant of RPQ evaluation where one is given a graph database G, nodes s, t, and an RPQ r, and then needs to decide if G has a path from s to t (possibly with loops) that matches r. For arbitrary and shortest paths, this problem is well-known to be tractable, since it boils down to testing intersection emptiness of two NFAs.

Mendelzon and Wood [23] studied the problem for simple paths, which are paths without node repetitions. They observed that the problem is already NP-complete for regular expressions a^*ba^* and $(aa)^*$. These two results rely heavily on the work of Fortune et al. [13] and LaPaugh and Papadimitriou [19].

Our work is most closely related to the work of Bagan et al. [5] who, like us, studied the complexity of RPQ evaluation where the RPQ is fixed. They proved a trichotomy for the case where the RPQ should only match simple paths. In this paper we will refer to this class as $\mathsf{SP}_{\mathsf{tract}}$, since it contains the languages for which the *simple path* problem is tractable, whereas we are interested in a class for *trails*. Martens and Trautner [22] refined this trichotomy of Bagan et al. [5] for *simple transitive expressions*, by analyzing the complexity where the input consists of both the expression and the graph.

Trails versus Simple Paths. We conclude with a note on the relationship between simple paths and trails. For many computational problems, the complexities of dealing with simple paths or trails are the same due to two simple reductions, namely: (1) constructing the line graph or (2) splitting each node into two, see for example Perl and Shiloach [28, Theorem 2.1 and 2.2]. As soon as we consider labeled graphs, the line graph technique still works, but not the nodes-splitting technique, because the labels on paths change. As a consequence, we know that finding trails is at most as hard as finding simple paths, but we do not know if it has the same complexity when we require that they match a certain RPQ r.

In this paper we show that the relationship is strict, assuming NL $\neq$ NP. An easy example is the language $(ab)^*$, which is NP-hard for simple paths [19, 23], but – assuming that a and b-edges are different – in NL for trails. This is because every path from s to t that matches $(ab)^*$ can be reduced to a trail from s to t that matches $(ab)^*$ by removing loops (in the path, not in the graph) that match $(ab)^*$ or $(ba)^*$. In Figure 1 we depict four small graphs, all of which have trails from s to t. (In the two rightmost graphs, there is exactly one path labeled $(ab)^*$, which is also a trail.)

2 Preliminaries

We use $[n]$ to denote the set of integers $\{1, \ldots, n\}$. By Σ we always denote a finite alphabet, i.e., a finite set of *symbols*. We always denote symbols by a, b, c, d and their variants, like a', a_1, b_1, etc. A *word* is a finite sequence $w = a_1 \cdots a_n$ of symbols.

We consider edge-labeled directed graphs $G = (V, E)$, where V is a finite set of nodes and $E \subseteq V \times \Sigma \times V$ is a set of (labeled) edges. A *path* p from node s to t is a sequence $(v_1, a_1, v_2)(v_2, a_2, v_3) \cdots (v_m, a_m, v_{m+1})$ with $v_1 = s$ and $v_{m+1} = t$ and such that $(v_i, a_i, v_{i+1}) \in E$ for each $i \in [m]$. By $|p|$ we denote the number of edges of a path. A path is a *trail* if all the edges (v_i, a_i, v_{i+1}) are different and a *simple path* if all the nodes v_i are different. (Notice that each simple path is a trail but not vice versa.) We denote $a_1 \cdots a_m$ by $\mathrm{lab}(p)$. Given a language $L \subseteq \Sigma^*$, path p *matches* L if $\mathrm{lab}(p) \in L$. For a subset $E' \subseteq E$, path p is E'-restricted if every edge of p is in E'. Given a trail p and two edges e_1 and e_2 in p, we denote the subpath of p from e_1 to e_2 by $p[e_1, e_2]$.

We define an NFA A to be a tuple $(Q, \Sigma, I, F, \delta)$ where Q is the finite set of states; $I \subseteq Q$ is a set of initial states; $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation; and $F \subseteq Q$ is the set of accepting states. Strongly connected components of (the graph of) A are simply called *components*. Unless noted otherwise, components will be non-trivial, i.e., containing at least one edge.

By $\delta(q, w)$ we denote the states reachable from state q by reading w. We denote by $q_1 \rightsquigarrow q_2$ that state q_2 is reachable from q_1. Finally, L_q denotes the set of all words accepted from q and $L(A) = \bigcup_{q \in I} L_q$ is the set of words accepted by A. For every state q, we denote by $\mathrm{Loop}(q)$ the set $\{w \in \Sigma^+ \mid \delta_L(q, w) = q\}$ of all non-empty words that allow to loop on q. For a word w and a language L, we define $wL = \{ww' \mid w' \in L\}$ and $w^{-1}L = \{w' \mid ww' \in L\}$.

A DFA is an NFA such that I is a singleton and for all $q \in Q, \sigma \in \Sigma$ $|\delta(q, \sigma)| \leq 1$. Let L be a regular language. We denote by $A_L = (Q_L, \Sigma, i_L, F_L, \delta_L)$ the (complete) minimal DFA for L and by N the number $|Q_L|$ of states. A language L is *aperiodic* if and only if $\delta_L(q, w^{N+1}) = \delta_L(q, w^N)$ for every state q and word w. Equivalently, L is aperiodic if and only if its minimal DFA of an aperiodic language L does not have simple cycles labeled w^k for $k > 1$ and $w \neq \varepsilon$. Thus, for "large enough n" we have: $uw^n v \in L$ iff $uw^{n+1}v \in L$. So, a language like $(aa)^*$ is not aperiodic (take $w = a$ and $k = 2$), but $(ab)^*$ is. (There are many characterizations of aperiodic languages [31].)

We study the *regular trail query (RTQ) problem* for a regular language L.

$\mathsf{RTQ}(L)$	
Given:	A graph $G = (V, E)$ and $(s, t) \in V \times V$.
Question:	Is there a trail from s to t that matches L?

A similar problem, which was studied by Bagan et al. [5], is the *RSPQ problem*. The $\mathsf{RSPQ}(L)$ problem asks if there exists a *simple path* from s to t that matches L.

3 The Tractable Class

In this section, we define and characterize a class of languages of which we will prove that it is exactly the class of regular languages L for which $\mathsf{RTQ}(L)$ is tractable (if NL $\neq$ NP).

3.1 Warm-Up: Downward Closed Languages

It is instructive to first discuss the case of downward closed languages. A language L is *downward closed* (DC) if it is closed under taking subsequences. That is, for every word $w = a_1 \cdots a_n \in L$ and every sequence $0 < i_1 < \cdots < i_k < n + 1$ of integers, we have that

$a_{i_1} \cdots a_{i_k} \in L$. Perhaps surprisingly, *downward closed languages are always regular* [16]. Furthermore, they can be defined by a clean class of regular expressions (which was shown by Jullien [18] and later rediscovered by Abdulla et al. [1]), which is defined as follows.

▶ **Definition 3.1.** *An* atomic expression *over* Σ *is an expression of the form* $(a + \varepsilon)$ *or of the form* $(a_1 + \cdots + a_n)^*$, *where* $a, a_1, \ldots, a_n \in \Sigma$. *A* product *is a (possibly empty) concatenation* $e_1 \cdots e_n$ *of atomic expressions* $e_1, \ldots, e_n$. *A* simple regular expression *is of the form* $p_1 + \cdots + p_n$, *where* $p_1, \ldots, p_n$ *are products.*

Another characterization is by Mendelzon and Wood [23], who show that a regular language L is downward closed if and only if its minimal DFA $A_L = (Q_L, \Sigma, i_L, F_L, \delta_L)$ exhibits the *suffix language containment property*, which says that if $\delta_L(q_1, a) = q_2$ for some symbol $a \in \Sigma$, then we have $L_{q_2} \subseteq L_{q_1}$.[2] Since this property is transitive, it is equivalent to require that $L_{q_2} \subseteq L_{q_1}$ for every state q_2 that is reachable from q_1.

▶ **Theorem 3.2** ([1, 16, 18, 23])**.** *The following are equivalent:*

(1) *L is a downward closed language.*

(2) *L is definable by a simple regular expression.*

(3) *The minimal DFA of L exhibits the suffix language containment property.*

Obviously, $\mathsf{RTQ}(L)$ is tractable for every downward closed language L, since it is equivalent to deciding if there exists a path from s to t that matches L. For the same reason, deciding if there is a *simple path* from s to t that matches L is also tractable for downward closed languages. However, there are languages that are not downward closed for which we show $\mathsf{RTQ}(L)$ to be tractable, such as a^*bc^* and $(ab)^*$. For these two languages, the simple path variant of the problem is intractable.

3.2 Main Definitions and Equivalence

The following definitions are the basis of the class of languages for which $\mathsf{RTQ}(L)$ is tractable.

▶ **Definition 3.3.** *An NFA A satisfies the* left-synchronized containment property *if there exists an $n \in \mathbb{N}$ such that the following implication holds for all $q_1, q_2 \in Q$:*

$$\text{If } q_1 \rightsquigarrow q_2 \text{ and if } w_1 \in \mathrm{Loop}(q_1), w_2 \in \mathrm{Loop}(q_2) \text{ with } w_1 = aw_1' \text{ and } w_2 = aw_2', \text{ then } w_2^n L_{q_2} \subseteq L_{q_1}.$$

Similarly, A satisfies the right-synchronized containment property *if the same condition holds with $w_1 = w_1'a$ and $w_2 = w_2'a$.*

We note that every downward closed language L satisfies the left-synchronized containment property.

▶ **Definition 3.4.** *A regular language L is* closed under left-synchronized power abbreviations *(resp.,* closed under right-synchronized power abbreviations*) if there exists an $n \in \mathbb{N}$ such that for all words $w_\ell, w_m, w_r \in \Sigma^*$ and all words $w_1 = aw_1'$ and $w_2 = aw_2'$ (resp., $w_1 = w_1'a$ and $w_2 = w_2'a$) we have that $w_\ell w_1^n w_m w_2^n w_r \in L$ implies $w_\ell w_1^n w_2^n w_r \in L$.*

[2] They restrict q_1, q_2 to be on paths from i_L to some state in F_L, but the property trivially holds for q_2 being a sink-state.

We note that Definition 3.4 is equivalent to requiring that there exists an $n \in \mathbb{N}$ such that the implication holds for all $i \geq n$. The reason is that, given $i > n$ and a word of the form $w_\ell w_1^i w_m w_2^i w_r$, we can write it as $w'_\ell w_1^n w_m w_2^n w'_r$ with $w'_\ell = w_\ell w_1^{i-n}$ and $w'_r = w_2^{i-n} w_r$, for which the implication holds by Definition 3.4.

Next, we show that all conditions defined in Definitions 3.3 and 3.4 are equivalent for DFAs.

▶ **Theorem 3.5.** *For a regular language L with minimal DFA A_L, the following are equivalent:*
(1) *A_L satisfies the left-synchronized containment property.*
(2) *A_L satisfies the right-synchronized containment property.*
(3) *L is closed under left-synchronized power abbreviations.*
(4) *L is closed under right-synchronized power abbreviations.*

In Theorem 4.1 we will show that, if NL $\neq$ NP, the languages L that satisfy the above properties are precisely those for which $\mathsf{RTQ}(L)$ is tractable. To simplify terminology, we will henceforth refer to this class as $\mathsf{T}_{\mathsf{tract}}$.

▶ **Definition 3.6.** *A regular language L belongs to $\mathsf{T}_{\mathsf{tract}}$ if L satisfies one of the equivalent conditions in Theorem 3.5.*

For example, $(ab)^*$ and $(abc)^*$ are in $\mathsf{T}_{\mathsf{tract}}$, whereas a^*ba^*, $(aa)^*$ and $(aba)^*$ are not. The following property immediately follows from the definition of $\mathsf{T}_{\mathsf{tract}}$.

▶ **Observation 3.7.** *Every regular expression for which each alphabet symbol under a Kleene star occurs at most once in the expression defines a language in $\mathsf{T}_{\mathsf{tract}}$.*

A special case of these expressions are those in which every alphabet symbol occurs at most once. These are known as *single-occurrence regular expressions (SORE)* [8]. SOREs were studied in the context of learning schema languages for XML [8], since they occur very often in practical schema languages.

3.3 A Syntactic Characterization

As we have seen before, regular expressions in which every symbol occurs at most once define languages in $\mathsf{T}_{\mathsf{tract}}$. We will define a similar notion on automata.

▶ **Definition 3.8.** *A component C of some NFA A is called* memoryless*, if for each symbol $a \in \Sigma$, there is at most one state q in C, such that there is a transition (p, a, q) with p in C.*

The following theorem provides (in a non-trivial proof that requires several steps) a syntactic condition for languages in $\mathsf{T}_{\mathsf{tract}}$. The syntactic condition is item (4) of the theorem, which we define after its statement. Condition (5) emposes an additional restriction on condition (4), and we later use it to prove that $\mathsf{T}_{\mathsf{tract}} \subseteq \mathbf{FO}^2[<, +]$.

▶ **Theorem 3.9.** *For a regular language L, the following properties are equivalent:*
(1) *$L \in \mathsf{T}_{\mathsf{tract}}$*
(2) *There exists an NFA A for L that satisfies the left-synchronized containment property.*
(3) *There exists an NFA A for L that satisfies the left-synchronized containment property and only has memoryless components.*
(4) *There exists a detainment automaton for L with consistent jumps.*
(5) *There exists a detainment automaton for L with consistent jumps and only memoryless components.*

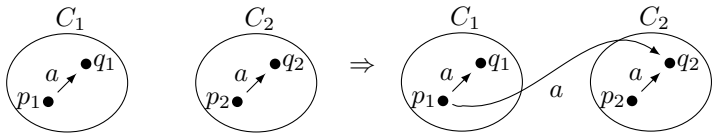


Figure 2 Consistent jump condition (simplified, i.e.: without preconditions, counter and update) used in Theorem 3.12. C_1 and C_2 are components (not necessarily different) such that C_2 is reachable from C_1.

We use *finite automata with counters or CNFAs* from Gelade et al. [15], that we slightly adapt to make the construction easier.[3] For convenience, we provide a full definition in Appendix A. Let A be a CNFA with one counter c. Initially, the counter has value 0. The automaton has transitions of the form $(q_1, a, P; q_2, U)$ where P is a precondition on c and U an update operation on c. For instance, the transition $(q_1, a, c = 5; q_2, c := c - 1)$ means: if A is in state q_1, reads a, and the value of c is five, then it can move to q_2 and decrease c by one. If we decrease a counter with value zero, its value remains zero. We denote the precondition that is always fulfilled by true.

We say that A is a *detainment automaton* if, for every component C of A:

- every transition inside C is of the form $(q_1, a, \mathsf{true}; q_2, c := c - 1)$;
- every transition that leaves C is of the form $(q_1, a, c = 0; q_2, c := k)$ for some $k \in \mathbb{N}$;[4]

Intuitively, if a detainment automaton enters a non-trivial component C, then it must stay there for at least some number of steps, depending on the value of the counter c. The counter c is decreased for every transition inside C and the automaton can only leave C once $c = 0$. We say that A has *consistent jumps* if, for every pair of components C_1 and C_2, if $C_1 \rightsquigarrow C_2$ and there are transitions $(p_i, a, \mathsf{true}; q_i, c := c - 1)$ inside C_i for all $i \in \{1, 2\}$, then there is also a transition $(p_1, a, P; q_2, U)$ for some $P \in \{\mathsf{true}, c = 0\}$ and some update U.[5] We note that C_1 and C_2 may be the same component. The consistent jump property is the syntactical counterpart of the left-synchronized containment property. The *memoryless* condition carries over naturally to CNFAs, ignoring the counter.

Proof sketch of Theorem 3.9. The implications (3) ⇒ (2) and (5) ⇒ (4) are trivial. We sketch the proofs of (1) ⇒ (5) ⇒ (3) and (4) ⇒ (2) ⇒ (1) below, establishing the theorem.

(1) ⇒ (5) uses a very technical construction that essentially exploits that – if the automaton stays in the same component for a long time – the reached state only depends on the last N^2 symbols read in the component. This is formalized in Lemma 4.3 and allows us to merge any pair of two states p, q which contradict that some component is memoryless. To preserve the language, words that stay in some component C for less than N^2 symbols have to be dealt with separately, essentially avoiding the component altogether. Finally, the left-synchronized containment property allows us to simply add transitions required to satisfy the consistent jumps property without changing the language.

(5) ⇒ (3) and (4) ⇒ (2): We convert a given CNFA to an NFA by simulating the counter (which is bounded) in the set of states. The consistent jump property implies the left-synchronized containment property on the resulting NFA. The property that all components are memoryless is preserved by the construction.

(2) ⇒ (1): One can show that the left-synchronized containment property is invariant under the powerset construction. ◀

[3] The adaptation is that we let counters decrease instead of increase. Furthermore, it only needs zero-tests.
[4] If q_2 is in a trivial component, then k should be 0 for the transition to be useful.
[5] The values of P and U depend on whether C_1 is the same as C_2 or not.

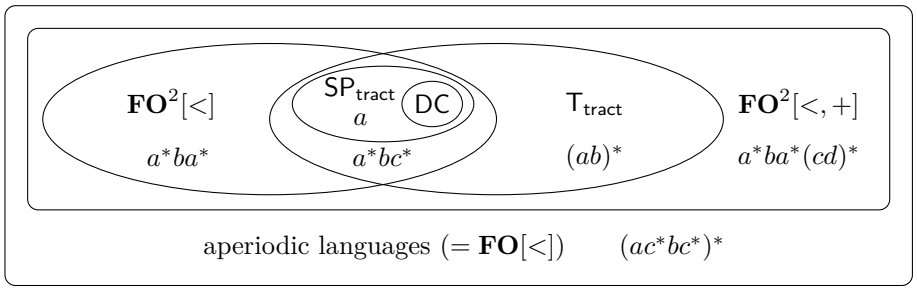


Figure 3 Expressiveness of subclasses of the aperiodic languages.

3.4 Comparison to Other Classes

We compare $\mathsf{T}_{\mathsf{tract}}$ to some closely related and yardstick languages to get an idea of its expressiveness. For example, every downward closed (DC) language is in $\mathsf{T}_{\mathsf{tract}}$, since $\mathsf{T}_{\mathsf{tract}}$ relaxed the containment property.

Bagan et al. [5] introduced the class $\mathsf{SP}_{\mathsf{tract}}$, which characterizes the class of regular languages L for which the *regular simple path query (RSPQ)* problem is tractable.

▶ **Theorem 3.10** (Theorem 2 in Bagan et al. [5]). *Let L be a regular language.*
(1) *If L is finite, then RSPQ$(L) \in AC^0$.*
(2) *If $L \in \mathsf{SP}_{\mathsf{tract}}$ and L is infinite, then RSPQ(L) is NL-complete.*
(3) *If $L \notin \mathsf{SP}_{\mathsf{tract}}$, then RSPQ$(L)$ is NP-complete.*
One characterization of $\mathsf{SP}_{\mathsf{tract}}$ is the following (Theorem 4 in [5]):

▶ **Definition 3.11.** $\mathsf{SP}_{\mathsf{tract}}$ *is the set of regular languages L such that there exists an $i \in \mathbb{N}$ for which the following holds: for all $w_\ell, w, w_r \in \Sigma^*$ and $w_1, w_2 \in \Sigma^+$ we have that, if $w_\ell w_1^i w w_2^i w_r \in L$, then $w_\ell w_1^i w_2^i w_r \in L$.*

From this definition it is easy to see that every language in $\mathsf{SP}_{\mathsf{tract}}$ is also in $\mathsf{T}_{\mathsf{tract}}$, since our definition imposes an extra "synchronizing" condition on w_1 and w_2, namely that they share the same first (or last) symbol (Definition 3.4). We now fully classify the expressiveness of $\mathsf{T}_{\mathsf{tract}}$ and $\mathsf{SP}_{\mathsf{tract}}$ compared to yardsticks as DC, $\mathbf{FO}^2[<]$, and $\mathbf{FO}^2[<,+]$ (see also Figure 3).

Here, $\mathbf{FO}^2[<]$ and $\mathbf{FO}^2[<,+]$ are the two-variable restrictions of $\mathbf{FO}[<]$ and $\mathbf{FO}[<,+]$ over words, respectively. By $\mathbf{FO}[<,+]$ we mean the first-order logic with unary predicates P_a for all $a \in \Sigma$ (denoting positions carrying the letter a) and the binary predicates $+1$ and $<$ (denoting the successor relation and the order relation among positions). $\mathbf{FO}[<]$ is $\mathbf{FO}[<,+]$ without the binary predicate $+1$.

▶ **Theorem 3.12.**
(a) $\mathsf{DC} \subsetneq \mathsf{SP}_{\mathsf{tract}} \subsetneq (\mathbf{FO}^2[<] \cap \mathsf{T}_{\mathsf{tract}})$
(b) $\mathsf{T}_{\mathsf{tract}} \subsetneq \mathbf{FO}^2[<,+]$
(c) $\mathsf{T}_{\mathsf{tract}}$ *and* $\mathbf{FO}^2[<]$ *are incomparable*
Since $\mathbf{FO}^2[<,+] \subsetneq \mathbf{FO}[<]$, we also have $\mathsf{T}_{\mathsf{tract}} \subsetneq \mathbf{FO}[<]$. Thus, every language in $\mathsf{T}_{\mathsf{tract}}$ is aperiodic.

Next, we show where $\mathsf{SP}_{\mathsf{tract}}$ and $\mathsf{T}_{\mathsf{tract}}$ are in the concatenation hierarchy (also known as Straubing-Thérien hierarchy) and the dot-depth hierarchy (also known as Brzozowski hierarchy).

▶ **Proposition 3.13.**
(a) $\mathsf{SP}_{\mathsf{tract}}$ *is in $\mathcal{V}_{3/2}$, the 3/2th level of the concatenation hierarchy.*
(b) *Every language L in $\mathsf{T}_{\mathsf{tract}} \cap \Sigma^+$ is in $\mathcal{B}_1$, the 1st level of the dot-depth hierarchy.*

Thus Proposition 3.13 implies that every language in $\mathsf{SP}_{\mathsf{tract}}$ can be described by a formula in $\Sigma_2[<]$ and every language in tractable language $\mathsf{T}_{\mathsf{tract}}$ by a boolean combination of formulas in $\Sigma_1[<, +, \min, \max]$, see Pin [30, Theorem 4.1].

4 The Trichotomy

This section is devoted to the proof of the following theorem.

▶ **Theorem 4.1.** *Let L be a regular language.*
(1) *If L is finite then $\mathsf{RTQ}(L) \in AC^0$.*
(2) *If $L \in \mathsf{T}_{\mathsf{tract}}$ and L is infinite, then $\mathsf{RTQ}(L)$ is NL-complete.*
(3) *If $L \notin \mathsf{T}_{\mathsf{tract}}$, then $\mathsf{RTQ}(L)$ is NP-complete.*

We will prove Theorem 4.1 only for simple graphs, but it extends to graphs with multi-edges, which are graphs with multiple edges with the same label between the same nodes. Equivalently, this can be seen as a variation of the problem where edges are accompanied with numbers that say how often they can be traversed. For example, we could say that e_1 may be used at most twice, while e_2 may be used at most 30 times. Here, the number of occurrences of edges can even be encoded in binary. We discuss this in Section 4.4.

4.1 Finite Languages

We now turn to proving Theorem 4.1. We start with Theorem 4.1(1). Clearly, we can express every finite language L as an FO-formula. Since we can also test in FO that no edge is used more than once, the graphs for which $\mathsf{RTQ}(L)$ holds are FO-definable. By Immerman [17], this implies that $\mathsf{RTQ}(L)$ is in AC^0.

4.2 Languages in $\mathsf{T}_{\mathsf{tract}}$

We now sketch the proof of Theorem 4.1(2). We note that we define several concepts (trail summary, local edge domains, admissible trails) that have a natural counterpart for simple paths in Bagan et al.'s proof of the trichotomy for simple paths [5, Theorem 2]. However, the underlying proofs of the technical lemmas are quite different. For instance, components of languages in $\mathsf{SP}_{\mathsf{tract}}$ behave similarly to A^* for some $A \subseteq \Sigma$, while components of languages in $\mathsf{T}_{\mathsf{tract}}$ are significantly more complex. Furthermore, the trichotomy for trails leads to a strictly larger class of tractable languages.

For the remainder of this section, we fix the constant $K = N^2$. We first describe the NL algorithm. Then we observe that, if the algorithm answers "yes", we can also output a shortest trail. We will show that in the case where L belongs to $\mathsf{T}_{\mathsf{tract}}$, we can identify a number of edges that suffice to check if the path is (or can be transformed into) a trail that matches L. This number of edges only depends on L and is therefore constant for the $\mathsf{RTQ}(L)$ problem. These edges will be stored in a *path summary*. We will define path summaries formally and explain how to use them to check whether a trail between the input nodes that matches L exists.

To this end, we need a few definitions. Let $A = (Q, \Sigma, I, F, \delta)$ be an NFA. We extend δ to paths, in the sense that we denote by $\delta(q, p)$ the set of states that A can reach from q after reading $\mathrm{lab}(p)$. For $q_0 \in Q$, we say that a *run from* q_0 of A over a path $p = (v_1, a_1, v_2)$ $(v_2, a_2, v_3) \cdots (v_m, a_m, v_{m+1})$ is a sequence $q_0 \cdots q_m$ of states such that $q_i \in \delta(q_{i-1}, a_i)$, for every $i \in [m]$. When A is a DFA and q_0 its initial state, we also simply call it *the run of A over p*.

▶ **Definition 4.2.** *Let $p = e_1 \cdots e_m$ be a path and $r = q_0 \cdots q_m$ the run of A_L over p. For a set C of states of A_L, we denote by left_C the first edge e_i with $q_{i-1} \in C$ and by right_C the last edge e_j with $q_j \in C$. A component C of A_L is a* long run component of p *if left_C and right_C are defined and $|p[\mathsf{left}_C, \mathsf{right}_C]| > K$.*

Next, we want to reduce the amount of information that we require for trails. To this end, we use the following synchronization property for A_L.

▶ **Lemma 4.3.** *Let $L \in \mathsf{T}_{\mathsf{tract}}$, let C be a component of A_L, let $q_1, q_2 \in C$, and let w be a word of length N^2. If $\delta_L(q_1, w) \in C$ and $\delta_L(q_2, w) \in C$, then $\delta_L(q_1, w) = \delta_L(q_2, w)$.*

The lemma motivates the use of *summaries*, which we define next.

▶ **Definition 4.4.** *Let* Cuts *denote the set of components of A_L and* $\mathsf{Abbrv} = \mathsf{Cuts} \times (V \times Q) \times E^K$. *A* component abbreviation *$(C, (v, q), e_K \cdots e_1) \in \mathsf{Abbrv}$ consists of a component C, a node v of G and state $q \in C$ to start from, and K edges $e_K \cdots e_1$. A trail π* matches *a component abbreviation, denoted $\pi \models (C, (v, q), e_K \cdots e_1)$, if $\delta_L(q, \pi) \in C$, it starts at v, and its suffix is $e_K \cdots e_1$. Given an arbitrary set of edges E', we write $\pi \models_{E'} (C, (v, q), e_K \cdots e_1)$ if $\pi \models (C, (v, q), e_K \cdots e_1)$ and all edges of π are from $E' \cup \{e_1, \ldots, e_K\}$. For convenience, we write $e \models_\emptyset e$.*

If p is a trail, then the summary *S_p of p is the sequence obtained from p by replacing, for each long run component C the subsequence $p[\mathsf{left}_C, \mathsf{right}_C]$ by the abbreviation $(C, (v, q), p_{\mathsf{suff}})$, where v is the source node of the edge left_C, q is the state in which A_L is immediately before reading left_C, and p_{suff} is the suffix of length K of $p[\mathsf{left}_C, \mathsf{right}_C]$.*

We note that the length of a summary is always bounded by $O(N^3)$, i.e., a constant that depends on L. Indeed, A_L has at most N components and, for each of them, we store at most $K + 3$ many things (namely, C, v, q, and K edges). Our goal is to find a summary S and replace all abbreviations with matching pairwise edge-disjoint trails which do not use any other edge in S, because this results in a trail that matches L. However, not every sequence of edges and abbreviations is a summary, because a summary needs to be obtained from a trail. So, we will work with candidate summaries instead.

▶ **Definition 4.5.** *A* candidate summary *S is a sequence of the form $S = \alpha_1 \cdots \alpha_m$ with $m \leq N$ where each α_i is either (1) an edge $e \in E$ or (2) an abbreviation $(C, (v, q), e_K \cdots e_1) \in \mathsf{Abbrv}$. Furthermore, all components and all edges appearing in S are distinct. A path p that is derived from S by replacing each $\alpha_i \in \mathsf{Abbrv}$ by a trail p_i such that $p_i \models \alpha_i$ is called a* completion *of the candidate summary S.*

The following corollary is immediate from the definitions and Lemma 4.3, as the lemma ensures that the state after reading p inside a component does not depend on the whole path but only on the labels of the last K edges, which are fixed.

▶ **Corollary 4.6.** *Let L be a language in $\mathsf{T}_{\mathsf{tract}}$. Let S be the summary of a trail p that matches L and let p' be a completion of S. Then, p' is a path that matches L.*

Together with the following lemma, Corollary 4.6 can be used to obtain an NL algorithm that gives us a completion of a summary S. The lemma heavily relies on other results on the structure of components in A_L.

▶ **Lemma 4.7.** *Let $L \in \mathsf{T}_{\mathsf{tract}}$, let $(C, (v, q), e_K \cdots e_1)$ be an abbreviation and $E' \subseteq E$. There exists an NL algorithm that outputs a shortest trail p such that $p \models_{E'} (C, (v, q), e_K \cdots e_1)$ if it exists and rejects otherwise.*

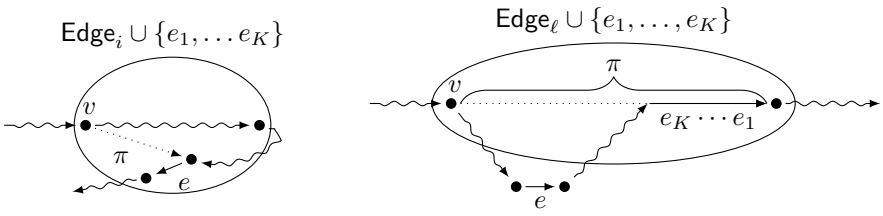


Figure 4 Sketch of case (1) and (2) in the proof of Lemma 4.10.

Using the algorithm of Lemma 4.7 we can, in principle, output a completion of S that matches L using nondeterministic logarithmic space. However, such a completion does not necessarily correspond to a trail. The reason is that, even though each trail p_C we guess for some abbreviation involving a component C is a trail, the trails for different components may not be disjoint. Therefore, we will define pairwise disjoint subsets of edges that can be used for the completion of the components.

The following definition fulfills the same purpose as the local domains on nodes in Bagan et al. [5, Definition 5]. Since our components can be more complex, we require extra conditions on the states (the $\delta_L(q, \pi) \in C$ condition).

▶ **Definition 4.8** (Local Edge Domains). *Let $S = \alpha_1 \cdots \alpha_k$ be a candidate summary and $E(S)$ be the set of edges appearing in S. We define the local edge domains $\mathsf{Edge}_i \subseteq E_i$ inductively for each i from 1 to k, where E_i are the remaining edges defined by $E_1 = E \setminus E(S)$ and $E_{i+1} = E_i \setminus \mathsf{Edge}_i$. If there is no trail p such that $p \models \alpha_i$ or if α_i is a simple edge, we define $\mathsf{Edge}_i = \emptyset$.*

Otherwise, let $\alpha_i = (C, (v, q), e_K \cdots e_1)$. We denote by m_i the minimal length of a trail p with $p \models_{E_i} \alpha_i$ and define Edge_i as the set of edges used by trails π that start at v, only use edges in E_i, are of length at most $m_i - K$, and satisfy $\delta_L(q, \pi) \in C$.

We note that the sets $E(S)$ and $(\mathsf{Edge}_i)_{i \in [k]}$ are always disjoint.

▶ **Definition 4.9** (Admissible Trail). *We say that a trail p is* admissible *if there exist a candidate summary $S = \alpha_1 \cdots \alpha_k$ and trails $p_1, \dots, p_k$ such that $p = p_1 \cdots p_k$ is a completion of S and $p_i \models_{\mathsf{Edge}_i} \alpha_i$ for every $i \in [k]$.*

We show that *shortest* trails that match L are always admissible. Thus, the existence of a trail is equivalent to the existence of an admissible trail.

▶ **Lemma 4.10.** *Let G and (s, t) be an instance for RTQ(L), with $L \in \mathsf{T}_{\mathsf{tract}}$. Then every shortest trail from s to t in G that matches L is admissible.*

Proof sketch. We assume towards a contradiction that there is a shortest trail p from s to t in G that matches L and is not admissible. That means there is some $\ell \in \mathbb{N}$, and an edge e used in p_ℓ with $e \notin \mathsf{Edge}_\ell$. There are two possible cases: (1) $e \in \mathsf{Edge}_i$ for some $i < \ell$ and (2) $e \notin \mathsf{Edge}_i$ for any i. In both cases, we construct a shorter trail p that matches L, which then leads to a contradiction. We depict the two cases in Figure 4. We construct the new trail by substituting the respective subtrail with π. ◀

So, if there is a solution to RTQ(L), we can find it by enumerating the candidate summaries and completing them using the local edge domains. We next prove that testing if an edge is in Edge_i can be done in logarithmic space. We will name this decision problem $P_{\mathsf{edge}}(L)$ and define it as follows:

$P_{\mathsf{edge}}(L)$	
Given:	A graph $G = (V, E)$, nodes s, t, a candidate summary S, an edge $e \in E$ and an integer i.
Question:	Is $e \in \mathsf{Edge}_i$?

▶ **Lemma 4.11.** *$P_{\mathsf{edge}}(L)$ is in NL for every $L \in \mathsf{T}_{\mathsf{tract}}$.*

With this, we can finally give an NL algorithm that decides whether a candidate summary can be completed to an admissible trail that matches L.

▶ **Lemma 4.12.** *Let $L \in \mathsf{T}_{\mathsf{tract}}$ and L be infinite. Then, RTQ(L) is NL-complete.*

▶ **Corollary 4.13.** *Let $L \in \mathsf{T}_{\mathsf{tract}}$, G be a graph, and s, t be nodes in G. If there exists a trail from s to t that matches L, then we can output a shortest such trail in polynomial time (and in nondeterministic logarithmic space).*

4.3 Languages not in $\mathsf{T}_{\mathsf{tract}}$

The proof of Theorem 4.1(3) is by reduction from the following NP-complete problem:

TwoEdgeDisjointPaths	
Given:	A language L, a graph $G = (V, E)$, and two pairs of nodes (s_1, t_1), (s_2, t_2).
Question:	Are there two paths p_1 from s_1 to t_1 and p_2 from s_2 to t_2 such that p_1 and p_2 are edge-disjoint?

The proof is very close to the corresponding proof for simple paths by Bagan et al. [5, Lemma 2] (which is a reduction from the two vertex-disjoint paths problem).

4.4 Extension to Multigraphs

We believe that Theorem 4.1 can be extended to graphs with multi-edges. For each edge e, denote by max_e the number of occurrences of e in the multigraph. First, consider the case where L is a finite language. Let m be the length of longest word in L. Notice that m is a constant, since it only depends on L. Every edge can be used at most m times in paths that match L, so we can check in FO whether an edge e is used at most $n_e = min\{m, max_e\}$ times. The rest of the argument is analogous to Section 4.1.

We now turn to languages in $\mathsf{T}_{\mathsf{tract}}$. The length of the candidate summaries S (Definition 4.5) only depends on L and is therefore constant. Instead of testing whether all edges appearing in S are distinct, we have to check if they occur at most the maximal number of times. (Therefore, listing all candidate summaries is still in $O((\log |G|)^N)$, and thus in polynomial time.) For the local edge domains (Definition 4.8), we define E_1 as an ordinary graph, i.e., non-multigraph, containing all edges that have not exhausted their maximal number of occurrences in S already. With this graph we can continue just as for ordinary graphs.

5 Recognition and Closure Properties

The following theorem establishes the complexity of deciding if a regular language is in $\mathsf{T}_{\mathsf{tract}}$.

▶ **Theorem 5.1.** *Testing whether a regular language L belongs to $\mathsf{T}_{\mathsf{tract}}$ is*
(1) *NL-complete if L is given by a DFA and*
(2) *PSPACE-complete if L is given by an NFA or by a regular expression.*

We wondered if, similarly to Theorem 3.2, it could be the case that languages closed under left-synchronized power abbreviations are always regular, but this is not the case. For example, the (infinite) Thue-Morse word [32, 24] has no subword that is a cube (i.e., no subword of the form w^3) [32, Satz 6]. The language containing all prefixes of the Thue-Morse word thus trivially is closed under left-synchronized power abbreviations (with $i = 3$), yet it is not regular.

We now give some closure properties of $\mathsf{SP}_{\mathsf{tract}}$ and $\mathsf{T}_{\mathsf{tract}}$.

▶ **Lemma 5.2.** *Both classes* $\mathsf{SP}_{\mathsf{tract}}$ *and* $\mathsf{T}_{\mathsf{tract}}$ *are closed under* (i) *finite unions,* (ii) *finite intersections,* (iii) *reversal,* (iv) *left and right quotients,* (v) *inverses of non-erasing morphisms,* (vi) *removal and addition of individual strings.*

This lemma implies that $\mathsf{SP}_{\mathsf{tract}}$ and $\mathsf{T}_{\mathsf{tract}}$ each are a positive C_{ne}-variety of languages, i.e., a positive variety of languages that is closed under inverse non-erasing homomorphisms.

▶ **Lemma 5.3.** *The classes* $\mathsf{SP}_{\mathsf{tract}}$ *and* $\mathsf{T}_{\mathsf{tract}}$ *are not closed under complement.*

Proof. Let $\Sigma = \{a, b\}$. The language of the expression b^* clearly is in $\mathsf{SP}_{\mathsf{tract}}$ and $\mathsf{T}_{\mathsf{tract}}$. Its complement is the language L containing all words with at least one a. It can be described by the regular expression $\Sigma^* a \Sigma^*$. Since $b^i a b^i \in L$ for all i, but $b^i b^i \notin L$ for any i, the language L is neither in $\mathsf{SP}_{\mathsf{tract}}$ nor in $\mathsf{T}_{\mathsf{tract}}$. ◀

It is an easy consequence of Lemma 5.2 (vi) that there do not exist best lower or upper approximations for regular languages outside $\mathsf{SP}_{\mathsf{tract}}$ or $\mathsf{T}_{\mathsf{tract}}$.

▶ **Corollary 5.4.** *Let* $\mathcal{C} \in \{\mathsf{SP}_{\mathsf{tract}}, \mathsf{T}_{\mathsf{tract}}\}$. *For every regular language* L *such that* $L \notin \mathcal{C}$ *and*

- *for every upper approximation* L'' *of* L *(i.e.,* $L \subsetneq L''$*) with* $L'' \in \mathcal{C}$ *it holds that there exists a language* $L' \in \mathcal{C}$ *with* $L \subsetneq L' \subsetneq L''$*;*
- *for every lower approximation* L'' *of* L *(i.e.,* $L'' \subsetneq L$*) it holds that there exists a language* $L' \in \mathcal{C}$ *with* $L'' \subsetneq L' \subsetneq L$.

The corollary implies that Angluin-style learning of languages in $\mathsf{SP}_{\mathsf{tract}}$ or $\mathsf{T}_{\mathsf{tract}}$ is not possible. However, learning algorithms for single-occurrence regular expressions (SOREs) exist [8] and can therefore be useful for an important subclass of $\mathsf{T}_{\mathsf{tract}}$.

6 Enumeration

In this section we state that – using the algorithm from Theorem 4.1 – the enumeration result from [36] transfers to the setting of enumerating trails matching L.

▶ **Theorem 6.1.** *Let* L *be a regular language,* G *be a graph and* (s, t) *a pair of nodes in* G. *If NL* $\neq$ *NP, then one can enumerate trails from* s *to* t *that match* L *in polynomial delay in data complexity if and only if* $L \in \mathsf{T}_{\mathsf{tract}}$.

Proof sketch. The algorithm is an adaptation of Yen's algorithm [36] that enumerates the k shortest simple paths for some given number k, similar to what was done by Martens and Trautner [22, Theorem 18]. It uses the algorithm from Corollary 4.13 as a subprocedure. ◀

7 Conclusions and Lessons Learned

We have defined the class $\mathsf{T}_{\mathsf{tract}}$ of regular languages L for which finding trails in directed graphs that are labeled with L is tractable iff NL $\neq$ NP. We have investigated $\mathsf{T}_{\mathsf{tract}}$ in depth in terms of closure properties, characterizations, and the recognition problem, also touching upon the closely related class $\mathsf{SP}_{\mathsf{tract}}$ (for which finding *simple paths* is tractable) when relevant.

In our view, graph database manufacturers can have the following tradeoffs in mind concerning simple path ($\mathsf{SP}_{\mathsf{tract}}$) and trail semantics ($\mathsf{T}_{\mathsf{tract}}$) in database systems:

- $\mathsf{SP}_{\mathsf{tract}} \subsetneq \mathsf{T}_{\mathsf{tract}}$, that is, there are strictly more languages for which finding regular paths under trail semantics is tractable than under simple path semantics. Some of the languages in $\mathsf{T}_{\mathsf{tract}}$ but outside $\mathsf{SP}_{\mathsf{tract}}$ are of the form $(ab)^*$ or a^*bc^*, which were found to be relevant in several application scenarios involving network problems, genomic datasets, and tracking provenance information of food products [29] and appear in query logs [10, 9].
- Both $\mathsf{SP}_{\mathsf{tract}}$ and $\mathsf{T}_{\mathsf{tract}}$ can be syntactically characterized but, currently, the characterization for $\mathsf{SP}_{\mathsf{tract}}$ (Section 3.5 in [5]) is simpler than the one for $\mathsf{T}_{\mathsf{tract}}$. This is due to the fact that connected components for automata for languages in $\mathsf{T}_{\mathsf{tract}}$ can be much more complex than for automata for languages in $\mathsf{SP}_{\mathsf{tract}}$.
- On the other hand, the *single-occurrence* condition, i.e., each alphabet symbol occurs at most once, is a sufficient condition for regular expressions to be in $\mathsf{T}_{\mathsf{tract}}$. This condition is trivial to check and also captures languages outside $\mathsf{SP}_{\mathsf{tract}}$ such as $(ab)^*$ and a^*bc^*. Moreover, the condition seems to be useful: we analyzed the 50 million RPQs found in the logs of [11] and discovered that over 99.8% of the RPQs are single-occurrence.
- In terms of closure properties, learnability, or complexity of testing if a given regular language belongs to $\mathsf{SP}_{\mathsf{tract}}$ or $\mathsf{T}_{\mathsf{tract}}$, the classes seem to behave the same.
- The tractability for the decision version of RPQ evaluation can be lifted to the enumeration problem, in which case the task is to output matching paths with only a polynomial delay between answers.

As an open question remains the trichotomy for 2RPQs, that is, when we allow RPQs to follow a directed edge also in its reverse direction. We briefly discuss why this is challenging. Let us denote by $\hat{}a$ the backward navigation of an edge labeled a. Then, the case of ordinary RPQs can be seen as a special case of 2RPQs on directed graphs: it only has bidirectional navigation of the form $(a + \hat{}a)$. It has been open problem since 1991 whether evaluating $(aaa)^*$ on undirected graphs is in P or NP-complete [4].

References

1 Parosh Aziz Abdulla, Aurore Collomb-Annichini, Ahmed Bouajjani, and Bengt Jonsson. Using forward reachability analysis for verification of lossy channel systems. *Formal Methods in System Design*, 25(1):39–65, 2004.

2 Renzo Angles, Marcelo Arenas, Pablo Barceló, Peter A. Boncz, George H. L. Fletcher, Claudio Gutierrez, Tobias Lindaaker, Marcus Paradies, Stefan Plantikow, Juan F. Sequeda, Oskar van Rest, and Hannes Voigt. G-CORE: A core for future graph query languages. In *International Conference on Management of Data (SIGMOD)*, pages 1421–1432, 2018.

3 Marcelo Arenas, Sebastián Conca, and Jorge Pérez. Counting beyond a yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. In *International Conference on World Wide Web (WWW)*, pages 629–638, 2012.

4 Esther M. Arkin, Christos H. Papadimitriou, and Mihalis Yannakakis. Modularity of cycles and paths in graphs. *J. ACM*, 38(2):255–274, 1991.

5 Guillaume Bagan, Angela Bonifati, and Benoît Groz. A trichotomy for regular simple path queries on graphs. In *Symposium on Principles of Database Systems (PODS)*, pages 261–272, 2013.

6 Pablo Barceló. Querying graph databases. In *Symposium on Principles of Database Systems (PODS)*, pages 175–188, 2013.

7 Pablo Barceló, Leonid Libkin, and Juan L. Reutter. Querying graph patterns. In *PODS*, pages 199–210. ACM, 2011.

8 Geert Jan Bex, Frank Neven, Thomas Schwentick, and Stijn Vansummeren. Inference of concise regular expressions and dtds. *ACM Trans. Database Syst.*, 35(2):11:1–11:47, 2010.

9 Angela Bonifati, Wim Martens, and Thomas Tim. Navigating the maze of wikidata query logs. In *The Web Conference (WWW)*. ACM, 2019. To appear.

10 Angela Bonifati, Wim Martens, and Thomas Timm. An analytical study of large SPARQL query logs. *PVLDB*, 11(2):149–161, 2017.

11 Angela Bonifati, Wim Martens, and Thomas Timm. DARQL: deep analysis of SPARQL queries. In *WWW (Companion Volume)*, pages 187–190. ACM, 2018.

12 Dbpedia. `https://wiki.dbpedia.org`.

13 Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science (TCS)*, 10(2):111–121, 1980.

14 Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property graphs. In *SIGMOD Conference*, pages 1433–1445. ACM, 2018.

15 Wouter Gelade, Marc Gyssens, and Wim Martens. Regular expressions with counting: Weak versus strong determinism. *SIAM J. Comput.*, 41(1):160–190, 2012.

16 Leonard H. Haines. On free monoids partially ordered by embedding. *Journal of Combinatorial Theory*, 6(1):94–98, 1969.

17 Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, 1988.

18 Pierre Jullien. *Contribution à l'étude des types d'ordres dispersés*. PhD thesis, Universite de Marseille, 1969.

19 Andrea S. LaPaugh and Christos H. Papadimitriou. The even-path problem for graphs and digraphs. *Networks*, 14(4):507–513, 1984.

20 Katja Losemann and Wim Martens. The complexity of regular expressions and property paths in SPARQL. *ACM Transactions on Database Systems*, 38(4):24:1–24:39, 2013.

21 Wim Martens, Matthias Niewerth, and Tina Trautner. A trichotomy for regular trail queries. *CoRR*, abs/1903.00226, 2019. `arXiv:1903.00226`.

22 Wim Martens and Tina Trautner. Evaluation and enumeration problems for regular path queries. In *ICDT*, volume 98 of *LIPIcs*, pages 19:1–19:21. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

23 Alberto O. Mendelzon and Peter T. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6):1235–1258, December 1995.

24 Harold Marston Morse. Recurrent geodesics on a surface of negative curvature. *Transactions of the American Mathematical Society*, 22(1):84–100, January 1921. `doi:10.2307/1988844`.

25 Neo4j. `https://neo4j.com/`.

26 Cypher query language reference, version 9, mar. 2018. `https://github.com/opencypher/openCypher/blob/master/docs/openCypher9.pdf`.

27 Oracle spatial and graph. `https://www.oracle.com/database/technologies/spatialandgraph.html`.

28 Yehoshua Perl and Yossi Shiloach. Finding two disjoint paths between two pairs of vertices in a graph. *J. ACM*, 25(1):1–9, 1978.

29 Neo4J Petra Selmer. Personal communication.
30 Jean-Éric Pin. The dot-depth hierarchy, 45 years later. In *The Role of Theory in Computer Science*, pages 177–202. World Scientific, 2017.
31 Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.
32 Axel Thue. *Über unendliche Zeichenreihen*. Skrifter udg. af Videnskabs-Selskabet i Christiania : 1. Math.-Naturv. Klasse. Dybwad [in Komm.], 1906.
33 Tigergraph. `https://www.tigergraph.com/`.
34 SPARQL 1.1 query language. `https://www.w3.org/TR/sparql11-query/`, 2013. World Wide Web Consortium.
35 Wikidata. `https://www.wikidata.org/`.
36 Jin Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

A Background on NFAs with Counters

We recall the definition of counter NFAs from Gelade et al. [15]. We introduce a minor difference, namely that counters count down instead of up, since this makes our construction easier to describe. Furthermore, since our construction only requires a single counter, zero tests, and setting the counter to a certain value, we immediately simplify the definition to take this into account.

Let c be a *counter variable*, taking values in $\mathbb{N}$. A *guard* on c is a statement γ of the form true or $c = 0$. We denote by $c \models \gamma$ that c satisfies the guard γ. In the case where γ is true, this is trivially fulfilled and, in the case where γ is $c = 0$, this is fulfilled if c equals 0. By G we denote the set of guards on c. An *update* on c is a statement of the form $c := c - 1$, $c := c$, or $c := k$ for some constant $k \in \mathbb{N}$. By U we denote the set of updates on c.

▶ **Definition A.1.** *A* non-deterministic counter automaton *(CNFA) with a single counter is a 6-tuple $A = (Q, I, c, \delta, F, \tau)$ where Q is the finite set of states; $I \subseteq Q$ is a set of initial states; c is a counter variable; $\delta \subseteq Q \times \Sigma \times G \times Q \times U$ is the transition relation; and $F \subseteq Q$ is the set of accepting states. Furthermore, $\tau \in \mathbb{N}$ is a constant such that every update is of the form $c := k$ with $k \leq \tau$.*

Intuitively, A can make a transition $(q, a, \gamma; q', \pi)$ whenever it is in state q, reads a, and $c \models \gamma$, i.e., guard γ is true under the current value of c. It then updates c according to the update π, in a way we explain next, and moves into state q'. To explain the update mechanism formally, we introduce the notion of configuration. A *configuration* is a pair (q, ℓ) where $q \in Q$ is the current state and $\ell \in \mathbb{N}$ is the value of c. Finally, an update π defines a function $\pi : \mathbb{N} \to \mathbb{N}$ as follows. If $\pi = (c := k)$ then $\pi(\ell) = k$ for every $\ell \in \mathbb{N}$. If $\pi = (c := c - 1)$ then $\pi(\ell) = \max(\ell - 1, 0)$. Otherwise, i.e., if $\pi = (c := c)$, then $\pi(\ell) = \ell$. So, counters never become negative.

An *initial configuration* is $(q_0, 0)$ with $q_0 \in I$. A configuration (q, ℓ) is *accepting* if $q \in F$ and $\ell = 0$. A configuration $\alpha' = (q', \ell')$ *immediately follows* a configuration $\alpha = (q, \ell)$ by reading $a \in \Sigma$, denoted $\alpha \to_a \alpha'$, if there exists $(q, a, \gamma; q', \pi) \in \delta$ with $c \models \gamma$ and $\ell' = \pi(\ell)$.

For a string $w = a_1 \cdots a_n$ and two configurations α and α', we denote by $\alpha \Rightarrow_w \alpha'$ that $\alpha \to_{a_1} \cdots \to_{a_n} \alpha'$. A configuration α is *reachable* if there exists a string w such that $\alpha_0 \Rightarrow_w \alpha$ for some initial configuration α_0. A string w is *accepted* by A if $\alpha_0 \Rightarrow_w \alpha_f$ where α_0 is an initial configuration and α_f is an accepting configuration. We denote by $L(A)$ the set of strings accepted by A.

It is easy to see that CNFA accept precisely the regular languages. (Due to the value τ, counters are always bounded by a constant.)

Descriptive Complexity on Non-Polish Spaces

Antonin Callard
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
antonin.callard@loria.fr

Mathieu Hoyrup
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
mathieu.hoyrup@inria.fr

Abstract

Represented spaces are the spaces on which computations can be performed. We investigate the descriptive complexity of sets in represented spaces. We prove that the standard representation of a countably-based space preserves the effective descriptive complexity of sets. We prove that some results from descriptive set theory on Polish spaces extend to arbitrary countably-based spaces. We study the larger class of coPolish spaces, showing that their representation does not always preserve the complexity of sets, and we relate this mismatch with the sequential aspects of the space. We study in particular the space of polynomials.

2012 ACM Subject Classification Theory of computation → Turing machines; Mathematics of computing → Point-set topology

Keywords and phrases Represented space, Computable analysis, Descriptive set theory, CoPolish spaces

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.8

Acknowledgements We want to thank Takayuki Kihara, Arno Pauly and Victor Selivanov for useful discussions on this topic.

1 Introduction

Several branches of theoretical computer science, such as semantics of programming language, domain theory or computability theory, have demonstrated the intimate relationship between computation and topology, one of the simplest manifestations of this being that computable functions are continuous. Descriptive Set Theory (DST) and its effective version provide a natural framework in which the interaction between computations and topology can be fully studied.

However, a large part of DST focuses on Polish spaces, leaving aside many topological spaces relevant to theoretical computer science. A first extension to ω-continuous domains was developed by Selivanov [13]. A further extension allowing a unification with Polish spaces was achieved by de Brecht [2] who introduced the class of quasi-Polish spaces, which can be thought of as the largest class of countably-based spaces on which DST extends. Still, many topological spaces which are meaningful to theoretical computer science fall outside the class: for instance the Kleene-Kreisel spaces important in higher-order computation theory [8], the recently introduced coPolish spaces, which admit a well-behaved computational complexity theory [12], more generally the represented spaces in computable analysis.

Therefore, there is a need to extend DST to more general topological spaces. Such an extension was proposed and initiated in [10] for represented spaces. Some negative results were obtained in [5] for spaces of Kleene-Kreisel functionals. With a different approach, a study of quotients of countably-based spaces (QCB-spaces) was done in [4]. It would be interesting to explore the relationship between DST on represented spaces and equivalence relations on standard Borel spaces, a representation naturally inducing an equivalence relation on names.

In this paper, we investigate general countably-based spaces and the class of coPolish spaces, in particular the space $\mathbb{R}[X]$ of real polynomials. We investigate the validity of some classical theorems from DST on those topological spaces, identifying when they extend and when they do not.

All these spaces have natural representations, which are essential to the development of the theory. The role of representations in this work can be interpreted in two different ways:

1. Representations can simply be seen as a tool in the study of certain topological spaces. Many results can be stated without any mention of representations, and of a purely topological interest.
2. Representations can also be seen as providing a structure alternative to topology, inducing in particular different measures of descriptive complexity of sets. As we briefly mention at the end of the paper, several results from DST that fail on some topological spaces can be recovered if one uses the notions induced by the representation rather than the topology. This interpretation supports the viewpoint adopted in [10], suggesting a development of DST in the category of represented spaces rather than topological spaces (which makes no difference when restricting to Polish or quasi-Polish spaces where these two approaches are equivalent).

We think that it is too early to choose between these two viewpoints, and that they both have their merits.

We now give an overview of the results.

In a topological space, the descriptive complexity of a set A measures the complexity of expressing A in terms of open sets. In a represented space (X, δ_X), where $\delta_X :\subseteq \mathcal{N} \to X$ is a partial surjective function from the Baire space $\mathcal{N} = \mathbb{N}^{\mathbb{N}}$ ($p \in \mathrm{dom}(\delta_X)$ being a code for the point $\delta_X(p) \in X$, to be given as input to a Turing machine), an alternative approach is to measure the descriptive complexity of the set of codes of points of A, which measures the complexity of testing membership of points in the set, when points are given by codes.

Represented spaces have a canonical topology (the final topology of the representation), so two competing measures of complexity are available on these spaces.

▶ **Problem 1.** *When do the notions of descriptive complexity induced by a representation and its corresponding topology coincide?*

We show that they coincide on countably-based spaces in a uniform computable way (effectivizing a result of de Brecht [2]) and that they can differ on other spaces, including $\mathbb{R}[X]$. In the class of coPolish spaces, we characterize the spaces on which the two notions of complexity coincide (at least in the low complexity levels) as the class of Fréchet-Urysohn spaces (the spaces in which closures and sequential closures coincide). It suggests that the mismatch between the topology and the representation is related to the difference between the topological and sequential aspects of the space, and that the complexity induced by the representation reflects the sequential rather than topological aspects of the space.

▶ **Problem 2.** *In a topological space, how to establish a lower bound on the descriptive complexity of a given set?*

On a Polish space, in order to prove that a set A does not belong to a descriptive complexity class Γ, it is necessary and sufficient to prove that A is $\check{\Gamma}$-hard (when Γ is not self-dual, i.e. when $\check{\Gamma} \neq \Gamma$). We show that for complexity classes below $\underset{\sim}{\mathbf{\Delta}}^0_2$, more precisely for the classes of the difference hierarchy, this result is surprisingly valid on *arbitrary* countably-based spaces. However it becomes false on $\mathbb{R}[X]$, precisely because of Problem 1: the hardness of a set is not a measure of its topological complexity, but of the complexity of its preimage

under the representation. Therefore, we need new techniques to measure the topological complexity of a set in non-countably-based spaces. We develop a criterion, which is necessary and sufficient, to prove that a set does not belong to a class below $\underset{\sim}{\mathbf{\Delta}}^0_2$, in any topological space.

Finally, we investigate the validity, outside Polish spaces, of a famous result from DST, the Hausdorff-Kuratowski Theorem. In its simplest form, it states that the class $\underset{\sim}{\mathbf{\Delta}}^0_2$ can be classified exhaustively using the difference hierarchy (the general result also considers classes $\underset{\sim}{\mathbf{\Delta}}^0_\alpha$).

▶ **Problem 3.** *On which topological spaces does the Hausdorff-Kuratowski Theorem hold?*

Using the previous results, we show that the Hausdorff-Kuratowski Theorem holds on a countably-based space if and only if that space does not contain any $\underset{\sim}{\mathbf{\Delta}}^0_2$-complete set. We show that the Hausdorff-Kuratowski Theorem does not hold on the topological space $\mathbb{R}[X]$. However, when seeing $\mathbb{R}[X]$ as a represented space and measuring the complexity of sets according to the representation rather than the topology, the Hausdorff-Kuratowski Theorem becomes true.

In Section 2 we give a minimalist summary of the background needed to state and prove our results. In Section 3 we present our results concerning Problem 2, which will be needed to give answers to Problem 1 in Section 4. In Section 5, we investigate the class of coPolish spaces, which includes spaces that are not countably-based.

2 Background

The Baire space is $\mathcal{N} = \mathbb{N}^{\mathbb{N}}$ with the product topology generated by the cylinders $[\sigma]$, with $\sigma \in \mathbb{N}^*$. A *represented space* is a pair (X, δ_X) where X is a set and $\delta_X :\subseteq \mathcal{N} \to X$ is onto. A *realizer* of a function $f : (X, \delta_X) \to (Y, \delta_Y)$ is any function $F : \mathrm{dom}(\delta_X) \to \mathrm{dom}(\delta_Y)$ such that $f \circ \delta_X = \delta_Y \circ F$. f is *computable* if it has a computable realizer.

A represented space (X, δ_X) is *admissible* if the continuously realizable functions $f :\subseteq \mathcal{N} \to X$ are precisely the continuous functions (for the final topology of δ_X).

An effective countably-based space is a countably-based topological space X with a numbered basis of the topology $(B_i)_{i\in\mathbb{N}}$ such that intersection of basic open sets is computable: $B_i \cap B_j = \bigcup_{k\in W_{f(i,j)}} B_k$ for some computable $f : \mathbb{N}^2 \to \mathbb{N}$, where $(W_e)_{e\in\mathbb{N}}$ is an effective enumeration of the c.e. subsets of $\mathbb{N}$. The standard representation, which is admissible, is defined by representing $x \in X$ by any listing of the set $\{i \in \mathbb{N} : x \in B_i\}$. A particularly useful property of these spaces is that the standard representation is *effectively open*: $\delta([\sigma]) = \bigcup_{i\in W_{g(\sigma)}} B_i$, for some computable g. The class $\Sigma^0_1(X)$ of effective open sets consists of c.e. unions of basic open sets. More details can be found in [17, 9].

2.1 Hierarchies on topological spaces

2.1.1 Borel hierarchy

We should emphasize that although we work in represented spaces, we are using the topology to define the descriptive complexity classes. It contrasts with the approach developed in [10] in which the complexity of a set is defined as the descriptive complexity of the corresponding set of names, i.e. the preimage of the set under the representation. Our general goal is precisely to compare the topological complexity of sets with the complexity of its corresponding set of names.

The Borel hierarchy, usually defined on Polish spaces, can be extended immediately to any topological space X, with a slight modification to handle correctly the non-Hausdorff spaces, in which open sets are not always unions of closed sets [13].

- $\underset{\sim}{\mathbf{\Sigma}}^0_1(X)$ is the class of open sets,
- For $1 < \alpha < \omega_1$, $A \in \underset{\sim}{\mathbf{\Sigma}}^0_\alpha(X)$ if $A = \bigcup_{i\in\mathbb{N}} A_i \setminus B_i$ where $A_i, B_i \in \underset{\sim}{\mathbf{\Sigma}}^0_{\alpha_i}$ with $\alpha_i < \alpha$.

We also define $\underset{\sim}{\mathbf{\Pi}}^0_\alpha(X)$ as the class of complements of sets in $\underset{\sim}{\mathbf{\Sigma}}^0_\alpha(X)$, and $\underset{\sim}{\mathbf{\Delta}}^0_\alpha(X) = \underset{\sim}{\mathbf{\Sigma}}^0_\alpha(X) \cap \underset{\sim}{\mathbf{\Pi}}^0_\alpha(X)$.

2.1.2 Difference hierarchy

Let X be a topological space. The difference hierarchy $(\underset{\sim}{\mathbf{D}}_\alpha(\underset{\sim}{\mathbf{\Sigma}}^0_\beta(X)))_{1\le\alpha<\omega_1}$ based on $\underset{\sim}{\mathbf{\Sigma}}^0_\beta(X)$ is defined by transfinite induction as follows [13]:

- $\underset{\sim}{\mathbf{D}}_1(\underset{\sim}{\mathbf{\Sigma}}^0_\beta(X)) = \underset{\sim}{\mathbf{\Sigma}}^0_\beta(X)$,
- $A \in \underset{\sim}{\mathbf{D}}_{\alpha+1}(\underset{\sim}{\mathbf{\Sigma}}^0_\beta(X))$ if $A = U \setminus B$ where $U \in \underset{\sim}{\mathbf{\Sigma}}^0_\beta(X)$ and $B \in \underset{\sim}{\mathbf{D}}_\alpha(\underset{\sim}{\mathbf{\Sigma}}^0_\beta(X))$,
- For a limit ordinal λ, $A \in \underset{\sim}{\mathbf{D}}_\lambda(\underset{\sim}{\mathbf{\Sigma}}^0_\beta(X))$ if $A = \bigcup_{\alpha<\lambda,\alpha \text{ even}} B_{\alpha+1} \setminus B_\alpha$ where $(B_\alpha)_{\alpha<\lambda}$ is a growing sequence of sets in $\underset{\sim}{\mathbf{\Sigma}}^0_\beta(X)$.

We define $\check{\underset{\sim}{\mathbf{D}}}_\alpha(\underset{\sim}{\mathbf{\Sigma}}^0_\beta(X))$ as the class of complements of sets in the class $\underset{\sim}{\mathbf{D}}_\alpha(\underset{\sim}{\mathbf{\Sigma}}^0_\beta(X))$. We will mainly focus on the hierarchy based on $\underset{\sim}{\mathbf{\Sigma}}^0_1(X)$, and we denote $\underset{\sim}{\mathbf{D}}_\alpha(\underset{\sim}{\mathbf{\Sigma}}^0_1(X))$ by $\underset{\sim}{\mathbf{D}}_\alpha(X)$.

In any topological space X, the difference hierarchy based on $\underset{\sim}{\mathbf{\Sigma}}^0_\beta(X)$ is contained in $\underset{\sim}{\mathbf{\Delta}}^0_{\beta+1}(X)$. On Polish spaces and even quasi-Polish spaces, the Hausdorff-Kuratowski Theorem states that the hierarchy entirely exhausts $\underset{\sim}{\mathbf{\Delta}}^0_{\beta+1}(X)$ (Theorem 70 in [2]).

2.1.3 Representations of sets

Representations of descriptive complexity classes have been investigated in [1, 14].

As soon as one has chosen a representation of open sets, at least the finite levels of the hierarchies have an obvious representation. For instance, a set in $\underset{\sim}{\mathbf{D}}_2(X)$ is represented by pairing two names of open subsets of X. A set in $\underset{\sim}{\mathbf{\Sigma}}^0_{n+1}(X)$ is inductively represented by two sequences of names of sets in $\underset{\sim}{\mathbf{\Sigma}}^0_n(X)$.

If (X, δ_X) is a represented set, then the canonical topology on X is the final topology of δ_X. An open subset U of X can be represented by a name of any open subset of $\mathcal{N}$ whose intersection with $\mathrm{dom}(\delta_X)$ is $\delta_X^{-1}(U)$.

3 Measuring the topological complexity of a set

When studying the descriptive complexity of sets in topological spaces, an important task is to prove that a set does not belong to a given class. In the traditional theory on Polish spaces, it can be achieved using the notion of hardness. We investigate on which topological spaces this technique is still valid, and develop an alternative technique working on all topological spaces.

Let Γ be a complexity class. We say that $A \subseteq X$ is Γ***-hard*** if for every $C \in \Gamma(\mathcal{N})$, there is a continuous reduction from C to A, i.e. a continuous map $f : \mathcal{N} \to X$ such that $C = f^{-1}(A)$.

On any Polish space X, if a class Γ is not self-dual (i.e., $\Gamma \neq \check{\Gamma}$) and is closed under continuous preimages, then for $A \subseteq X$,

$$A \notin \Gamma(X) \iff A \text{ is } \check{\Gamma}(X)\text{-hard.} \tag{1}$$

This is essentially Wadge's lemma in [7]. We call this equivalence the *hardness criterion*. This result can easily be extended to the countably-based spaces admitting a *total* admissible representation δ (they are called quasi-Polish spaces [2]).

As we will see in this paper, it fails in other spaces such as $\mathbb{R}[X]$, because the hardness of a set does not measure its topological complexity, but the complexity of its preimage under the representation, which may differ. Therefore, we need some techniques to prove that a set does not belong to a given class. We are going to see that for classes below $\underset{\sim}{\mathbf{\Delta}}^0_2$,

- There is a characterization that is valid in any topological space,
- The hardness criterion (1) surprisingly holds for any countably-based space.

3.1 Arbitrary topological spaces

In order to characterize the classes of the difference hierarchy, we adapt the proof of the Hausdorff-Kuratowski Theorem presented in [7], in which the level of a set in the difference hierachy is essentially captured by iterating an operator on the set, and identifying the level at which the set becomes empty. However, the operator used in [7], which takes a set to its boundary, is too coarse to distinguish between the classes $\underset{\sim}{\mathbf{D}}_\eta$ and $\check{\underset{\sim}{\mathbf{D}}}_\eta$. We refine it and show how it captures precisely the complexity of a set.

Intuitively, iterating the operator progressively removes the simple parts of the set, so that the level at which the set is emptied measures the complexity of the set. Let ω_1 be the first uncountable ordinal.

▶ **Definition 3.1.** *Let X be a topological space. For a set $A \subseteq X$, we define the sequence of closed sets $(H_\eta)_{\eta<\omega_1}$ by transfinite induction on η as follows:*

$$\begin{aligned} H_0(A) &= X, \\ H_{\eta+1}(A) &= \overline{A \cap H_\eta(A^c)}, \\ H_\lambda(A) &= \bigcap_{\eta<\lambda} H_\eta(A) \quad \text{for a limit ordinal } \lambda. \end{aligned}$$

Intuitively, $x \in H_2(A)$ if x is arbitrarily close to points of A that are arbitrarily close to points of A^c. At the next level, $x \in H_3(A)$ if x is arbitrarily close to points of A that are arbitrarily close to points of A^c that are themselves arbitrarily close to points of A. More generally, $H_\eta(A)$ contains the points that are sufficiently deep inside the boundary of A.

We now study the basic properties of that sequence (the proofs are given in the appendix). First, it is decreasing,

▶ **Proposition 3.2.** *If $\alpha \leq \beta$ then $H_\alpha(A) \supseteq H_\beta(A)$.*

At the limit levels, A and A^c induce the same set.

▶ **Proposition 3.3.** *For each limit ordinal λ, one has $H_\lambda(A) = H_\lambda(A^c)$.*

We only consider countable ordinals because on a large class of spaces including represented spaces, the sequence reaches a fixed point at some countable ordinal. A topological space is hereditarily Lindelöf if every family of open sets contains a countable subfamily with the same union. The final topology of a representation is always hereditarily Lindelöf.

▶ **Proposition 3.4.** *If X is a hereditarily Lindelöf topological space, then for any $A \subseteq X$ the sequence $(H_\eta)_{\eta<\omega_1}$ is eventually constant.*

Proof. Let $U = X \setminus \bigcap_{\eta<\omega_1} H_\eta(A)$. The growing family of open sets $(X \setminus H_\eta(A))_{\eta<\omega_1}$ covers U, so contains a countable subfamily covering U. As a result, $U = X \setminus H_\eta(A)$ for some $\eta < \omega_1$, and $H_\alpha(A) = H_\eta(A)$ for all $\alpha \geq \eta$. ◀

Whether a point belongs to $H_\eta(A)$ only depends on the local behavior of A around that point, which can be formulated as follows.

▶ **Lemma 3.5.** *Let U be open. If $A \cap U = B \cap U$, then $H_\eta(A) \cap U = H_\eta(B) \cap U$.*

In particular, $H_\eta(A) \cap U = H_\eta(A \cap U) \cap U = H_\eta(A \cup U^c) \cap U$.

The main result of this section is that the topological complexity of a set is captured by this sequence, which gives an operative way of identifying the complexity of a set, and will be used in the sequel.

▶ **Theorem 3.6.** *Let X be any topological space. For a set $A \subseteq X$, one has*

$$A \in \underset{\sim}{\mathbf{D}}_\eta(X) \iff H_{\eta+1}(A) = \emptyset.$$

The proofs are rather technical and given in the appendix. We simply mention that one direction has the following more general formulation.

▶ **Lemma 3.7.** *For any set $A \subseteq X$, one has $A \setminus H_{\eta+1}(A) \in \underset{\sim}{\mathbf{D}}_\eta(X)$.*

3.2 Countably-based spaces

The preceding result enables us to go further by extending the hardness criterion (1) to any countably-based space, at least for the classes of the difference hierarchy.

▶ **Theorem 3.8** (Hardness criterion)**.** *Let X be countably-based. Let $\eta < \omega_1$ be a countable ordinal. For every set $A \subseteq X$, one has*

$$A \notin \underset{\sim}{\mathbf{D}}_\eta(X) \iff A \text{ is } \check{\underset{\sim}{\mathbf{D}}}_\eta\text{-hard.}$$

Proof. The implication $\Leftarrow$ holds on any space, we prove the other implication.

We prove by induction on η that for any countably-based space X, if $C \in \underset{\sim}{\mathbf{D}}_\eta(\mathcal{N})$ and $H_{\eta+1}(A) \neq \emptyset$ then there exists a continuous reduction ϕ from C to A^c (or equivalently, from C^c to A).

This induction hypothesis implies in particular that for any countably-based space X and any open set $B \subseteq X$, if $B \cap H_{\eta+1}(A) \neq \emptyset$ then there exists a continuous reduction from C to A^c on any cylinder $[\sigma]$, with values in B. Indeed, in the subspace B, the set $H^B_{\eta+1}(A \cap B)$ (which is $H_{\eta+1}(A \cap B)$) defined in the space B) is the intersection of $H^X_{\eta+1}(A)$ with B.

The case $\eta = 0$ is straightforward: $C = \emptyset$ and $A \neq \emptyset$ so one can take a constant function with value in A.

Now assume the induction hypothesis for some η. Let $C \in \underset{\sim}{\mathbf{D}}_{\eta+1}(\mathcal{N})$ and $H_{\eta+2}(A) \neq \emptyset$. We build a continuous reduction ϕ from C^c to A. We adopt the language of computability, by explaining how to compute $\phi(x)$ from $x \in \mathcal{N}$. Computing $\phi(x) \in X$ means enumerating the basic neighborhoods of $\phi(x)$. The computation is made relative to some suitable oracle encoding whatever is needed.

One has $C = U \setminus C'$ with U open and $C' \in \underset{\sim}{\mathbf{D}}_\eta(\mathcal{N})$. First define $\phi : U^c \to X$ with some constant value in $A \cap H_{\eta+1}(A^c)$ which is non-empty. Given $x \in \mathcal{N}$, start computing $\phi(x)$ as if $x \in U^c$. If eventually one discovers that $x \in U$, then we obtain some $[\sigma] \subseteq U$ with $x \in [\sigma]$. So far, an open neighborhood B of $\phi(x)$ has been enumerated. One has $B \cap H_{\eta+1}(A^c) \neq \emptyset$. By induction hypothesis applied to η, $[\sigma] \cap C'$ and A^c, one can define ϕ on $[\sigma]$ with values in B, reducing $[\sigma] \cap C'$ to $A \cap B$.

We now prove the case of limit ordinals. Let λ be a limit ordinal, and assume the induction hypothesis for all $\eta < \lambda$.

Let $C \in \underset{\sim}{\mathbf{D}}_\lambda(\mathcal{N})$ and $H_{\lambda+1}(A) \neq \emptyset$. Let $(U_\alpha)_{\alpha<\lambda}$ be open sets defining C, and U their union. We first define ϕ on U^c with some constant value in $A \cap H_\lambda(A^c)$, which is non-empty. Given $x \in \mathcal{N}$, start computing $\phi(x)$ as if $x \in U^c$. If eventually one discovers that $x \in U$, then we obtain some $[\sigma] \subseteq U_\eta$ with $x \in [\sigma]$ and $\eta < \lambda$. So far, an open neighborhood B of $\phi(x)$ has been enumerated. By Proposition 3.3, one has $B \cap H_\lambda(A) = B \cap H_\lambda(A^c) \neq \emptyset$ hence $B \cap H_{\eta+2}(A) \neq \emptyset$. As $[\sigma] \subseteq U_\eta$, one has $[\sigma] \cap C \in \underset{\sim}{\mathbf{D}}_{\eta+1}(\mathcal{N})$. By induction, one can define ϕ on $[\sigma]$ with values in B, reducing $[\sigma] \cap C$ to A^c. ◀

We leave open the question whether such the hardness criterion holds for classes above $\underset{\sim}{\mathbf{\Delta}}^0_2$ in countably-based spaces.

We will see later that in the space $\mathbb{R}[X]$ of polynomials, it already fails at the level $\underset{\sim}{\mathbf{D}}_2$.

Theorem 3.8 gives for free a characterization of the countably-based spaces on which the Hausdorff-Kuratowski Theorem holds. A set is said to be $\underset{\sim}{\mathbf{\Delta}}^0_2$-complete if it belongs to $\underset{\sim}{\mathbf{\Delta}}^0_2(X)$ and is $\underset{\sim}{\mathbf{\Delta}}^0_2$-hard.

▶ **Corollary 3.9.** *In a countably-based space, the Hausdorff-Kuratowski Theorem holds iff there is no $\underset{\sim}{\mathbf{\Delta}}^0_2$-complete set.*

Proof. If the HK theorem fails then there is $A \in \underset{\sim}{\mathbf{\Delta}}^0_2(X)$ such that $A \notin \underset{\sim}{\mathbf{D}}_\eta(X)$ for any $\eta < \omega_1$. By Theorem 3.8, A is $\check{\underset{\sim}{\mathbf{D}}}_\eta$-hard for each η. As a result, A is $\underset{\sim}{\mathbf{\Delta}}^0_2$-hard hence $\underset{\sim}{\mathbf{\Delta}}^0_2$-complete.

If the HK theorem holds then no $\underset{\sim}{\mathbf{\Delta}}^0_2$-set can be complete. Indeed, if $A \in \underset{\sim}{\mathbf{\Delta}}^0_2(X)$ then $A \in \underset{\sim}{\mathbf{D}}_\eta(X)$ for some $\eta < \omega_1$ by the HK theorem. If A is $\underset{\sim}{\mathbf{\Delta}}^0_2$-hard then every $C \in \underset{\sim}{\mathbf{\Delta}}^0_2(\mathcal{N})$ is continuously reducible to A so $C \in \underset{\sim}{\mathbf{D}}_\eta(\mathcal{N})$, contradicting the fact that the difference hierarchy does not collapse on $\mathcal{N}$. ◀

For instance, the space $\mathcal{C} = \{f : \mathbb{N} \to \mathbb{N} : f \text{ is eventually constant}\} \subseteq \mathcal{N}$ is countably-based and contains a $\underset{\sim}{\mathbf{\Delta}}^0_2$-complete set $\mathcal{C}_0 = \{f : \mathbb{N} \to \mathbb{N} : f \text{ is eventually null}\}$. Therefore, to show that a set is $\underset{\sim}{\mathbf{\Delta}}^0_2$-hard, it is sufficient to reduce $\mathcal{C}_0$ to that set. One may ask whether it is always possible. We answer positively in the case of countably-based spaces.

▶ **Proposition 3.10.** *In a countably-based space X, a set $A \subseteq X$ is $\underset{\sim}{\mathbf{\Delta}}^0_2$-hard iff there exists a continuous function $\phi : \mathcal{C} \to X$ such that $\mathcal{C}_0 = \phi^{-1}(A)$.*

Proof. If A is $\underset{\sim}{\mathbf{\Delta}}^0_2$-hard, then we show that there is a non-empty closed set F such that both $F \cap A$ and $F \setminus A$ are dense in F. From this result, we can build a reduction as follows. Given $f \in \mathcal{C}$, one can decide with finitely many mind-changes whether $f \in \mathcal{C}_0$. We can assume that the first guess is that $f \in \mathcal{C}_0$. We start outputting a point in $F \cap A$; each time we change our mind, if our new guess is that $f \notin \mathcal{C}_0$ then we move to a point in $F \setminus A$, and if our new guess is that $f \in \mathcal{C}_0$, then we move to a point in $F \cap A$. We can do that because both sets are dense in F, so whatever the current neighborhood of the point we have already output, we can move. After some finite time, there is no more mind-change, so we indeed output a point, which belongs to A iff $f \in \mathcal{C}_0$.

Let us now prove the existence of such a F. For any $\eta < \omega_1$, one has $A \notin \underset{\sim}{\mathbf{D}}_\eta(X)$ (otherwise every set in $\underset{\sim}{\mathbf{\Delta}}^0_2(\mathcal{N})$ would belong to $\underset{\sim}{\mathbf{D}}_\eta(\mathcal{N})$, so the difference hierarchy would collapse on $\mathcal{N}$). By Theorem 3.6, one has $H_{\eta+1}(A) \neq \emptyset$ for all $\eta < \omega_1$. Let η be such that $H_\alpha(A) = H_\eta(A)$ for all $\alpha \geq \eta$. Let $F = H_\eta(A)$. It is a closed set, and both A and A^c are dense in it. Indeed, $H_\eta(A) = H_{\eta+2}(A) \subseteq H_{\eta+1}(A^c) \subseteq H_\eta(A)$ so they are all equal. Therefore $H_\eta(A) = H_{\eta+1}(A^c) = \overline{A^c \cap H_\eta(A)}$, so A^c is dense in $H_\eta(A)$, and $H_\eta(A) = H_{\eta+2}(A) = \overline{A \cap H_\eta(A)}$ so A is dense in $H_\eta(A)$. ◀

4 Preservation of descriptive complexity by representations

4.1 Countably-based spaces

It is proved in [2] that in a countably-based space X with a continuous open or admissible representation $\delta :\subseteq \mathcal{N} \to X$, the descriptive complexity of $\delta^{-1}(A)$ in $\mathrm{dom}(\delta)$ coincides with the descriptive complexity of A in X. The proof is based on [11, 16] and we point out that it is actually effective. The effective version of the $\mathbf{\Sigma}^0_2$ case was proved in [6]. The classes D_m and $\Sigma^0_n(X)$ is the effective version of $\underset{\sim}{\mathbf{D}}_m$ and $\underset{\sim}{\mathbf{\Sigma}}^0_n(X)$.

▶ **Theorem 4.1** (Preservation). *Let X be an effective countably-based space. The inverse of the function $\delta^{-1} : \underset{\sim}{\mathbf{D}}_m(\underset{\sim}{\mathbf{\Sigma}}^0_n(X)) \to \underset{\sim}{\mathbf{D}}_m(\underset{\sim}{\mathbf{\Sigma}}^0_n(\mathrm{dom}(\delta)))$ is computable. In particular, for every $A \subseteq X$,*

$$\delta^{-1}(A) \in \mathrm{D}_m(\Sigma^0_n(\mathrm{dom}(\delta))) \iff A \in \mathrm{D}_m(\Sigma^0_n(X)).$$

This result is used and generalized in [15]. We follow the lines of the proofs given in [11, 2]. For $A \subseteq \mathcal{N}$, let $B(A) = \{x \in X : A \cap \delta^{-1}(x) \text{ is not meager in } \delta^{-1}(x)\}$.

▶ **Lemma 4.2.** $B(\bigcup_{i\in\mathbb{N}} S_i) = \bigcup_{i\in\mathbb{N}} B(S_i)$.

Proof. $\delta^{-1}(x)$ is Polish so in that space, $\bigcup_i S_i$ is not meager iff some S_i is not meager. ◀

▶ **Lemma 4.3.** *For any Borel set $S \subseteq \mathcal{N}$, one has $B(S) = \bigcup_\sigma \delta([\sigma]) \setminus B([\sigma] \setminus S)$.*

Proof. One has $x \in B(S)$ iff $S \cap \delta^{-1}(x)$ is not meager in $\delta^{-1}(x)$, iff there exists σ such that $[\sigma]$ intersects $\delta^{-1}(x)$ (which means that $x \in \delta([\sigma])$) and $[\sigma] \cap S \cap \delta^{-1}(x)$ is comeager in $[\sigma] \cap \delta^{-1}(x)$. The latter property can be reformulated as follows:

$$\begin{aligned}
&[\sigma] \cap S \cap \delta^{-1}(x) \text{ is comeager in } [\sigma] \cap \delta^{-1}(x)\\
&\iff [\sigma] \setminus S \cap \delta^{-1}(x) \text{ is meager in } [\sigma] \cap \delta^{-1}(x)\\
&\iff [\sigma] \setminus S \cap \delta^{-1}(x) \text{ is meager in } \delta^{-1}(x)\\
&\iff x \notin B([\sigma] \setminus S).
\end{aligned}$$

As a result, $x \in B(S)$ iff there exists σ such that $x \in \delta([\sigma]) \setminus B([\sigma] \setminus S)$. ◀

▶ **Lemma 4.4.** *Let X be an effective countably-based space. The function $B : \underset{\sim}{\mathbf{\Sigma}}^0_n(\mathcal{N}) \to \underset{\sim}{\mathbf{\Sigma}}^0_n(X)$ is computable.*

Proof. We prove it by induction on n. For $n = 1$, if $A \in \underset{\sim}{\mathbf{\Sigma}}^0_1(\mathcal{N})$ then $B(A) = \delta(A)$ so the result follows because δ is effectively open.

Let $A \in \underset{\sim}{\mathbf{\Sigma}}^0_{n+1}(\mathcal{N})$. A is given as $\bigcup_i A_i$ with $A_i \in \underset{\sim}{\mathbf{\Pi}}^0_n(\mathcal{N})$. By Lemmas 4.2 and 4.3,

$$B(A) = \bigcup_i B(A_i) = \bigcup_{i,\sigma} \delta([\sigma]) \setminus B([\sigma] \setminus A_i).$$

We can conclude by applying the induction hypothesis to $[\sigma] \setminus A_i \in \underset{\sim}{\mathbf{\Sigma}}^0_n(\mathcal{N})$. ◀

Proof of Theorem 4.1. One has $B(\delta^{-1}(A)) = A$ so the inverse of δ^{-1} is exactly B, which is computable by Lemma 4.4. It shows the case $m = 1$. For $m > 1$, one can show as in [2] that if $\delta^{-1}(A) = S \setminus T$ then $A = B(S) \setminus B(T)$, so the result follows by induction on m. ◀

The possibility of converting the description of the preimage of a set into a description of the set in a uniform continuous way is actually a characterization of countably-based spaces, as shown by the next result.

▶ **Theorem 4.5.** *Let X be an admissibly represented space. The function $\delta^{-1} : \underset{\sim}{\mathbf{D}}_2(X) \to \underset{\sim}{\mathbf{D}}_2(\mathrm{dom}(\delta))$ is computably invertible relative to an oracle iff X is countably-based.*

To prove the theorem, we need the next Lemma, whose proof is given in the appendix. In a topological space, the specialization preorder is $x \leq y$ if every neighborhood of x is a neighborhood of y.

▶ **Lemma 4.6.** *For every admissibly represented space, there is an admissible representation δ such that the sets $\delta([\sigma])$ are upward closed for the specialization preorder.*

Proof. We use the following characterization of admissibly represented spaces [9]: the canonical injection $X \to \mathcal{O}(\mathcal{O}(X))$ sending x to the set of its open neighborhoods has a continuously realizable inverse.

For any represented set Y, the canonical representation of $\mathcal{O}(Y)$ has the upward closedness property: the specialization preorder on $\mathcal{O}(Y)$ is inclusion and every prefix of a name of an open set U can be extended to the name of any open set $V \supseteq U$: the prefix encodes a finite list of cylinders whose intersection with $\mathrm{dom}(\delta_Y)$ is contained in $\delta_Y^{-1}(U)$, hence in $\delta_Y^{-1}(V)$.

Now by taking $Y = \mathcal{O}(X)$ we get that the representation of $\mathcal{O}(\mathcal{O}(X))$ has this property. Therefore any subspace also has this property. In particular, as X is admissibly represented, X is a represented subspace of $\mathcal{O}(\mathcal{O}(X))$. ◀

Proof of Theorem 4.5. We can assume w.l.o.g. that δ has the upward closedness property as in Lemma 4.6. Indeed, δ is equivalent to such a representation δ_*, and if δ^{-1} is continuously invertible as in the statement then so is δ_*^{-1}. Indeed, one has $\delta = \delta_* \circ F$ for some continuous $F : \mathrm{dom}(\delta) \to \mathrm{dom}(\delta^*)$, the function $F^{-1} : \underset{\sim}{\mathbf{D}}_2(\mathrm{dom}(\delta_*)) \to \underset{\sim}{\mathbf{D}}_2(\mathrm{dom}(\delta))$ is continuous, so given $\delta_*^{-1}(A) \in \underset{\sim}{\mathbf{D}}_2(\mathrm{dom}(\delta_*))$ one can continuously obtain $\delta^{-1}(A) = F^{-1}(\delta_*^{-1}(A)) \in \underset{\sim}{\mathbf{D}}_2(\mathrm{dom}(\delta))$, from which one can continuously obtain $A \in \underset{\sim}{\mathbf{D}}_2(X)$.

Assume that X is not countably-based. For each finite union of cylinders $C_i \subseteq \mathcal{N}$, the interior B_i of $\delta(C_i)$ is an open subset of X. As X is not countably-based, the sequence $(B_i)_{i\in\mathbb{N}}$ is not a basis, therefore there exists an open set $U \subseteq X$ which is not a union of B_i's. It means that there exists $x \in U$ such that $x \notin B_i$ for each $B_i \subseteq U$.

Assume that the inverse ϕ of $\delta^{-1} : \underset{\sim}{\mathbf{D}}_2(X) \to \underset{\sim}{\mathbf{D}}_2(\mathrm{dom}(\delta))$ is continuously realizable, let $\Phi :\subseteq \mathcal{N} \to \mathcal{N}$ be a continuous realizer of ϕ.

We build a set $A \in \underset{\sim}{\mathbf{D}}_2(X)$ by producing a name of $\delta^{-1}(A) \in \underset{\sim}{\mathbf{D}}_2(\mathrm{dom}(\delta))$, feeding it to Φ and observing its output, which must be a name of $A \in \underset{\sim}{\mathbf{D}}_2(X)$. In other words, we feed Φ with a pair of names of open sets $E_0, E_1 \subseteq \mathcal{N}$ such that $\delta^{-1}(A) = E_1 \setminus E_0 \cap \mathrm{dom}(\delta)$ and we observe names of open sets $F_0, F_1 \subseteq \mathcal{N}$ such that $A = A_1 \setminus A_0$ with $\delta^{-1}(A_i) = F_i \cap \mathrm{dom}(\delta)$. Our goal is to make Φ fail.

We pick a particular name p of x. We start with $E_0 = \emptyset$ and $E_1 = \delta^{-1}(U)$, and start feeding names of them to Φ. We wait for p to appear in F_1 (which must happen, otherwise $A = U$ but $x \notin A_1 \setminus A_0$). When p appears in F_1, we stop our enumeration of E_1, which is currently some C_i, and let $E_0 = C_i$. We start extending the names of E_0 and E_1 so that $E_0 = E_1 = C_i$, and wait for p to appear in F_0 (it must happen, otherwise $A = \emptyset$ but $x \in A_1 \setminus A_0$). When p appears in F_0, we do the following.

As $\delta(C_i) \subseteq U$, x does not belong to the interior of $\delta(C_i)$. As x belongs to the open set A_0, this open set cannot be contained in $\delta(C_i)$. As a result, there exists $y \in A_0 \setminus \delta(C_i)$, and we wait until we find a name of such a y in F_0. The representation δ has the property that the image of any cylinder is upward closed, in particular $\delta(C_i)$ is upward closed. As a result, the closure of $\{y\}$, denoted by $\overline{\{y\}}$, is disjoint from $\delta(C_i)$ (indeed, $z \in \overline{\{y\}}$ iff $z \leq y$, so $z \notin \delta(C_i)$).

Now, we switch to $A = \overline{\{y\}}$, with $E_1 = \mathcal{N}$ and E_0 an open set such that $\mathrm{dom}(\delta) \cap E_0 = \mathrm{dom}(\delta) \setminus \delta^{-1}(\overline{\{y\}})$. It is indeed possible to find such an open set containing C_i, because $\delta(C_i)$ is disjoint from $\overline{\{y\}}$, and C_i is the part of E_0 that has already been enumerated. The output of Φ is mistaken, because $y \in A$ but $y \notin A_1 \setminus A_0$ (we chose $y \in A_0$).

Therefore we get a contradiction, which implies that Φ cannot exist. ◀

5 CoPolish spaces

So far, we have essentially obtained positive results, in particular that the standard representation of countably-based spaces preserve descriptive complexity of sets.

We now investigate what happens on non-countably based spaces. It is however a very vast class, so we focus on one of the simplest classes of spaces, the coPolish spaces.

CoPolish spaces are a nice class of spaces going beyond countably-based spaces. They were studied in [12] where they arise as a natural class of spaces on which complexity theory can be developed. We start by showing that in coPolish spaces, the representation does not always preserve descriptive complexity. Moreover, we give a simple characterization of the coPolish spaces whose representation preserves the descriptive complexity of sets (at least for low level complexity classes). We take the next definition from [3].

▶ **Definition 5.1.** *A* **coPolish space** *is a direct limit of an increasing sequence of compact metrizable subspaces X_n.*

In other words $X = \bigcup_n X_n$ and a set $U \subseteq X$ is open if for each n, $U \cap X_n$ is open on X_n. In this topology, a converging sequence is entirely contained in some X_n [12]. An admissible representation δ_X is obtained as follows: a point $x \in X$ is represented by a pair $(n, p) \in \mathbb{N} \times \mathcal{N} \cong \mathcal{N}$ where $n \in \mathbb{N}$ is such that $x \in X_n$, and $p \in \mathcal{N}$ is a name of x in X_n.

▶ Remark 5.2. For a descriptive complexity class Γ in the Borel and difference hierarchies, one has

$$\delta_X^{-1}(A) \in \Gamma(\mathrm{dom}(\delta_X)) \iff \forall n, A \cap X_n \in \Gamma(X_n).$$

Indeed, $\delta_X^{-1}(A)$ is the disjoint union over $n \in \mathbb{N}$ of $[n] \cap \delta_X^{-1}(A)$, so $\delta_X^{-1}(A)$ belongs to a class iff each member of the disjoint union belongs to that class. Now observe that on $[n]$, δ_X is simply δ_{X_n}. We conclude by observing that as X_n is countably-based, the complexity of $A \cap X_n$ is the same as the complexity of its preimage under δ_{X_n}.

We recall that a topological space is Fréchet-Urysohn if the closure of each set is the set of limits of sequences of points in the set.

▶ **Theorem 5.3.** *For a coPolish space X, the following statements are equivalent:*

- *X is Fréchet-Urysohn,*
- *For every $A \subseteq X$, $\delta_X^{-1}(A) \in \underset{\sim}{\mathbf{D}}_2(\mathrm{dom}(\delta_X))$ implies $A \in \underset{\sim}{\mathbf{D}}_2(X)$,*
- *For every $A \subseteq X$ and every $n < \omega$, $\delta_X^{-1}(A) \in \underset{\sim}{\mathbf{D}}_n(\mathrm{dom}(\delta_X))$ implies $A \in \underset{\sim}{\mathbf{D}}_n(X)$.*

We need the following results.

▶ **Lemma 5.4.** *Let X be a Fréchet-Urysohn coPolish space. If a sequence x_i converges to some x, with $x_i \neq x$ for all i, then there exists p such that $x_i \in \mathrm{int}(X_p)$ for almost all i.*

Proof. Assume that for each p, there exist infinitely many i such that $x_i \notin \mathrm{int}(X_p)$. We can extract a subsequence $x_{i_p} \notin \mathrm{int}(X_p)$ with $i_p < i_{p+1}$. Let U_p be a neighborhood of x_{i_p} such that $x \notin \overline{U_p}$ (it exists as $x_{i_p} \neq x$ and X is Hausdorff). Let $U = \bigcup_p U_p \setminus X_p$. One has $x_{i_p} \in \overline{U}$

for all p, so $x \in \overline{U}$. As the space if Fréchet-Urysohn, there exists p_0 such that $x \in \overline{U \cap X_{p_0}}$ (indeed, there exists a sequence in U converging to x, and that sequence must belong to some X_{p_0}). However, $U \cap X_{p_0} \subseteq \bigcup_{p<p_0} U_p \setminus X_p$ so its closure does not contain x, giving a contradiction. ◀

Let $H_n^{X_p}(A \cap X_p)$ be the set $H_n(A \cap X_p)$, but defined in the space X_p.

▶ **Lemma 5.5.** *If X is a Fréchet-Urysohn coPolish space, then $H_n(A) = \bigcup_p H_n^{X_p}(A \cap X_p)$.*

Proof. We prove it by induction on n. For $n = 0$ the result is clear. Assume the result for $n \in \mathbb{N}$.

Let $x \in H_{n+1}(A) = \overline{A \cap H_n(A^c)}$. There exists a sequence $x_i \in A \cap H_n(A^c)$ converging to x. If $x = x_i$ for some i, then $x \in A \cap H_n(A^c)$ so $x \in A \cap H_n^{X_p}(A^c \cap X_p)$ for some p by induction hypothesis, hence $x \in H_{n+1}^{X_p}(A \cap X_p)$. If $x \neq x_i$ for all i, then by Lemma 5.4 there exists p such that $x_i \in \mathrm{int}(X_p)$ for almost all i. As a result, using Lemma 3.5 we see that $x_i \in H_n^{X_p}(A^c \cap X_p)$ for almost all i. We can take p large enough so that $x \in X_p$, and obtain that $x \in H_{n+1}^{X_p}(A \cap X_p)$. ◀

Proof of Theorem 5.3. Let X be Fréchet-Urysohn. If $\delta^{-1}(A) \in \underset{\sim}{\mathbf{D}}_n(\mathrm{dom}(\delta_X))$ then for each $p \in \mathbb{N}$, $A \cap X_p \in \underset{\sim}{\mathbf{D}}_n(X_p)$ by Remark 5.2 so $H_{n+1}^{X_p}(A \cap X_p) = \emptyset$. As a result, Lemma 5.5 implies that $H_{n+1}(A) = \emptyset$ which implies that $A \in \underset{\sim}{\mathbf{D}}_n(X)$.

If X is not Fréchet-Urysohn, then there exists a double-sequence $x_{n,p} \in X$ such that for each n, $x_{n,p}$ converges to some x_n as $p \to \infty$, which in turn converges to some x as $n \to \infty$, with no sequence in $\{x_{n,p} : n, p \in \mathbb{N}\}$ converging to x, and such that $C = \{x\} \cup \{x_n : n \in \mathbb{N}\} \cup \{x_{n,p} : n, p \in \mathbb{N}\}$ is closed (it was proved in [3], Proposition 66). Let $A := \{x\} \cup \{x_{n,p} : n, p \in \mathbb{N}\}$. One easily checks that $H_3(A) = \{x\} \neq \emptyset$ so $A \notin \underset{\sim}{\mathbf{D}}_2(X)$. However, $\delta_X^{-1}(A) \in \underset{\sim}{\mathbf{D}}_2(\mathrm{dom}(\delta_X))$. For each i, the set $\{n : \exists p, x_{n,p} \in A \cap X_i\}$ is finite. Therefore, one can easily see that $A \cap X_i \in \underset{\sim}{\mathbf{D}}_2(X_i)$. It implies that $\delta_X^{-1}(A) \in \underset{\sim}{\mathbf{D}}_2(\mathrm{dom}(\delta_X))$. ◀

The result does not extend to higher complexity levels. The space $\mathbb{R}/\mathbb{Z}$ is coPolish and Fréchet-Urysohn, but the representation does not preserve the level ω of the difference hierarchy.

▶ **Proposition 5.6.** *There exists $A \subseteq \mathbb{R}/\mathbb{Z}$ such that $\delta^{-1}(A) \in \underset{\sim}{\mathbf{D}}_\omega(\mathrm{dom}(\delta))$ but $A \notin \underset{\sim}{\mathbf{D}}_\omega(\mathbb{R}/\mathbb{Z})$.*

Proof. The space $X = \mathbb{R}/\mathbb{Z}$ is the direct limit of $X_n = [-n, n]/\mathbb{Z}$. For each n, let $A_n \subseteq [n, n + 1/2]$ be such that $A_n \in \underset{\sim}{\mathbf{D}}_{n+1} \setminus \underset{\sim}{\mathbf{D}}_n$, with $n \in H_{n+1}(A_n)$. Let A be the quotient of $\bigcup_n A_n$. For each n, one has $A \cap X_n \in \underset{\sim}{\mathbf{D}}_{n+1}(X_n) \subseteq \underset{\sim}{\mathbf{D}}_\omega(X_n)$, so $\delta^{-1}(A) \in \underset{\sim}{\mathbf{D}}_\omega(\mathrm{dom}(\delta))$.

However, $A \notin \underset{\sim}{\mathbf{D}}_\omega(X)$ because $0 \in H_{\omega+1}(A) \neq \emptyset$. Indeed, $0 \in H_{n+1}(A) \subseteq H_n(A^c)$ for all n, so $0 \in A \cap H_\omega(A^c) \subseteq H_{\omega+1}(A)$. ◀

On coPolish spaces, the representation may not preserve low complexity classes. However, it always preserves classes $\underset{\sim}{\mathbf{\Delta}}_2^0$ and above.

▶ **Proposition 5.7.** *Let X be coPolish. For each $\alpha \geq 1$ and $A \subseteq X$, one has*

$$\delta_X^{-1}(A) \in \underset{\sim}{\mathbf{\Sigma}}_\alpha^0(\mathrm{dom}(\delta_X)) \iff A \in \underset{\sim}{\mathbf{\Sigma}}_\alpha^0(X).$$

Proof. For $\alpha = 1$, it follows from the admissibility of δ_X. Let $\alpha \geq 2$. As observed earlier, one has $\delta_X^{-1}(A) \in \underset{\sim}{\mathbf{\Sigma}}_\alpha^0(\mathrm{dom}(\delta_X)) \iff \forall n, A \cap X_n \in \underset{\sim}{\mathbf{\Sigma}}_\alpha^0(X_n)$. As $X_n \in \underset{\sim}{\mathbf{\Pi}}_1^0(X) \subseteq \underset{\sim}{\mathbf{\Sigma}}_\alpha^0(X)$, it implies that $A \cap X_n \in \underset{\sim}{\mathbf{\Sigma}}_\alpha^0(X)$. Therefore, $A = \bigcup_n A \cup X_n \in \underset{\sim}{\mathbf{\Sigma}}_\alpha^0(X)$. ◀

In particular, if $\delta_X^{-1}(A) \in \underset{\sim}{\mathbf{D}}_2(\mathrm{dom}(\delta_X))$ then $A \in \underset{\sim}{\mathbf{\Delta}}_2^0(X)$. We show that this gap cannot be reduced in the space of polynomials $\mathbb{R}[X]$. This space is obtained as the direct limit of the spaces $\mathbb{R}^n$, consisting of the polynomials of degree $\leq n$, identified with their coefficients. A polynomial is then represented by giving any upper bound on its degree, and the finite list of its coefficients.

▶ **Theorem 5.8.** *There exists a set $A \subseteq \mathbb{R}[X]$ such that $\delta^{-1}(A) \in \mathrm{D}_2(\mathrm{dom}(\delta))$ but $A \notin \underset{\sim}{\mathbf{D}}_\alpha(\mathbb{R}[X])$ for any $\alpha < \omega_1$.*

Proof. Let

$$A = \left\{ \frac{1}{k_1} + \frac{1}{k_2} X^{k_1} + \ldots + \frac{1}{k_{n+1}} X^{k_n} : k_1 < k_2 < \ldots < k_{n+1}, k_n \text{ even} \right\}.$$

First, the closure of A can be easily obtained by taking the definition of A without the evenness condition on n. Inside $\overline{A}$, the complement of A is dense. No set of the difference hierarchy can be dense and with a dense complement.

However, we now show that $A \cap X_d \in \mathrm{D}_2(\mathbb{R}^d)$. Let $P = \sum_{i \leq d} p_i X^i \in \overline{A}$ be given by a name (in particular, we know d). One can compute the degree of P with one mind-change. Indeed, all the non-null coefficients except the dominant one must be at least $\frac{1}{d}$. So for each $i \leq d$, we test in parallel whether $p_i < \frac{1}{d}$ and whether $p_i > 0$, and wait that one of them stops (which must happen). Let $i \leq d$ be maximal such that the test $p_i > 0$ stops first. That i is our current guess for the degree of P. We then start testing $p_j > 0$ for all $i < j \leq d$. If such a j is eventually found, then it is the degree of P.

We now show how to decide whether $P \in A$ with at most two mind changes, starting with rejection. We start rejecting P. If i is even then we change our mind and accept P. If we eventually realize that the degree of P is some $j > i$, then if j is even, we accept P, otherwise we reject P.

Now, given $P \in \mathbb{R}^d$, we run the previous algorithm and in parallel test whether $P \notin \overline{A}$. If the latter condition is eventually found true, then we stop the algorithm and definitely reject P. ◀

This example has several consequences, showing that many results working on countably-based spaces fail on $\mathbb{R}[X]$. First, the hardness criterion (1), which can be extended to countably-based spaces (Theorem 3.8) does not hold on the space of polynomials.

▶ **Corollary 5.9.** *In $\mathbb{R}[X]$, there exists a set $A \notin \underset{\sim}{\mathbf{D}}_2(X)$ that is not $\underset{\sim}{\check{\mathbf{D}}}_2$-hard.*

Proof. We take the set $A \notin \underset{\sim}{\mathbf{D}}_2(\mathbb{R}[X])$ with $\delta^{-1}(A) \in \underset{\sim}{\mathbf{D}}_2(\mathrm{dom}(\delta))$. Take some $C \in \underset{\sim}{\check{\mathbf{D}}}_2(\mathcal{N}) \setminus \underset{\sim}{\mathbf{D}}_2(\mathcal{N})$. A continuous reduction $\phi : \mathcal{N} \to X$ from C to A is continuously realizable because the representation is admissible. Any continuous realizer is a continuous reduction from C to $\delta^{-1}(A)$, which implies that $C \in \underset{\sim}{\mathbf{D}}_2(\mathcal{N})$, contradicting the choice of C. Hence C is not continuously reducible to A. ◀

According to Corollary 3.9, the Hausdorff-Kuratowski Theorem holds on a countably-based space iff it contains no $\underset{\sim}{\mathbf{\Delta}}_2^0$-complete set. We show that this characterization fails on $\mathbb{R}[X]$.

▶ **Proposition 5.10.** *The Hausdorff-Kuratowski Theorem fails in the topological space $\mathbb{R}[X]$.*

Proof. Theorem 5.8 provides a set A such that $\delta^{-1}(A) \in \underset{\sim}{\mathbf{D}}_2(\mathrm{dom}(\delta))$, hence $A \in \underset{\sim}{\mathbf{\Delta}}_2^0(\mathbb{R}[X])$, but $A \notin \underset{\sim}{\mathbf{D}}_\eta(\mathbb{R}[X])$ for any $\eta < \omega_1$. ◀

▶ **Proposition 5.11.** *A space with a total admissible representation has no $\underset{\sim}{\mathbf{\Delta}}_2^0$-complete set.*

Proof. If $A \in \underset{\sim}{\mathbf{\Delta}}^0_2(X)$ then $\delta^{-1}(A) \in \underset{\sim}{\mathbf{\Delta}}^0_2(\mathcal{N})$. If A is $\underset{\sim}{\mathbf{\Delta}}^0_2$-hard, then $\delta^{-1}(A)$ is $\underset{\sim}{\mathbf{\Delta}}^0_2$-hard by admissibility. As there is no $\underset{\sim}{\mathbf{\Delta}}^0_2$-complete set in $\mathcal{N}$, there is no $\underset{\sim}{\mathbf{\Delta}}^0_2$-complete set in X. ◀

Note that every coPolish space, in particular $\mathbb{R}[X]$, has a total admissible representation.

5.1 Complexity via representation

As we have just seen, several results from descriptive set theory fail in $\mathbb{R}[X]$. However, when the complexity of a set is measured using the representation, these results can be recovered. The next results apply to any space with a total admissible representation (investigated in [14]), in particular to any coPolish space. First, the hardness criterion (1) can be recovered by measuring the complexity of a set via the representation.

▶ **Proposition 5.12.** *Let (X, δ_X) be an admissibly represented space with δ_X total. Let Γ be a class of the Borel or difference hierarchies that is non-self-dual in $\mathcal{N}$. For every $A \subseteq X$,*

$$\delta_X^{-1}(A) \notin \Gamma(\mathcal{N}) \iff A \text{ is } \check{\Gamma}\text{-hard.}$$

Proof. As δ_X is admissible, any continuous reduction from some $C \in \check{\Gamma}(\mathcal{N})$ to A has a continuous realizer, which is a continuous reduction from C to $\delta_X^{-1}(A)$. As a result, A is $\check{\Gamma}$-hard iff $\delta_X^{-1}(A)$ is $\check{\Gamma}$-hard iff $\delta_X^{-1}(A) \notin \Gamma(\mathcal{N})$. ◀

When the admissible representation is not total, we still have the equivalence for low complexity classes.

▶ **Proposition 5.13.** *Let (X, δ_X) be an admissibly represented space. Let $\Gamma = \underset{\sim}{\mathbf{D}}_\eta(X)$ for some $\eta < \omega_1$. For every $A \subseteq X$, one has*

$$\delta_X^{-1}(A) \notin \Gamma(\mathrm{dom}(\delta_X)) \iff A \text{ is } \check{\Gamma}\text{-hard.}$$

Proof. Again, admissibility of δ_X implies that A is $\check{\Gamma}$-hard iff $\delta_X^{-1}(A)$, as a subset of the space $\mathrm{dom}(\delta_X)$, is $\check{\Gamma}$-hard. As $\mathrm{dom}(\delta_X)$ is a countably-based space (it is a subspace of $\mathcal{N}$), $\delta_X^{-1}(A)$ is $\check{\Gamma}$-hard there iff $\delta_X^{-1}(A) \notin \Gamma(\mathrm{dom}(\delta_X))$ by Theorem 3.8. ◀

References

1. Vasco Brattka. Effective Borel measurability and reducibility of functions. *Mathematical Logic Quarterly*, 51(1):19–44, 2005. doi:10.1002/malq.200310125.
2. Matthew de Brecht. Quasi-Polish spaces. *Ann. Pure Appl. Logic*, 164(3):356–381, 2013. doi:10.1016/j.apal.2012.11.001.
3. Matthew de Brecht, Arno Pauly, and Matthias Schröder. Overt choice. *CoRR*, abs/1902.05926, 2019. arXiv:1902.05926.
4. Matthew de Brecht, Matthias Schröder, and Victor Selivanov. Base-complexity classifications of qcb_0-spaces. *Computability*, 5(1):75–102, 2016. doi:10.3233/COM-150044.
5. Mathieu Hoyrup. Results in descriptive set theory on some represented spaces. Preprint, 2017. URL: http://arxiv.org/abs/1712.03680.
6. Mathieu Hoyrup, Cristobal Rojas, Victor L. Selivanov, and Donald M. Stull. Computability on quasi-Polish spaces. In Michal Hospodár, Galina Jirásková, and Stavros Konstantinidis, editors, *Descriptional Complexity of Formal Systems - 21st IFIP WG 1.02 International Conference, DCFS 2019, Košice, Slovakia, July 17-19, 2019, Proceedings*, volume 11612 of *Lecture Notes in Computer Science*, pages 171–183. Springer, 2019. doi:10.1007/978-3-030-23247-4_13.
7. Alexander S. Kechris. *Classical Descriptive Set Theory*. Springer, January 1995.

8 Dag Normann. Chapter 8 - The continuous functionals. In Edward R. Griffor, editor, *Handbook of Computability Theory*, volume 140 of *Studies in Logic and the Foundations of Mathematics*, pages 251–275. Elsevier, 1999. doi:10.1016/S0049-237X(99)80024-X.

9 Arno Pauly. On the topological aspects of the theory of represented spaces. *Computability*, 5(2):159–180, 2015. doi:10.3233/COM-150049.

10 Arno Pauly and Matthew de Brecht. Descriptive set theory in the category of represented spaces. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 438–449. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.48.

11 Jean Saint Raymond. Preservation of the Borel class under countable-compact-covering mappings. *Topology and its Applications*, 154(8):1714–1725, 2007. doi:10.1016/j.topol.2007.01.006.

12 Matthias Schröder. Spaces allowing type-2 complexity theory revisited. *Math. Log. Q.*, 50(4-5):443–459, 2004. doi:10.1002/malq.200310111.

13 Victor L. Selivanov. Difference hierarchy in φ-spaces. *Algebra and Logic*, 43(4):238–248, July 2004. doi:10.1023/B:ALLO.0000035115.44324.5d.

14 Victor L. Selivanov. Total representations. *Logical Methods in Computer Science*, 9(2), 2013. doi:10.2168/LMCS-9(2:5)2013.

15 Victor L. Selivanov. Effective Wadge hierarchy in computable quasi-Polish spaces. Preprint, 2019. URL: http://arxiv.org/abs/1910.13220.

16 Robert Vaught. Invariant sets in topology and logic. *Fundamenta Mathematicae*, 82(3):269–294, 1974. URL: http://eudml.org/doc/214667.

17 Klaus Weihrauch. *Computable Analysis*. Springer, Berlin, 2000.

A Proofs in Section 3.1

Proof of Proposition 3.2. If β is a limit ordinal then by definition of $H_\beta(A)$, if $\alpha < \beta$ then $H_\beta(A) \subseteq H_\alpha(A)$.

Let λ be a limit ordinal. We prove by induction on $n \in \mathbb{N}$ that

$$\text{for all } A, H_{\lambda+n+1}(A) \subseteq H_{\lambda+n}(A). \tag{2}$$

We first prove it for $n = 0$. One has $H_{\lambda+1}(A) = \overline{A \cap H_\lambda(A^c)} \subseteq H_\lambda(A^c)$. If $\eta < \lambda$ then $H_\lambda(A^c) \subseteq H_{\eta+1}(A^c) \subseteq H_\eta(A)$ by definition, so $H_{\lambda+1}(A) \subseteq \bigcap_{\eta<\lambda} H_\eta(A) = H_\lambda(A)$.

Assuming (2) for $n \in \mathbb{N}$, we apply it to A^c and obtain $H_{\lambda+n+2}(A) = \overline{A \cap H_{\lambda+n+1}(A^c)} \subseteq \overline{A \cap H_{\lambda+n}(A^c)} = H_{\lambda+n+1}(A)$. ◀

Proof of Proposition 3.3. One has

$$\begin{aligned} H_\lambda(A) = \bigcap_{\eta<\lambda} H_\eta(A) &= \bigcap_{\eta<\lambda} H_{\eta+1}(A) \\ &\subseteq \bigcap_{\eta<\lambda} H_\eta(A^c) = H_\lambda(A^c), \end{aligned}$$

and the other inclusion is obtained by exchanging A and A^c. ◀

Proof of Lemma 3.5. First observe that for an open set U and any set S,

$$\overline{S \cap U} \cap U = \overline{S} \cap U. \tag{3}$$

We show by induction on η that

$$\forall A, B, A \cap U = B \cap U \text{ implies } H_\eta(A) \cap U = H_\eta(B) \cap U. \tag{4}$$

For $\eta = 0$, one has $H_\eta(A) = H_\eta(B) = X$.

Assume (4) for η. Let A, B satisfy $A \cap U = B \cap U$. One has $A^c \cap U = B^c \cap U$ so $H_\eta(A^c) \cap U = H_\eta(B^c) \cap U$ by induction hypothesis. Therefore, $A \cap H_\eta(A^c) \cap U = B \cap H_\eta(B^c) \cap U$ so

$$\begin{aligned} H_{\eta+1}(A) \cap U &= \overline{A \cap H_\eta(A^c)} \cap U, \\ &= \overline{A \cap H_\eta(A^c) \cap U} \cap U && \text{by (3)} \\ &= \overline{B \cap H_\eta(B^c) \cap U} \cap U \\ &= H_{\eta+1}(B) \cap U. \end{aligned}$$

If λ is a limit ordinal, assuming (4) for all $\eta < \lambda$, one has $H_\lambda(A) \cap U = \bigcap_{\eta<\lambda} H_\eta(A) \cap U = \bigcap_{\eta<\lambda} H_\eta(B) \cap U = H_\lambda(B) \cap U$. ◀

B Proof of Theorem 3.6

We first prove Lemma 3.7, which states that for any set $A \subseteq X$, one has $A \setminus H_{\eta+1}(A) \in \underset{\sim}{\mathbf{D}}_\eta(X)$.

Proof of Lemma 3.7. First observe that because $H_\eta(A^c)$ is closed, one has

$$A \cap H_{\eta+1}(A) = A \cap H_\eta(A^c), \tag{5}$$
$$A \setminus H_{\eta+1}(A) = A \setminus H_\eta(A^c). \tag{6}$$

We prove the statement by induction on η. Assume the result for some η and every A. Let $A \subseteq X$. One has

$$\begin{aligned} A \setminus H_{\eta+2}(A) &= A \setminus H_{\eta+1}(A^c) \\ &= H_{\eta+1}(A^c)^c \setminus A^c \\ &= H_{\eta+1}(A^c)^c \setminus (A^c \setminus H_{\eta+1}(A^c)) \\ &= U \setminus C, \end{aligned}$$

where $U = H_{\eta+1}(A^c)^c$ is open and $C = A^c \setminus H_{\eta+1}(A^c) \in \underset{\sim}{\mathbf{D}}_\eta(X)$ by induction hypothesis. As $C \subseteq U$, one has $A = U \setminus C \in \underset{\sim}{\mathbf{D}}_{\eta+1}(X)$, which is what we wanted to prove.

The case of limit ordinals λ is proved without the induction hypothesis.

▷ Claim B.1. One has

$$A \setminus H_{\lambda+1}(A) = \bigcup_{\eta<\lambda,\text{even}} H_\eta(A) \setminus H_{\eta+1}(A^c).$$

The claim implies that $A \setminus H_{\lambda+1}(A) \in \underset{\sim}{\mathbf{D}}_\lambda(X)$: let $B_\eta = H_\eta(A)^c$ if η is even, $B_\eta = H_\eta(A^c)$ is η is odd, so the right-hand side in the claim equality can be rewritten as $\bigcup_{\eta<\lambda,\text{even}} B_{\eta+1} \setminus B_\eta$, which fits the definition of $\underset{\sim}{\mathbf{D}}_\lambda(X)$. We now prove the claim.

$$\begin{aligned} A \setminus H_{\lambda+1}(A) &= A \setminus H_\lambda(A^c) \\ &= A \setminus H_\lambda(A) \\ &= \bigcup_{\eta<\lambda,\text{even}} A \setminus H_\eta(A) \\ &= \bigcup_{\eta<\lambda,\text{even}} A \setminus H_{\eta+2}(A) \setminus (A \setminus H_\eta(A)) \\ &= \bigcup_{\eta<\lambda,\text{even}} A \cap H_\eta(A) \setminus H_{\eta+2}(A). \end{aligned}$$

Now,

$$\begin{aligned} A \cap H_\eta(A) \setminus H_{\eta+2}(A) &= A \cap H_\eta(A) \setminus H_{\eta+1}(A^c) \\ &= A \cap H_\eta(A) \setminus \overline{A^c \cap H_\eta(A)} \\ &= H_\eta(A) \setminus H_{\eta+1}(A^c). \end{aligned}$$ ◀

We now prove the other direction in Theorem 3.6.

▶ **Lemma B.2.** *If $A \in \underset{\sim}{\mathbf{D}}_\eta(X)$ then $H_{\eta+1}(A) = \emptyset$.*

Proof. Assume the result for η. Let $B \in \underset{\sim}{\mathbf{D}}_{\eta+1}(X)$. One has $B = U \setminus A$ for some open set U and some $A \in \underset{\sim}{\mathbf{D}}_\eta(X)$ with $A \subseteq U$. One has

$$\begin{aligned} H_{\eta+2}(B) &= \overline{U \setminus A \cap H_{\eta+1}(A \cup U^c)} \\ &= \overline{U \setminus A \cap H_{\eta+1}(A)} \quad \text{by Lemma 3.5} \\ &= \emptyset \quad \text{as } H_{\eta+1}(A) = \emptyset \text{ by induction.} \end{aligned}$$

If $A \in \underset{\sim}{\mathbf{D}}_\lambda(X)$ then let $(A_\eta)_{\eta<\lambda}$ be an increasing sequence of open sets such that $A = \bigcup_{\eta<\lambda,\text{even}} A_{\eta+1} \setminus A_\eta$.

It is not hard to see that for each $\eta < \lambda$, one has $A^c \cap A_\eta \in \underset{\sim}{\mathbf{D}}_{\eta+1}(X)$. By induction hypothesis, $H_{\eta+2}(A^c \cap A_\eta) = \emptyset$, so $H_{\eta+2}(A^c) \cap A_\eta = \emptyset$ by Lemma 3.5. As a result, $H_\lambda(A^c) \cap A_\eta = \emptyset$ for each $\eta < \lambda$. As $A \subseteq \bigcup_{\eta<\lambda} A_\eta$, one has $H_\lambda(A^c) \cap A = \emptyset$, so $H_{\lambda+1}(A) = \overline{A \cap H_\lambda(A^c)} = \emptyset$. ◀

NP-Completeness, Proof Systems, and Disjoint NP-Pairs

Titus Dose
Julius-Maximilians-Universität Würzburg, Germany

Christian Glaßer
Julius-Maximilians-Universität Würzburg, Germany

Abstract

The article investigates the relation between three well-known hypotheses.

H_{union}: the union of disjoint $\leq^p_m$-complete sets for NP is $\leq^p_m$-complete

H_{opps}: there exist optimal propositional proof systems

H_{cpair}: there exist $\leq^{pp}_m$-complete disjoint NP-pairs

The following results are obtained:

- The hypotheses are pairwise independent under relativizable proofs, except for the known implication $H_{opps} \Rightarrow H_{cpair}$.
- An answer to Pudlák's question for an oracle relative to which $\neg H_{cpair}$, $\neg H_{opps}$, and UP has $\leq^p_m$-complete sets.
- The converse of Köbler, Messner, and Torán's implication $NEE \cap TALLY \subseteq coNEE \Rightarrow H_{opps}$ fails relative to an oracle, where $NEE \stackrel{df}{=} NTIME(2^{O(2^n)})$.
- New characterizations of H_{union} and two variants in terms of coNP-completeness and p-producibility of the set of hard formulas of propositional proof systems.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Proof complexity; Theory of computation → Oracles and decision trees

Keywords and phrases NP-complete, propositional proof system, disjoint NP-pair, oracle

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.9

Related Version Full version available as [10]: `https://eccc.weizmann.ac.il/report/2019/050/`.

Funding *Titus Dose*: supported by the German Academic Scholarship Foundation.

1 Introduction

The three hypotheses studied in this paper came up in the context of fascinating questions. The first one states a simple closure property for the class of NP-complete sets. The second one addresses the existence of optimal propositional proof systems. It is equivalent to the existence of a finitely axiomatized theory that proves the finite consistency of each finitely axiomatized theory by a proof of polynomial length [25]. The third hypothesis is motivated and also implied by the second one.

Below we explain the context in which these hypotheses came up and discuss further connections to complete sets for promise classes like UP, to the security of public-key cryptosystems, and to complete functions for NPSV, the class of single-valued functions computable by NP-machines. At the end of this section we summarize our results.

Hypothesis H_{union}: unions of disjoint $\leq^p_m$-complete sets for NP are $\leq^p_m$-complete

The beauty of hypothesis H_{union} lies in its simplicity. It states that the class of NP-complete sets is closed under unions of disjoint sets. The question of whether H_{union} holds was raised by Selman [37] in connection with the study of self-reducible sets in NP.[1]

An interesting example for a union of disjoint NP-complete sets is the Clique-Coloring pair, which is due to Pudlák [31]:

$$\begin{aligned} C_0 &= \{(G,k) \mid G \text{ is a graph that has a clique of size } k\} \\ C_1 &= \{(G,k) \mid G \text{ is a graph that can be colored with } k-1 \text{ colors}\} \end{aligned}$$

The sets are NP-complete and disjoint, since a clique of size k cannot be colored with $k-1$ colors. C_0 and C_1 are P-separable [31], which means that there exists an $S \in \mathrm{P}$, the separator, such that $C_0 \subseteq S$ and $C_1 \subseteq \overline{S}$. The P-separability of C_0 and C_1 is a result based on deep combinatorial arguments by Lovász [26] and Tardos [38]. It implies that $C_0 \cup C_1$ is NP-complete.

Glaßer et al. [14, 17] give several equivalent formulations of H_{union} and show that the union of disjoint sets that are $\leq^p_m$-complete for NP is complete with respect to strongly nondeterministic, polynomial-time Turing reducibility. Moreover, the union is also nonuniformly polynomial-time many-one complete for NP under the assumption that NP is not infinitely-often in coNP. Moreover, Glaßer et al. [13] provide sufficient and necessary conditions for H_{union} in terms of refuters that distinguish languages $L \in \mathrm{NP}$ with $\mathrm{SAT} \cap L = \emptyset$ from $\overline{\mathrm{SAT}}$.

Hypothesis H_{opps}: there exist optimal propositional proof systems

Cook and Reckhow [6] define a propositional proof system (pps) as a polynomial-time computable function f whose range is TAUT, the set of tautologies. If $f(x) = y$, then x is a proof for y. A pps f is simulated by a pps g, if proofs in g are at most polynomially longer than proofs in f. We say that f is P-simulated by g, if additionally for a given proof in f we can compute in polynomial time a corresponding proof in g. A pps g is optimal (resp., P-optimal) if it simulates (resp., P-simulates) each pps.

The question of whether H_{opps} holds was raised by Krajíček and Pudlák [25] in an exciting context:[2] Let $\mathrm{Con}_T(n)$ denote the finite consistency of a theory T, which is the statement that T does not have proofs of contradiction of length $\leq n$. Krajíček and Pudlák [25] showed that H_{opps} is equivalent to the statement that there is a finitely axiomatized theory S that proves the finite consistency $\mathrm{Con}_T(n)$ for each finitely axiomatized theory T by a proof of polynomial length in n. In other words, H_{opps} expresses that a weak version of Hilbert's program (to prove the consistency of all mathematical theories) is possible [30].

Krajíček and Pudlák [25] also show that NE = coNE implies H_{opps} and that E = NE implies the existence of P-optimal pps. The converses of these implications do not hold relative to an oracle constructed by Verbitskii [40]. Köbler, Messner, and Torán [24] prove similar implications with weaker assumptions and reveal a connection to promise classes. For $\mathrm{EE} \stackrel{df}{=} \mathrm{DTIME}(2^{O(2^n)})$ and $\mathrm{NEE} \stackrel{df}{=} \mathrm{NTIME}(2^{O(2^n)})$ they show that $\mathrm{NEE} \cap \mathrm{TALLY} \subseteq \mathrm{coNEE}$ implies H_{opps}, which in turn implies that $\mathrm{NP} \cap \mathrm{SPARSE}$ has $\leq^p_m$-complete sets. Moreover, $\mathrm{NEE} \cap \mathrm{TALLY} \subseteq \mathrm{EE}$ implies the existence of P-optimal pps, which in turn implies that UP has $\leq^p_m$-complete sets.

---

[1] The analog of H_{union} in computability theory holds [39], since the many-one complete c.e. sets are creative [27].

[2] The analog of H_{opps} in computability theory holds trivially, since there the notion of simulation does not have any bounds for the length of proofs and hence each proof system is optimal.

Sadowski [36] proves that $\mathrm{H}_{\mathrm{opps}}$ is equivalent to the statement that the class of all easy subsets of TAUT is uniformly enumerable. Beyersdorff [2, 3, 4, 5] investigates connections between disjoint NP-pairs and pps, and in particular studies the hypotheses $\mathrm{H}_{\mathrm{cpair}}$ and $\mathrm{H}_{\mathrm{opps}}$. Pudlák [30, 32] provides comprehensive surveys on the finite consistency problem, its connection to propositional proof systems, and related open questions. In a recent paper, Khaniki [23] shows new relations between the conjectures discussed in [32] and constructs two oracles that separate several of these conjectures. In a couple of further papers [9, 8, 7], one of the authors also builds oracles separating several of the conjectures in [32].

Hypothesis $\mathrm{H}_{\mathrm{cpair}}$: there exist $\leq^{\mathrm{pp}}_{\mathrm{m}}$-complete disjoint NP-pairs

Even, Selman, and Yacobi [12, 11] show that the security of public-key cryptosystems depends on the computational complexity of certain promise problems. The latter can be written as disjoint NP-pairs, i.e., pairs (A, B) of disjoint sets $A, B \in \mathrm{NP}$. The Clique-Coloring pair mentioned above is an interesting example for a P-separable disjoint NP-pair. Even, Selman, and Yacobi [12, 11] conjecture that every disjoint NP-pair has a separator that is not $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard for NP. If the conjecture holds, then there are no public-key cryptosystems that are NP-hard to crack. Grollmann and Selman [20] observe that secure public-key cryptosystems exist only if P-inseparable disjoint NP-pairs exist.

The question of whether $\mathrm{H}_{\mathrm{cpair}}$ holds was raised by Razborov [34] in the context of pps.[3] To explain this connection we need the notions of reducibility and completeness for disjoint NP-pairs. (A, B) polynomial-time many-one reduces to (C, D), written as $(A, B) \leq^{\mathrm{pp}}_{\mathrm{m}} (C, D)$, if there is a polynomial-time computable h such that $h(A) \subseteq C$ and $h(B) \subseteq D$. A disjoint NP-pair (A, B) is $\leq^{\mathrm{pp}}_{\mathrm{m}}$-complete, if each disjoint NP-pair $\leq^{\mathrm{pp}}_{\mathrm{m}}$-reduces to (A, B). Razborov [34] defines for each pps f a corresponding disjoint NP-pair, the canonical pair of f. He shows that the canonical pair of an optimal pps is an $\leq^{\mathrm{pp}}_{\mathrm{m}}$-complete disjoint NP-pair, i.e.,

$$\mathrm{H}_{\mathrm{opps}} \;\Rightarrow\; \mathrm{H}_{\mathrm{cpair}}. \tag{1}$$

This means that the open question of whether optimal pps exist can be settled by proving that $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete disjoint NP-pairs do not exist. As we will see, (1) is the only nontrivial implication between the three hypotheses and their negations that holds relative to all oracles. For the relationship between $\mathrm{H}_{\mathrm{cpair}}$ and $\mathrm{H}_{\mathrm{opps}}$ this is shown by Glaßer et al. [16] who construct two oracles such that $\mathrm{H}_{\mathrm{cpair}}$ holds relative to both oracles, but $\mathrm{H}_{\mathrm{opps}}$ holds relative to the first one and $\neg\mathrm{H}_{\mathrm{opps}}$ relative to the second one.

Pudlák [31] further investigates the connection between pps and disjoint NP-pairs and shows that the canonical pair of the resolution proof system is symmetric. Glaßer, Selman, and Sengupta [15] characterize $\mathrm{H}_{\mathrm{cpair}}$ in several ways, e.g., by the uniform enumerability of disjoint NP-pairs and by the existence of $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete functions in NPSV. Glaßer, Selman, and Zhang [18] prove that disjoint NP-pairs and pps have identical degree structures. Moreover, they show the following statement, which connects disjoint NP-pairs, pps, and $\mathrm{H}_{\mathrm{union}}$ [19]: If $\mathrm{NP} \neq \mathrm{coNP}$ and each disjoint NP-pair (SAT, B) is strongly polynomial-time many-one equivalent to the canonical pair of a pps, then $\mathrm{H}_{\mathrm{union}}$ holds.

Our Contribution

The results of this paper improve our understanding on the three hypotheses and their relationships in the following way.

[3] The analog of $\mathrm{H}_{\mathrm{cpair}}$ in computability theory holds [35, Ch. 7., Thm XII(c)].

1. *Relativized independence of the hypotheses.* We show that $\mathrm{H}_{\mathrm{union}}$, $\mathrm{H}_{\mathrm{opps}}$, and $\mathrm{H}_{\mathrm{cpair}}$ are pairwise independent under relativizable proofs (except for the known implication $\mathrm{H}_{\mathrm{opps}} \Rightarrow \mathrm{H}_{\mathrm{cpair}}$). For each two of these hypotheses and every combination of their truth values there exists an appropriate oracle, except for $\mathrm{H}_{\mathrm{opps}} \wedge \neg\mathrm{H}_{\mathrm{cpair}}$ which is impossible. The relativized relationships between $\mathrm{H}_{\mathrm{opps}}$ and $\mathrm{H}_{\mathrm{cpair}}$ were settled by Glaßer et al. [16]. The remaining ones are obtained from an oracle by Ogiwara and Hemachandra [28], an oracle by Homer and Selman [22], and three oracles constructed in the present paper.
2. *Answer to a question by Pudlák.* The oracle in Theorem 11 answers a question by Pudlák [32] who asks for an oracle relative to which $\neg\mathrm{H}_{\mathrm{cpair}}$ and UP has $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete sets, i.e., $\mathsf{DisjNP} \not\approx \mathsf{UP}$ in the notation of [32] (see subsection 4.1 for definitions). In particular, relative to this oracle there are no P-optimal pps, but UP has $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete sets, i.e., $\mathsf{CON} \not\approx \mathsf{UP}$. This is interesting, since $\mathsf{CON} \Leftarrow \mathsf{UP}$ is a theorem [24].
3. *Possibility of* $\mathrm{H}_{\mathrm{opps}}$ *without* $\mathrm{NEE} \cap \mathrm{TALLY} \subseteq \mathrm{coNEE}$. The oracle constructed in Theorem 12 shows that the converses of the following implications by Krajíček and Pudlák [25] and Köbler, Messner, and Torán [24] fail relative to an oracle. For the implications (a) and (b) this was known by Verbitskii [40], for the other implications this is a new result. It tells us that $\mathrm{H}_{\mathrm{opps}}$ might be possible under assumptions weaker than $\mathrm{NEE} \cap \mathrm{TALLY} \subseteq \mathrm{coNEE}$.
   - **(a)** [25] $\mathrm{NE} = \mathrm{coNE} \Rightarrow \mathrm{H}_{\mathrm{opps}}$
   - **(b)** [25] $\mathrm{E} = \mathrm{NE} \Rightarrow$ there exist P-optimal pps
   - **(c)** [24] $\mathrm{NEE} \cap \mathrm{TALLY} \subseteq \mathrm{coNEE} \Rightarrow \mathrm{H}_{\mathrm{opps}}$, where $\mathrm{NEE} \stackrel{df}{=} \mathrm{NTIME}(2^{O(2^n)})$
   - **(d)** [24] $\mathrm{NEE} \cap \mathrm{TALLY} \subseteq \mathrm{EE} \Rightarrow$ there exist P-optimal pps, where $\mathrm{EE} \stackrel{df}{=} \mathrm{DTIME}(2^{O(2^n)})$
4. *Characterization of* $\mathrm{H}_{\mathrm{union}}$. We characterize $\mathrm{H}_{\mathrm{union}}$ and two variants (one is weaker, the other one stronger) in several ways. For instance, $\mathrm{H}_{\mathrm{union}}$ (resp., its stronger version) is equivalent to the statement that for each pps, the set of hard formulas is coNP-complete (resp., p-producible). The latter notion was introduced by Hemaspaandra, Hemaspaandra, and Hempel [21] for the study of inverses of NP-problems.

2 Preliminaries

Throughout this paper let Σ be the alphabet $\{0,1\}$. We denote the length of a word $w \in \Sigma^*$ by $|w|$. The empty word is denoted by ε and the i-th letter of a word w for $0 \leq i < |w|$ is denoted by $w(i)$, i.e., $w = w(0)w(1)\cdots w(|w|-1)$. For $k \leq |w|$ let $\mathrm{pr}_k(w) = w(0)\cdots w(k-1)$ be the length k prefix of w. If v is a prefix (resp., proper prefix) of w, then we write $v \sqsubseteq w$ (resp., $v \sqsubsetneq w$). A function $f : \Sigma^* \to \Sigma^*$ is length-increasing, if $|f(x)| > |x|$ for all $x \in \Sigma^*$. $\mathbb{N}$ (resp., $\mathbb{N}^+$) denotes the set of natural numbers (resp., positive natural numbers). The set of primes is denoted by $\mathbb{P} = \{2,3,5,\ldots\}$, the set of primes $\geq k$ by $\mathbb{P}^{\geq k} = \{n \in \mathbb{P} \mid n \geq k\}$. We identify Σ^* with $\mathbb{N}$ via the polynomial-time-computable, polynomial-time-invertible bijection $w \mapsto \sum_{i<|w|}(1+w(i))2^i$, which is a variant of the dyadic encoding. Hence notations, relations, and operations for Σ^* are transferred to $\mathbb{N}$ and vice versa. In particular, $|n|$ denotes the length of $n \in \mathbb{N}$. We eliminate the ambiguity of the expressions 0^i and 1^i by always interpreting them over Σ^*.

Let $\langle\cdot\rangle : \bigcup_{i\geq 0}\mathbb{N}^i \to \mathbb{N}$ be an injective, polynomial-time-computable, polynomial-time-invertible pairing function such that $|\langle u_1,\ldots,u_n\rangle| = 2(|u_1|+\cdots+|u_n|+n)$.

Given two sets A and B, $A - B = \{a \in A \mid a \notin B\}$. The complement of A relative to the universe U is denoted by $\overline{A} = U - A$. The universe will always be apparent from the context.

FP, P, and NP denote standard complexity classes [29]. Define $\mathrm{co}\mathcal{C} = \{A \subseteq \Sigma^* \mid \overline{A} \in \mathcal{C}\}$ for a class $\mathcal{C}$. Let UP denote the set of problems that can be accepted by a non-deterministic polynomial-time Turing machine that on every input x has at most one accepting path

and that accepts if and only if there exists an accepting path. TALLY denotes the class $\{A \mid A \subseteq \{0\}^*\}$. We adopt the following notions from Köbler, Messner, and Torán [24] with the remark that in the literature there exist inequivalent definitions for the double exponential time classes EE and NEE. To avoid confusion, we will recall these definitions where appropriate.

$$\mathrm{E} \stackrel{df}{=} \mathrm{DTIME}(2^{O(n)}) \qquad \mathrm{EE} \stackrel{df}{=} \mathrm{DTIME}(2^{O(2^n)})$$
$$\mathrm{NE} \stackrel{df}{=} \mathrm{NTIME}(2^{O(n)}) \qquad \mathrm{NEE} \stackrel{df}{=} \mathrm{NTIME}(2^{O(2^n)})$$

We also consider all these complexity classes in the presence of an oracle O and denote the corresponding classes by FP^O, P^O, NP^O, and so on. We use the usual oracle model where the length of queries is *not* bounded, e.g., exponential-time machines can ask queries of exponential length.

Let M be an oracle Turing machine. $M^D(x)$ denotes the computation of M on input x with D as an oracle. For an arbitrary oracle D we let $L(M^D) = \{x \mid M^D(x) \text{ accepts}\}$, where as usual if M is nondeterministic, the computation $M^D(x)$ accepts if and only if it has at least one accepting path. For a deterministic polynomial-time oracle Turing transducer F (i.e., a Turing machine computing a function), depending on the context, $F^D(x)$ either denotes the computation of F on input x with D as an oracle or the output of this computation.

If $A, B \in \mathrm{NP}$ and $A \cap B = \emptyset$, then we call (A, B) a disjoint NP-pair. The set of all disjoint NP-pairs is denoted by DisjNP.

We use the following reducibilities for sets $A, B \subseteq \Sigma^*$. $A \leq^{\mathrm{p}}_{\mathrm{m}} B$ if there exists an $f \in \mathrm{FP}$ such that $x \in A \Leftrightarrow f(x) \in B$. $A \leq^{\mathrm{p}}_{\mathrm{m,li}} B$ if $A \leq^{\mathrm{p}}_{\mathrm{m}} B$ via some length-increasing $f \in \mathrm{FP}$. For disjoint NP-pairs (A, B) and (C, D) we define specific reducibilities. $(A, B) \leq^{\mathrm{pp}}_{\mathrm{m}} (C, D)$ (resp., $(A, B) \leq^{\mathrm{pp}}_{\mathrm{m,li}} (C, D)$) if there exists an $f \in \mathrm{FP}$ (resp., a length-increasing $f \in \mathrm{FP}$) with $f(A) \subseteq C$ and $f(B) \subseteq D$. We use $A \leq^{\mathrm{pp}}_{\mathrm{m}} (C, D)$ as an abbreviation for $(A, \overline{A}) \leq^{\mathrm{pp}}_{\mathrm{m}} (C, D)$ and analogous notations for other reducibilities.

When we consider reducibilities in the presence of an oracle O, we write $\leq^{\mathrm{p},O}_{\mathrm{m}}$, $\leq^{\mathrm{p},O}_{\mathrm{m,li}}$, $\leq^{\mathrm{pp},O}_{\mathrm{m}}$, and $\leq^{\mathrm{pp},O}_{\mathrm{m,li}}$ to indicate that the reduction function has access to O.

For a complexity class $\mathcal{C}$ and some problem A, we say that A is $\leq$-hard for $\mathcal{C}$ if for all $B \in \mathcal{C}$ it holds $B \leq A$, where $\leq$ is some reducibility. A is called $\leq$-complete for $\mathcal{C}$ if A is $\leq$-hard for $\mathcal{C}$ and $A \in \mathcal{C}$. Let $\mathrm{NPC}^{\mathrm{p}}_{\mathrm{m}}$ (resp., $\mathrm{NPC}^{\mathrm{p}}_{\mathrm{m,li}}$, $\mathrm{NPC}^{\text{io-p/poly}}_{\mathrm{m}}$) be the set of problems that are $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete (resp., $\leq^{\mathrm{p}}_{\mathrm{m,li}}$-complete, $\leq^{\text{io-p/poly}}_{\mathrm{m}}$-complete) for NP, where the reducibility $\leq^{\text{io-p/poly}}_{\mathrm{m}}$ is given in Definition 6 below. If for all $A \in \mathrm{NP}$ it holds $A \leq^{\mathrm{pp}}_{\mathrm{m}} (C, D)$, then we say that (C, D) is $\leq^{\mathrm{pp}}_{\mathrm{m}}$-hard for NP.

Let SAT denote the set of satisfiable formulas and TAUT the set of tautologies. Without loss of generality, we assume that each word over Σ^* encodes a propositional formula.

▶ **Definition 1** ([6]). *A function $f \in \mathrm{FP}$ is called a* proof system *for the set* $\mathrm{ran}(f)$. *For $f, g \in \mathrm{FP}$ we say that f* is simulated by *g (resp., f* is P-simulated by *g) denoted by $f \leq g$ (resp., $f \leq^{\mathrm{p}} g$), if there exists a function π (resp., a function $\pi \in \mathrm{FP}$) and a polynomial p such that $|\pi(x)| \leq p(|x|)$ and $g(\pi(x)) = f(x)$ for all x. A function $g \in \mathrm{FP}$ is* optimal *(resp.,* P-optimal*), if $f \leq g$ (resp., $f \leq^{\mathrm{p}} g$) for all $f \in \mathrm{FP}$ with* $\mathrm{ran(f)} = \mathrm{ran(g)}$. *Corresponding relativized notions are obtained by using P^O, FP^O, and $\leq^{\mathrm{p},O}$ in the definitions above. A* propositional proof system *(pps) is a proof system for* TAUT.

▶ Remark 2. The notion of a *propositional* proof system does not have a canonical relativization. However, in view of Corollary 4 below, it is reasonable to use the following convention. We say that there exist P^O-optimal (resp., optimal) pps relative to an oracle O, if there exists a $\leq^{\mathrm{p},O}_{\mathrm{m}}$-complete $A \in \mathrm{coNP}^O$ that has a P^O-optimal (resp., optimal) proof system.

The following proposition states the relativized version of a result by Köbler, Messner, and Torán [24], which they show with a relativizable proof.

▶ **Proposition 3** ([24]). *For every oracle O, if A has a P^O-optimal (resp., optimal) proof system and $\emptyset \neq B \leq_{\mathrm{m}}^{\mathrm{p},O} A$, then B has a P^O-optimal (resp., optimal) proof system.*

▶ **Corollary 4.** *For every oracle O, if there exists a $\leq_{\mathrm{m}}^{\mathrm{p},O}$-complete $A \in \mathrm{coNP}^O$ that has a P^O-optimal (resp., optimal) proof system, then all non-empty sets in coNP^O have P^O-optimal (resp., optimal) proof systems.*

▶ **Definition 5.** *For $f \in \mathrm{FP}$ and a polynomial q, a word $y \in \mathrm{ran}(f)$ is q-hard w.r.t. the proof system f if there does not exist $x \in \Sigma^{\leq q(|y|)}$ such that $f(x) = y$. The set of elements that are q-hard w.r.t. the proof system f is denoted by f_q, i.e., $f_q = \{y \in \mathrm{ran}(f) \mid y \text{ is } q\text{-hard w.r.t. } f\}$.*

We introduce $\leq_{\mathrm{m}}^{\text{io-p/poly}}$-reducibility, which we use to study a weakened variant of $\mathrm{H}_{\mathrm{union}}$: the union of disjoint $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete sets for NP is $\leq_{\mathrm{m}}^{\text{io-p/poly}}$-complete.

P/poly is the class of sets $A \subseteq \Sigma^*$ for which there exist a $B \in \mathrm{P}$ and a function $h : \mathbb{N} \to \Sigma^*$ such that $|h(n)|$ is polynomially bounded in n and for all x it holds that $x \in A \Leftrightarrow (x, h(|x|)) \in B$. FP/poly is the class of total functions $f : \Sigma^* \to \Sigma^*$ for which there exist a $g \in \mathrm{FP}$ and a function $h : \mathbb{N} \to \Sigma^*$ such that $|h(n)|$ is polynomially bounded in n and for all x it holds that $f(x) = g(x, h(|x|))$. Two total functions $f, g : \Sigma^* \to \Sigma^*$ *agree infinitely often*, written as $f \overset{\text{io}}{=} g$, if for infinitely many n it holds that $\forall x \in \Sigma^n, f(x) = g(x)$. Two sets $A, B \subseteq \Sigma^*$ *agree infinitely often*, written as $A \overset{\text{io}}{=} B$, if their characteristic functions agree infinitely often. For a class $\mathcal{C}$ of functions or sets let io-$\mathcal{C} = \{A \mid \exists B \in \mathcal{C}, A \overset{\text{io}}{=} B\}$.

▶ **Definition 6.** *A set $A \subseteq \Sigma^*$ is* infinitely often P/poly reducible *to a set $B \subseteq \Sigma^*$, written as $A \leq_{\mathrm{m}}^{\text{io-p/poly}} B$, if there exists an $f \in$ io-FP/poly such that for all x it holds that $x \in A \Leftrightarrow f(x) \in B$.*

It should be mentioned that $\leq_{\mathrm{m}}^{\text{io-p/poly}}$ is an artificial reducibility notion (e.g., it is not transitive), which emerged from the attempt to express the right-hand side of the known implication $\mathrm{H}_{\mathrm{union}} \Rightarrow \mathrm{NP} \neq \mathrm{coNP}$ as a variant of $\mathrm{H}_{\mathrm{union}}$. In Theorem 10 we show that this is possible with $\leq_{\mathrm{m}}^{\text{io-p/poly}}$ reducibility.

In our oracle constructions we use the following notations: If a partial function t is not defined at point x, then $t \cup \{x \mapsto y\}$ denotes the extension t' of t that at x has value y and satisfies $\mathrm{dom}(t') = \mathrm{dom}(t) \cup \{x\}$.

If A is a set, then $A(x)$ denotes the characteristic function at point x, i.e., $A(x)$ is 1 if $x \in A$, and 0 otherwise. An oracle $D \subseteq \mathbb{N}$ is identified with its characteristic sequence $D(0)D(1)\cdots$, which is an ω-word. In this way, $D(i)$ denotes both, the characteristic function at point i and the i-th letter of the characteristic sequence, which are the same. A finite word w describes an oracle that is partially defined, i.e., only defined for natural numbers $x < |w|$. Occasionally, we use w instead of the set $\{i \mid w(i) = 1\}$ and write for example $A = w \cup B$, where A and B are sets. In particular, for an oracle Turing machine M, the notation $M^w(x)$ refers to $M^{\{i|w(i)=1\}}(x)$ (hence, oracle queries that w is not defined for are answered by "no"). Using w instead of $\{i \mid w(i) = 1\}$ additionally allows us to define the following notion: for a nondeterministic oracle Turing machine M, the computation $M^w(x)$ *definitely accepts* if it contains a path that accepts and all queries on this path are $< |w|$. The computation $M^w(x)$ *definitely rejects* if all paths reject and all queries are $< |w|$. We say that the computation $M^w(x)$ *is definite* if it definitely accepts or definitely rejects. Similarly, for a deterministic oracle Turing transducer F, the computation $F^w(x)$ is *definite* if all its queries are $< |w|$.

3 Are Unions of Disjoint NP-Complete Sets NP-Complete?

It is difficult to find out whether $\mathrm{H}_{\mathrm{union}}$ is true or not, since each outcome solves a long standing open problem:

$\mathrm{H}_{\mathrm{union}}$ is true $\Rightarrow$ NP $\neq$ coNP

$\mathrm{H}_{\mathrm{union}}$ is false $\Rightarrow$ P-inseparable disjoint NP-pairs exist if and only if P $\neq$ NP

Therefore, researchers approach the hypothesis $\mathrm{H}_{\mathrm{union}}$ by proving equivalent, necessary, and sufficient conditions. This section continues this program as follows. In subsection 3.1 we investigate a stronger variant of $\mathrm{H}_{\mathrm{union}}$, in 3.2 the original hypothesis, and in 3.3 a weaker variant. We characterize $\mathrm{H}_{\mathrm{union}}$ and its variants in several ways, e.g., in terms of p-producibility or coNP-completeness of the set of hard formulas of pps. Within each subsection all hypotheses are equivalent and hence the following implications hold.

$$\begin{array}{ccccc} \text{hypotheses in subsect. 3.1} & \Rightarrow & \text{hypotheses in subsect. 3.2} & \Rightarrow & \text{hypotheses in subsect. 3.3} \\ & & \Updownarrow & & \Updownarrow \\ & & \mathrm{H}_{\mathrm{union}} & & \mathrm{NP} \neq \mathrm{coNP} \end{array}$$

Note that under the assumption that all sets in $\mathrm{NPC}^{\mathrm{p}}_{\mathrm{m}}$ are complete w.r.t. length-increasing reductions (which holds for example under the Berman-Hartmanis conjecture), all hypotheses in the subsections 3.1 and 3.2 are equivalent.

3.1 Length-Increasing Polynomial-Time Reducibility

Consider the hypothesis that the union of SAT with a disjoint $B \in \mathrm{NP}$ is $\leq^{\mathrm{p}}_{\mathrm{m,li}}$-complete for NP. We show that this hypothesis can be characterized in terms of the p-producibility of the set of hard formulas of pps. The notion of p-producibility was introduced by Hemaspaandra, Hemaspaandra, and Hempel [21].

▶ **Definition 7** ([21]). *A set A is p-producible if and only if there is some $f \in \mathrm{FP}$ with $|f(x)| \geq |x|$ and $f(x) \in A$ for all x.*

▶ **Theorem 8.** *The following statements are equivalent:*

1. *For all $B \in \mathrm{NP}$ with $\mathrm{SAT} \cap B = \emptyset$ it holds $\mathrm{SAT} \cup B \in \mathrm{NPC}^{\mathrm{p}}_{\mathrm{m,li}}$.*
2. *For all $A, B \in \mathrm{NPC}^{\mathrm{p}}_{\mathrm{m,li}}$ with $A \cap B = \emptyset$ it holds $A \cup B \in \mathrm{NPC}^{\mathrm{p}}_{\mathrm{m,li}}$.*
3. *f_q is p-producible for all pps f and all polynomials q.*

Proof. 1 $\Rightarrow$ 2: Let $A, B \in \mathrm{NPC}^{\mathrm{p}}_{\mathrm{m,li}}$ be disjoint and $\mathrm{SAT} \leq^{\mathrm{p}}_{\mathrm{m,li}} A$ via a length-increasing $f \in \mathrm{FP}$. $B' = f^{-1}(B)$ is in NP and disjoint to SAT and hence $\mathrm{SAT} \cup B' \in \mathrm{NPC}^{\mathrm{p}}_{\mathrm{m,li}}$. $\mathrm{SAT} \cup B' \leq^{\mathrm{p}}_{\mathrm{m,li}} A \cup B$ via f and thus $A \cup B \in \mathrm{NPC}^{\mathrm{p}}_{\mathrm{m,li}}$.

2 $\Rightarrow$ 3: By assumption, NP $\neq$ coNP. Let f be a pps, q a polynomial, and define

$$B = \{\varphi \mid f(y) = \neg\varphi \text{ for some } y \text{ with } |y| \leq q(|\neg\varphi|)\}.$$

$B \cap \mathrm{SAT} = \emptyset$ and $\mathrm{SAT} \cup B \subsetneq \Sigma^*$. For $A' = 0\mathrm{SAT} \cup 1B$ and $B' = 1\mathrm{SAT} \cup 0B$ it holds $A' \cap B' = \emptyset$ and $A', B' \in \mathrm{NPC}^{\mathrm{p}}_{\mathrm{m,li}}$. By 2, $A' \cup B' = \{0,1\}(\mathrm{SAT} \cup B) \in \mathrm{NPC}^{\mathrm{p}}_{\mathrm{m,li}}$. In particular, $\mathrm{SAT} \leq^{\mathrm{p}}_{\mathrm{m,li}} \{0,1\}(\mathrm{SAT} \cup B)$. Hence $\mathrm{SAT} \leq^{\mathrm{p}}_{\mathrm{m}} \mathrm{SAT} \cup B$ via $h_1 \in \mathrm{FP}$ with $|x| \leq |h_1(x)|$. Let $h_2 \in \mathrm{FP}$ be length-increasing such that $\mathrm{SAT} \leq^{\mathrm{p}}_{\mathrm{m,li}} \mathrm{SAT}$ via h_2. Thus $\mathrm{SAT} \leq^{\mathrm{p}}_{\mathrm{m,li}} \mathrm{SAT} \cup B$ via $h(x) = h_1(h_2(x))$. We claim that f_q is p-producible via the length-increasing $g(x) = \neg h(x \wedge \neg x)$: As $h(x \wedge \neg x) \notin \mathrm{SAT} \cup B$, $g(x)$ is a tautology. If $g(x) \notin f_q$, then there exists y with $|y| \leq q(|g(x)|)$ and $f(y) = g(x) = \neg h(x \wedge \neg x)$. Hence $h(x \wedge \neg x) \in B$, a contradiction.

$3 \Rightarrow 1$: Choose B according to 1. Consider $B' = \{x \mid x \in B \text{ or } \exists z\, |z| \leq |x| \text{ and } x \vee z \in B\}$ and observe $B' \in \mathrm{NP}$, $B \subseteq B'$, and $B' \cap \mathrm{SAT} = \emptyset$. Let M be an NP-machine with $L(M) = B'$ running in polynomial time q. The following f is a pps.

$$\langle x, z\rangle \mapsto \begin{cases} x & M \text{ accepts } \neg x \text{ on path } z \text{ or } (|z| \geq 2^{|x|} \text{ and } x \text{ is a tautology}) \\ \mathrm{True} & \text{otherwise.} \end{cases}$$

Let q' be a polynomial such that $|\neg x| \leq q'(|x|)$. Choose $r(n) = 2 \cdot (q(q'(n)) + n + 1)$. By 3, f_r is p-producible via some $g \in \mathrm{FP}$ with $|g(x)| \geq |x|$. Consider the length-increasing $h \in \mathrm{FP}$ with $h(x) = \neg g(x) \vee x$. We show $\mathrm{SAT} \leq^{\mathrm{pp}}_{\mathrm{m,li}} (\mathrm{SAT}, \overline{\mathrm{SAT} \cup B})$ via h, which implies $\mathrm{SAT} \leq^{\mathrm{p}}_{\mathrm{m,li}} \mathrm{SAT} \cup B$ via h. As $g(x)$ is a tautology, $x \in \mathrm{SAT} \Leftrightarrow h(x) \in \mathrm{SAT}$. It remains to show $x \notin \mathrm{SAT} \Rightarrow h(x) \notin B$. Let $x \notin \mathrm{SAT}$. If $h(x) = \neg g(x) \vee x \in B$, then due to $|x| \leq |\neg g(x)|$ it holds $\neg g(x) \in B'$. Hence there is some path z such that M accepts $\neg g(x)$ on path z. Thus $|z| \leq q(q'(|g(x)|))$. Consequently, $f(\langle g(x), z\rangle) = g(x)$ and $|\langle g(x), z\rangle| \leq r(|g(x)|)$, in contradiction to $g(x) \in f_r$. ◀

3.2 Polynomial-Time Reducibility

We consider the hypothesis that the union of SAT with a disjoint $B \in \mathrm{NP}$ is $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete for NP. This is equivalent to $\mathrm{H}_{\mathrm{union}}$. We prove one more characterization stating that for each pps f the set of formulas hard for f is coNP-complete. In the following theorem, the equivalence $1 \Leftrightarrow 2$ was shown in [14].

▶ **Theorem 9.** *The following statements are equivalent:*

1. *For all $B \in \mathrm{NP}$ with $\mathrm{SAT} \cap B = \emptyset$ it holds $\mathrm{SAT} \cup B \in \mathrm{NPC}^{\mathrm{p}}_{\mathrm{m}}$.*
2. *For all $A, B \in \mathrm{NPC}^{\mathrm{p}}_{\mathrm{m}}$ with $A \cap B = \emptyset$ it holds $A \cup B \in \mathrm{NPC}^{\mathrm{p}}_{\mathrm{m}}$.*
3. *f_q is $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete for coNP for all pps f and all polynomials q.*

Proof. We argue for "$1 \Rightarrow 3$". By definition, $f_q = \{x \in \mathrm{TAUT} \mid \neg\exists z \in \Sigma^{\leq q(|x|)} f(z) = x\}$ and hence $f_q \in \mathrm{coNP}$. Let $B = \{x \in \Sigma^* \mid \exists z \in \Sigma^{\leq q(|\neg x|)} f(z) = \neg x\}$.

Observe that $B \in \mathrm{NP}$ and $\mathrm{SAT} \cap B = \emptyset$. By assumption, $\mathrm{SAT} \cup B \in \mathrm{NPC}^{\mathrm{p}}_{\mathrm{m}}$ and hence $\overline{\mathrm{SAT} \cup B}$ is $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete for coNP. Note $\overline{\mathrm{SAT} \cup B} = \{x \in \Sigma^* \mid \neg x \in \mathrm{TAUT} \wedge \neg\exists z \in \Sigma^{\leq q(|\neg x|)} f(z) = \neg x\}$. Thus $x \in \overline{\mathrm{SAT} \cup B} \Leftrightarrow \neg x \in f_q$ and hence f_q is $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete for coNP.

"$3 \Rightarrow 1$": Let $B \in \mathrm{NP}$ such that $\mathrm{SAT} \cap B = \emptyset$ and let M be a nondeterministic polynomial-time machine that accepts B. Choose a polynomial q such that for all $x \in \Sigma^*$ and all accepting paths y of $M(\neg x)$ it holds that $|\langle x, y\rangle| \leq q(|x|)$. Let

$$f(z) = \begin{cases} x, & \text{if } z = \langle x, y\rangle,\ |y| < 2^{|x|}, \text{ and } y \text{ is an accepting path of } M(\neg x) \\ x, & \text{if } z = \langle x, y\rangle,\ |y| = 2^{|x|}, \text{ and } x \in \mathrm{TAUT} \\ \mathrm{True}, & \text{otherwise.} \end{cases}$$

Observe that f is a pps. By assumption, the set $f_q = \{x \in \mathrm{TAUT} \mid \neg\exists z \in \Sigma^{\leq q(|x|)} f(z) = x\}$ is $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete for coNP. Observe $f_q \cap \Sigma^{\geq n} = \{x \in \mathrm{TAUT} \mid \neg x \notin B\} \cap \Sigma^{\geq n}$ for sufficiently large $n \in \mathbb{N}$. Hence for all $x \in \Sigma^{\geq n}$ it holds that $x \in f_q \Leftrightarrow \neg x \in \overline{\mathrm{SAT} \cup B}$. In the case $\overline{\mathrm{SAT} \cup B} \neq \emptyset$ this shows $f_q \leq^{\mathrm{p}}_{\mathrm{m}} \overline{\mathrm{SAT} \cup B}$ and hence $\mathrm{SAT} \cup B$ is $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete for NP.

It remains to argue that the case $\overline{\mathrm{SAT} \cup B} = \emptyset$ is not possible. If $\overline{\mathrm{SAT} \cup B} = \emptyset$, then $\mathrm{NP} = \mathrm{coNP}$ and hence there exists a polynomially bounded pps f'. Thus for some polynomial q' it holds $f'_{q'} = \emptyset$, which is not $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete for coNP, in contradiction to our assumption. ◀

3.3 Infinitely Often P/poly Reducibility

Consider the hypothesis that the union of SAT with a disjoint $B \in \mathrm{NP}$ is $\leq_{\mathrm{m}}^{\text{io-p/poly}}$-complete for NP. We show that this hypothesis is equivalent to $\mathrm{NP} \neq \mathrm{coNP}$.

▶ **Theorem 10.** *The following statements are equivalent:*

1. *For all* $B \in \mathrm{NP}$ *with* $\mathrm{SAT} \cap B = \emptyset$ *it holds* $\mathrm{SAT} \cup B \in \mathrm{NPC}_{\mathrm{m}}^{\text{io-p/poly}}$.
2. *For all* $A, B \in \mathrm{NPC}_{\mathrm{m}}^{\mathrm{p}}$ *with* $A \cap B = \emptyset$ *it holds* $A \cup B \in \mathrm{NPC}_{\mathrm{m}}^{\text{io-p/poly}}$.
3. $\mathrm{NP} \neq \mathrm{coNP}$ *(i.e., polynomially bounded pps do not exist).*

4 Oracle Constructions

4.1 An Oracle for P = UP and ¬$\mathrm{H}_{\mathrm{cpair}}$

We construct an oracle O relative to which $\mathrm{P} = \mathrm{UP}$ and $\neg\mathrm{H}_{\mathrm{cpair}}$. This answers open questions by Pudlák [32], who lists several conjectures and asks for equivalence proofs and oracles relative to which conjectures are different. Among these are:

$$\begin{aligned}
\mathsf{DisjNP} &\stackrel{df}{=} \text{“there are no } \leq_{\mathrm{m}}^{\mathrm{pp}}\text{-complete disjoint NP-pairs (i.e., } \neg\mathrm{H}_{\mathrm{cpair}}\text{)”}\\
\mathsf{CON} &\stackrel{df}{=} \text{“there are no P-optimal propositional proof systems”}\\
\mathsf{SAT} &\stackrel{df}{=} \text{“NP-complete sets do not have P-optimal proof systems”}\\
\mathsf{UP} &\stackrel{df}{=} \text{“UP does not have } \leq_{\mathrm{m}}^{\mathrm{p}}\text{-complete sets”}\\
\mathsf{NP} \cap \mathsf{coNP} &\stackrel{df}{=} \text{“NP} \cap \text{coNP does not have } \leq_{\mathrm{m}}^{\mathrm{p}}\text{-complete sets”}
\end{aligned}$$

Relative to O, DisjNP and NP ∩ coNP hold, but UP does not. Hence DisjNP and NP ∩ coNP do not imply UP. Moreover, relative to O, also the following conjectures mentioned by Pudlák [32] do not imply UP (as they are implied by DisjNP relative to all oracles): CON, CON ∨ SAT, and $\mathrm{P} \neq \mathrm{NP}$. The fact that relative to O, CON does not imply UP is of particular interest as the converse implication holds relative to all oracles.

▶ **Theorem 11.** *There exists an oracle* O *with the following properties.*

1. DisjNP^O *does not have* $\leq_{\mathrm{m}}^{\mathrm{pp},O}$*-complete pairs.*
2. $\mathrm{NP}^O \cap \mathrm{coNP}^O$ *does not have* $\leq_{\mathrm{m}}^{\mathrm{p},O}$*-complete sets.*
3. $\mathrm{P}^O = \mathrm{UP}^O$.

Sketch of the construction: For simplicity, we argue only for 1 and 3. Let $M_0, M_1, \ldots$ be a standard enumeration of nondeterministic, polynomial-time oracle Turing machines and let $F_0, F_1, \ldots$ be a standard enumeration of deterministic, polynomial-time oracle Turing transducers. We assume that for all i the running times of M_i and F_i are bounded by the polynomial $n^i + i$. Adopting an idea by Baker, Gill and Solovay [1], we start with a PSPACE-complete oracle that consists of words of odd length. During the construction we add words of lengths $e(n)$ to the oracle, where $e(0) = 2$ and $e(n+1) = 2^{2^{e(n)}}$. Since $e(n)$ is even, the PSPACE-complete set that we started with will not be damaged.

On the one hand, the construction tries to prevent that $L(M_i)$ and $L(M_j)$ are disjoint. If this is not possible, then M_i and M_j inherently accept disjoint sets. In this case, we make sure that there exists a disjoint NP-pair (A_{ij}, B_{ij}) that does not $\leq_{\mathrm{m}}^{\mathrm{pp}}$-reduce to $(L(M_i), L(M_j))$. This prevents the existence of complete disjoint NP-pairs. On the other hand, we try to prevent that M_i has the uniqueness property “for all x, the computation $M_i(x)$ has at most one accepting path”. If this is not possible, then M_i inherently has the uniqueness property, which allows us to show $L(M_i) \in \mathrm{P}$.

On the technical side, we maintain a growing collection t of properties that we demand in the further construction. If an oracle satisfies the properties defined by t, then we call it t-valid. The collection t contains properties of the following style:

V1: The oracle constructed so far guarantees that $L(M_i) \cap L(M_j) \neq \emptyset$ for all extensions of the oracle.
V2: It is impossible to reach $L(M_i) \cap L(M_j) \neq \emptyset$ and for the oracle constructed so far we have $A_{ij} \cap B_{ij} = \emptyset$. (In the future we restrict to extensions that maintain this property.)
V3: The oracle constructed so far guarantees that for all extensions of the oracle, M_i does not have the uniqueness property.
V4: It is impossible to destroy the uniqueness property of M_i.

The construction successively settles the following tasks:

- task (i, j): If possible, then realize V1 for the pair $(L(M_i), L(M_j))$, otherwise, V2 holds.
- task i: If possible, then realize V3 for M_i, otherwise, V4 holds.
- task (i, j, r): Make sure that F_r does not realize a reduction $(A_{ij}, B_{ij}) \leq_{\mathrm{m}}^{\mathrm{pp}} (L(M_i), L(M_j)))$.

The tasks (i, j) and (i, j, r) make sure that relative to the final oracle, $L(M_i) \cap L(M_j) \neq \emptyset$ or $(L(M_i), L(M_j))$ is not $\leq_{\mathrm{m}}^{\mathrm{pp}}$-complete. The task i ensures that machines having the uniqueness property are very special. An adaption of an argument by Rackoff [33] yields that these machines accept sets in P, hence P = UP.

4.2 An Oracle for $\mathrm{H}_{\mathsf{union}}$ and $\mathrm{H}_{\mathsf{opps}}$

This section constructs an oracle O relative to which the implication $\mathrm{H}_{\mathrm{opps}} \Rightarrow \neg \mathrm{H}_{\mathrm{union}}$ is false. Theorem 22 provides the analogous for the converse implication.

In addition, relative to O there exists a tally set in NEE − coNEE, where $\mathrm{NEE} \stackrel{df}{=} \mathrm{NTIME}(2^{O(2^n)})$. It shows that two conditions which are sufficient for the existence of an optimal (resp., a P-optimal) pps [24] are not necessary relative to O.

▶ **Theorem 12.** *There exists an oracle O with the following properties.*

1. *There exists a P^O-optimal propositional proof system f.*
2. *If A is $\leq_{\mathrm{m}}^{\mathrm{p},O}$-complete for NP^O and disjoint from $B \in \mathrm{NP}^O$, then $A \cup B$ is $\leq_{\mathrm{m}}^{\mathrm{p},O}$-complete for NP^O.*
3. $\mathrm{NEE}^O \cap \mathrm{TALLY} \not\subseteq \mathrm{coNEE}^O$, *where* $\mathrm{NEE}^O \stackrel{df}{=} \mathrm{NTIME}^O(2^{O(2^n)})$.

Proof. We only prove statements 1 and 2. Statement 3 follows (in a nontrivial way) from the construction below. Let $M_1, M_3, M_5, \ldots$ be a standard enumeration of nondeterministic, polynomial-time oracle Turing machines. Let $F_2, F_4, F_6, \ldots$ be a standard enumeration of deterministic, polynomial-time oracle Turing transducers. We assume that the running time of M_i for i odd (resp., F_j for $j > 0$ even) is bounded by the polynomial $n^i + i$ (resp., $n^j + j$).

For a (possibly partial) oracle D we define sets K^D and $K_\vee^D$.

$$K^D = \{\langle 0^i, 0^j, x\rangle \mid i \text{ is odd and } M_i^D(x) \text{ accepts within } j \text{ steps}\}$$
$$K_\vee^D = \{\langle z_1, \ldots, z_n\rangle \mid z_1 \in K^D \vee \cdots \vee z_n \in K^D\}$$

▷ Claim 13. For partial oracles v and w and all $y \leq \min(|v|, |w|)$, if $\mathrm{pr}_y(v) = \mathrm{pr}_y(w)$, then $K^w(y) = K^v(y)$ and $K_\vee^w(y) = K_\vee^v(y)$.

Proof. It suffices to show $K^w(y) = K^v(y)$. We may assume $y = \langle 0^i, 0^j, x\rangle$ for suitable i, j, x, since otherwise, $K^w(y) = K^v(y) = 0$. For each q that is queried within the first j steps of $M_i^w(x)$ or $M_i^v(x)$ it holds that $|q| \leq j < |y|$ and thus $q < y$. Hence these queries are answered the same way relative to w and v, showing that $M_i^w(x)$ accepts within j steps if and only if $M_i^v(x)$ accepts within j steps. ◁

K^D and $K_\vee^D$ are $\leq_{\mathrm{m}}^{\mathrm{p},D}$-complete for NP^D and their complements are $\leq_{\mathrm{m}}^{\mathrm{p},D}$-complete for coNP^D. We construct the oracle such that $\overline{K_\vee^D}$ has a P^O-optimal proof system $f \in \mathrm{FP}^O$. As $\overline{K_\vee^O}$ is $\leq_{\mathrm{m}}^{\mathrm{p},O}$-complete for coNP^O, this implies the first statement of the theorem.

For a (possibly partial) oracle D let

$$E^D = \{0^n \mid \exists x \in D \text{ such that } |x| = n\}$$

and observe that $E^D \in \mathrm{NP}^D$. Choose $e \geq 2$ such that $L(M_e^D) = E^D$ for all (possibly partial) oracles D and let $v_n = \langle 0^e, 0^{n^e+e}, 0^n\rangle$. Hence $v_n \in K^D$ if and only if $M_e^D(0^n)$ accepts, i.e., $v_n \in K^D \Leftrightarrow 0^n \in E^D$.

For $i \in 2\mathbb{N}^+$ and $x, y \in \mathbb{N}$ let $c(i,x,y) = \langle 0^i, 0^{(|x|^i+i)^{2ie}}, x, y\rangle$. These words are used to encode proofs into the oracle: if the oracle contains the codeword $c(i,x,y)$, then this means $F_i(x) = y$ and $y \notin K_\vee$, i.e., $c(i,x,y)$ is a proof for $y \notin K_\vee$.

▷ Claim 14. The following holds for all partial oracles w, all $i \in 2\mathbb{N}^+$ and $x, y \in \mathbb{N}$.

1. If $c(i,x,y) \leq |w|$, then $F_i^w(x)$ is definite and $F_i^v(x) = F_i^w(x) < |w|$ for all $v \sqsupseteq w$.
2. If $c(i,x,y) \leq |w|$, then $F_i^w(x)$ is definite and $F_i^w(x) \in K_\vee^w \Leftrightarrow F_i^v(x) \in K_\vee^v$ for all $v \sqsupseteq w$.

Proof. 1: $F_i^w(x)$ is definite, since for each q queried by $F_i^w(x)$ it holds that $|q| \leq |x|^i + i < |c(i,x,y)|$ and hence $q < c(i,x,y) \leq |w|$. The same argument shows $F_i^v(x) = F_i^w(x) < |w|$.
2: Follows from Claims 14.1 and 13. ◁

Preview of construction: On the one hand, the construction tries to prevent that F_i is a proof system for $\overline{K_\vee}$. If this is not possible, then F_i inherently is a proof system for $\overline{K_\vee}$. In this case, the codewords $c(i,x,y)$ are used to encode F_i-proofs into the oracle. These encodings finally yield a P-optimal proof system for $\overline{K_\vee}$. On the other hand, the construction also tries to prevent that M_i accepts a set disjoint from $K_\vee$. If this is not possible, then M_i inherently accepts a set disjoint from $K_\vee$. In this case, there will be a prime p such that the words v_{p^k} for $k \geq 1$ are neither in K nor in $L(M_i)$. It even holds $\langle v_{p^k}, u_1, \ldots, u_n\rangle \notin L(M_i)$ for all $u = \langle u_1, \ldots, u_n\rangle$ of length $\leq |v_{p^k}|$. This means that the v_{p^k} are difficult instances for M_i, since there is no linear-size proof u that allows M_i to recognize that $v_{p^k} \notin K$. Hence adding a sufficiently large v_{p^k} to an instance u does not change the membership to $K_\vee$, but guarantees that the result is not in $L(M_i)$. This yields a reduction $K_\vee \leq_{\mathrm{m}}^{\mathrm{p}} K_\vee \cup L(M_i)$ and implies that $K_\vee \cup L(M_i)$ is NP-complete.

During the construction we maintain a growing list of properties. This list belongs to the set $\mathcal{T} = \{(m_1, \ldots, m_n) \mid n \geq 0, m_1, \ldots, m_n \in \mathbb{N}, \text{ and } m_i < m_j \text{ for all } i < j \text{ with } m_j \neq 0\}$. If a partial oracle satisfies the properties defined by a list t, then we call it t-valid. For a list $t = (m_1, \ldots, m_n)$ and $a \in \mathbb{N}$ let $t(i) = m_i$, $|t| = n$, and $t + a = (m_1, \ldots, m_n, a)$. If the list t is a prefix of the list t', then we write $t \sqsubseteq t'$. We start with the empty list $t_0 = ()$, which defines no property. By successively appending an element we obtain lists t_1, t_2, and so on.

A partial oracle $w \in \Sigma^*$ is t-valid, where $t \in \mathcal{T}$, if the following holds:

V1: $w \subseteq \{c(i,x,y) \mid i \in 2\mathbb{N}^+ \text{ and } x, y \in \mathbb{N}\} \cup \{v \mid |v| = p^k \text{ for } p \in \mathbb{P}^{\geq 41} \text{ and } k \geq 1\}$
(meaning: the oracle contains only codewords $c(i,x,y)$ and words of length p^k)

V2: For all $c(i,x,y) \in w$ with $i \in 2\mathbb{N}^+$ and $x, y \in \mathbb{N}$ it holds that $F_i^w(x) = y \notin K_\vee^w$.
(meaning: if the oracle contains the codeword $c(i,x,y)$, then $F_i^w(x)$ outputs $y \notin K_\vee^w$; hence $c(i,x,y) \in w$ is a proof for $y \notin K_\vee^w$)

V3: For all positive even $i \leq |t|$ it holds that $t(i) \in 2\mathbb{N}$ and:

a. If $t(i) = m > 0$, then $c(i,x,y) \in w$ for all $x, y \in \mathbb{N}$ with $F_i^w(x) = y$ and $m \leq c(i,x,y) < |w|$.
(meaning: the oracle maintains codewords for F_i, i.e., if x is large enough and $F_i^w(x)$ outputs y, then w contains a proof for this, namely the codeword $c(i,x,y)$)

b. If $t(i) = 0$, then there exists x such that $F_i^w(x)$ is definite and outputs $y < |w|$ with $y \in K_\vee^w$.
(meaning: F_i is not a proof system for $\overline{K_\vee}$ relative to all extensions of w)

V4: For all odd $i \leq |t|$ it holds that $t(i) \in \{0\} \cup \mathbb{P}^{\geq 41}$ and:

a. If $t(i) = p > 0$, then $\{x \in w \mid |x| = p^k \text{ for } k \geq 1\} = \emptyset$ and for all positive even $j < i$ with $t(j) = 0$ it holds that $\{c(j,x,y) \in w \mid x, y \in \mathbb{N} \text{ and } |c(j,x,y)| \geq p\} = \emptyset$.
(meaning: the first part says $0^{p^k} \notin E^w$ and hence $v_{p^k} \notin K^w$ for all $k \geq 1$; the second part says that if F_j is not a proof system for $\overline{K_\vee}$ and has a smaller index than M_i, then the oracle contains no codewords $c(j,\cdot,\cdot)$ of length $\geq p$)

b. If $t(i) = 0$, then there exists $x < |w|$ such that $x \in K_\vee^w$ and $M_i^w(x)$ definitely accepts.
(meaning: M_i is not disjoint from $K_\vee$ relative to all extensions of w)

▷ Claim 15. The following holds in reference to the definition of t-valid.

1. In V1, the two sets are disjoint.
2. In V2, $F_i^w(x)$ is definite and $F_i^v(x) = y \notin K_\vee^v$ for all $v \sqsupseteq w$.
3. In V3a, $F_i^w(x)$ is definite.
4. In V3b, $y \in K_\vee^v$ for all $v \sqsupseteq w$.
5. In V4b, $x \in K_\vee^v$ for all $v \sqsupseteq w$.

Proof. V1: The union is disjoint, since $|c(i,x,y)|$ is even. V2+V3a: Follows from Claim 14. V3b+V4b: Follows from Claim 13. ◁

▷ Claim 16. Let u and w be t-valid. If $u \sqsubseteq v \sqsubseteq w$, then v is t-valid.

Proof. We show that v satisfies V1–V4. When we consider w and v as sets, then $v \subseteq w$. Therefore, v satisfies V1 and V4a. Moreover, $v \sqsubseteq w$ and Claim 14 imply that v satisfies V2 and V3a. Since u is t-valid, it satisfies V3b and V4b. From $u \sqsubseteq v$, Claim 15.4, and Claim 15.5 it follows that v satisfies V3b and V4b. ◁

Oracle construction: Let $t_0 = ()$ be the empty list and $w_0 = \varepsilon$, which is t_0-valid. We construct a sequence $t_0 \sqsubsetneq t_1 \sqsubsetneq \cdots$ of lists from $\mathcal{T}$ and a sequence $w_0 \sqsubsetneq w_1 \sqsubsetneq \cdots$ of partially defined oracles such that $|t_s| = s$ and w_s is t_s-valid. The final oracle is $O = \lim_{s\to\infty} w_s$. We describe step $s > 0$, which starts with a list t_{s-1} of length $s-1$ and a t_{s-1}-valid w_{s-1} and which defines a list $t_s \sqsupsetneq t_{s-1}$ of length s and a t_s-valid $w_s \sqsupsetneq w_{s-1}$.

- s even: If there is a t_{s-1}-valid $v \sqsupsetneq w_{s-1}$ such that for some x, $F_s^v(x)$ is definite and has an output $y < |v|$ with $y \in K_\vee^v$, then let $w_s = v$ and $t_s = t_{s-1} + 0$. Otherwise, choose $b \in \{0,1\}$ such that $w_{s-1}b$ is t_{s-1}-valid, let $w_s = w_{s-1}b$ and $t_s = t_{s-1} + m$ for an even $m > |w_s|$ that is greater than all elements in t_{s-1}.
(meaning: if possible, force that F_s is not a proof system for $\overline{K_\vee}$ relative to all extensions of v; otherwise, we start to maintain codewords for F_s, i.e., if x is large enough and $F_s(x)$ outputs y, then the oracle contains a proof for this, namely the codeword $c(s,x,y)$)
- s odd: If there is a t_{s-1}-valid $v \sqsupsetneq w_{s-1}$ such that for some $x < |v|$, $x \in K_\vee^v$ and $M_s^v(x)$ definitely accepts, then let $w_s = v$ and $t_s = t_{s-1} + 0$. Otherwise, let $w_s = w_{s-1}b$ for $b \in \{0,1\}$ such that $w_{s-1}b$ is t_{s-1}-valid and $t_s = t_{s-1} + p$ for $p \in \mathbb{P}^{\geq 41}$ large enough such that $(16|v_{p^k}|)^s < 2^{p^k}$ for all $k \in \mathbb{N}^+$, $p > |w_s|$, and p is greater than all elements in t_{s-1}.
(meaning: force $L(M_s) \cap K_\vee \neq \emptyset$ if possible; otherwise, choose a suitable prime p and make sure that the oracle contains no elements of length p^k and hence $v_{p^k} \notin K$ for all $k \geq 1$; the step corresponds to V4)

The subsequent claims refer to the construction above. We start by showing that the construction is possible and how one can extend a t_s-valid $w \sqsupseteq w_s$ by one bit. The proof can be found in [10].

▷ Claim 17. Let $s \in \mathbb{N}$. The choices of w_s and t_s are possible and w_s is t_s-valid. Moreover, for each t_s-valid $w \sqsupseteq w_s$ and $z = |w|$ the following holds.

1. If $z = c(i,x,y)$ for $i \in 2\mathbb{N}^+$, $x,y \in \mathbb{N}$ such that $i \leq s$, $t_s(i) > 0$, and $z \geq t_s(i)$, then:
   a. if $F_i^w(x) = y$, then $w1$ is t_s-valid and $w0$ is not.
   b. if $F_i^w(x) \neq y$, then $w0$ is t_s-valid and $w1$ is not.
2. If $z = c(i,x,y)$ for $i \in 2\mathbb{N}^+$, $x,y \in \mathbb{N}$ such that $i \leq s$ and $t_s(i) = 0$, then:
   a. $w0$ is t_s-valid.
   b. if $F_i^w(x) = y \notin K_\vee^w$ and there is no odd i' such that $i < i' \leq s$, $t_s(i') = p \in \mathbb{P}^{\geq 41}$, and $|z| \geq p$, then $w1$ is t_s-valid.
3. If $z = c(i,x,y)$ for $i \in 2\mathbb{N}^+$, $x,y \in \mathbb{N}$ such that $i > s$, then:
   a. $w0$ is t_s-valid.
   b. if $F_i^w(x) = y \notin K_\vee^w$, then $w1$ is t_s-valid.
4. If $|z| = p^k$ for $p \in \mathbb{P}^{\geq 41}$, $p \notin t_s$, and $k \geq 1$, then $w0$ and $w1$ are t_s-valid.
5. In all other cases $w0$ is t_s-valid.

▷ Claim 18. $M_s^O(\langle v_{p^k}, u_1, \ldots, u_n\rangle)$ rejects for all odd s with $t_s(s) = p \in \mathbb{P}^{\geq 41}$, all $k \in \mathbb{N}^+$, and all $u = \langle u_1, \ldots, u_n\rangle$ with $|u| \leq |v_{p^k}|$.

Proof. We assume that $M_s^O(u')$ accepts for $u' = \langle v_{p^k}, u_1, \ldots, u_n\rangle$ and show a contradiction. Choose $j > s$ large enough such that $M_s^{w_j}(u')$ definitely accepts, $|w_j| > u'$, and $|w_j| > q$ for all q with $|q| = p^k$. By construction, w_j is t_j-valid and hence t_{s-1}-valid. Let r be a definitely accepting path of $M_s^{w_j}(u')$. For r we inductively define the set of queries and their dependencies.

$$Q_0 = \{q \mid q \text{ is queried on } r\} \tag{2}$$

$$Q_{n+1} = \bigcup_{\substack{z \in Q_n \text{ with } z = c(i,x,y),\\ i < s,\, x,y \in \mathbb{N},\, t_{s-1}(i) > 0}} \{q \mid q \text{ is queried by } F_i^{w_j}(x)\} \tag{3}$$

Let $Q = \bigcup_{n \geq 0} Q_n$. It holds that $|Q| < 2^{p^k}$, which is seen as follows: For $m_n = \sum_{q \in Q_n} |q|$ we have $m_{n+1} \leq m_n/2$, since the sum of lengths of queries induced by $z = c(i,x,y)$ is at most $|x|^i + i \leq (|x|^i + i)^{2ie} \leq |z|/2$ by the definition of c and $\langle\cdot\rangle$. Thus the m_n form a geometric series. From $|u'| = |u| + 2|v_{p^k}| + 2 \leq 4|v_{p^k}|$ it follows $|Q| \leq 2m_0 \leq 2(|u'|^s + s) \leq 4|u'|^s \leq (16|v_{p^k}|)^s < 2^{p^k}$, where the latter inequality holds by the choice of p in step s.

Let $\bar{q}$ be the smallest word of length p^k that is not in Q. The word exists, since $|Q| < 2^{p^k}$. By the assumption that $|w_j| > q$ for all q with $|q| = p^k$, it holds in particular $|w_j| > \bar{q}$. By the choice of p in step s we have $p > |w_s|$ and hence $|w_{s-1}| < \bar{q} < |w_j|$. Thus for $v = \mathrm{pr}_{\bar{q}}(w_j)$ it holds that $w_{s-1} \sqsubsetneq v \sqsubsetneq w_j$, where w_{s-1} and w_j are t_{s-1}-valid. By Claim 16, v is t_{s-1}-valid. Moreover, $|v| = \bar{q}$, $|\bar{q}| = p^k$, and $p \notin t_{s-1}$, since step s chooses p greater than all elements in t_{s-1}. From Claim 17.4 it follows that $v1$ is t_{s-1}-valid.

We show that there is a t_{s-1}-valid $w' \sqsupseteq v1$ relative to which r is still a definitely accepting path. More precisely, $|w'| = |w_j|$ and for all $q \in Q$ it holds that $q \in w' \Leftrightarrow q \in w_j$. Below we describe how $v1$ is extended bit by bit to w', i.e., how the word $w \sqsupseteq v1 \sqsupseteq w_{s-1}$ constructed so far is extended by one bit b, where z denotes the length of w. We define b and argue that

$$wb \text{ is } t_{s-1}\text{-valid and if } z \in Q \text{ then } b = w_j(z), \tag{4}$$

where we follow the cases in Claim 17.

1. $z = c(i,x,y)$ for $i \in 2\mathbb{N}^+$, $x,y \in \mathbb{N}$, $i \leq s-1$, $t_{s-1}(i) > 0$: If $F_i^w(x) = y$, then $b = 1$ else $b = 0$. Note that $z > \bar{q} > p > t_{s-1}(i)$. By Claim 17.1, wb is t_{s-1}-valid. If $z \in Q$, then by (3), $q \in Q$ for all q queried by $F_i^w(x)$. For these q it holds that $q < z = |w|$ and hence $w(q) = w_j(q)$ by (4). Thus $F_i^w(x) = F_i^{w_j}(x)$. We know that w_j is t_{s-1}-valid and $z > t_{s-1}(i) > 0$. From V2 and V3(a) it follows that $z \in w_j \Leftrightarrow F_i^{w_j}(x) = y \Leftrightarrow F_i^w(x) = y \Leftrightarrow b = 1$. Hence $b = w_j(z)$, which proves (4).
2. $z = c(i,x,y)$ for $i \in 2\mathbb{N}^+$, $x,y \in \mathbb{N}$, $i \leq s-1$, $t_{s-1}(i) = 0$: Let $b = 0$. By Claim 17.2, wb is t_{s-1}-valid. Assume $b \neq w_j(z)$, i.e., $z \in w_j$. We are in the situation that w_j is t_j-valid, $s < j$ is odd, $t_j(s) = p$, $i \in 2\mathbb{N}^+$ with $i < s$, and $t_j(i) = 0$. By V4a, the set $\{c(i,x,y) \in w_j \mid x,y \in \mathbb{N} \text{ and } |c(i,x,y)| \geq p\}$ is empty. However, z belongs to this set, as $z = |w| > |v| = \bar{q}$ and hence $|z| \geq p^k \geq p$. This is a contradiction, which shows (4).
3. $z = c(i,x,y)$ for $i \in 2\mathbb{N}^+$, $x,y \in \mathbb{N}$, $i > s-1$: If $z \notin Q \cap w_j$, then $b = 0$ else $b = 1$. If $b = 0$, then wb is t_{s-1}-valid by Claim 17.3. Otherwise, $b = 1$ and $z \in Q \cap w_j$.
   We show $|x|^i + i < p^k$: Assume $|x|^i + i \geq p^k$. From $p \geq 41$, $e \geq 2$, $k \geq 1$, and $i \geq s \geq 1$ it follows that $(41 \cdot p^{ke})^s < p^{2ike}$. Moreover, $|v_{p^k}| = 2(e + p^{ke} + e + p^k + 3) \leq 10 \cdot p^{ke}$. Hence we obtain

$$|c(i,x,y)| > (|x|^i + i)^{2ie} \geq p^{2ike} > (41 \cdot (p^{ke})^s \geq (40 \cdot p^{ke})^s + s \geq (4|v_{p^k}|)^s + s \geq |u'|^s + s.$$

   Thus $|z| > |u'|^s + s \geq m_0 \geq m_1 \geq \cdots$ and hence $z \notin Q$, a contradiction. This proves $|x|^i + i < p^k$.
   We know that w_j is t_j-valid. By V2, $F_i^{w_j}(x) = y \notin K_\vee^{w_j}$. By $|x|^i + i < p^k$, the computation $F_i^{w_j}(x)$ stops within $|x|^i + i < p^k$ steps. Hence it can only ask queries of length $< p^k$ and $|y| < p^k$. Thus $F_i^w(x) = y \notin K_\vee^w$, since w and w_j coincide with respect to all words of length $< p^k$. By Claim 17.3, wb is t_{s-1}-valid.
   To show the second part of (4) assume $z \in Q$. If $b = 1$, then $z \in Q \cap w_j$ and hence $b = w_j(z)$. If $b = 0$, then $z \notin w_j$ and hence $b = w_j(z)$. This proves (4).
4. $|z| = {p'}^k$ for $p' \in \mathbb{P}^{\geq 41}$, $p' \notin t_s$, $k \geq 1$: Let $b = w_j(z)$. By Claim 17.4, wb is t_{s-1}-valid, which implies (4).
5. Otherwise: Let $b = 0$. By Claim 17.5, wb is t_{s-1}-valid. Assume $b \neq w_j(z)$, i.e., $z \in w_j$. We know that w_j is t_j-valid. From V1 it follows that z must be a word of length ${p'}^k$ for $p' \in \mathbb{P}^{\geq 41}$ and $p' \in t_{s-1}$ (note that the case $p' \notin t_{s-1}$ has already been considered in 4). Choose s' such that $t_{s-1}(s') = p'$ and note that s' is odd. From V4a it follows that $z \notin w_j$, a contradiction which implies (4).

This shows that there exists a t_{s-1}-valid $w' \sqsupseteq v1 \sqsupsetneq w_{s-1}$ such that $|w'| = |w_j| > u'$ and for all $q \in Q$ it holds that $q \in w' \Leftrightarrow q \in w_j$. Hence $M_s^{w'}(u')$ definitely accepts. Moreover, $|v| = \bar{q}$ and hence $\bar{q} \in w'$. From $|\bar{q}| = p^k$ it follows $v_{p^k} \in K^{w'}$ and $u' \in K_\vee^{w'}$. Therefore, step s of the construction defines $t_s = t_{s-1} + 0$ (and chooses for instance $w_s = w'$), which contradicts the assumption $t_s(s) = p \in \mathbb{P}^{\geq 41}$. ◁

▷ Claim 19. $K_\vee^O \cup B$ is $\leq_{\mathrm{m}}^{\mathrm{p},O}$-complete for NP^O for all $B \in \mathrm{NP}^O$ that are disjoint to $K_\vee^O$.

Proof. Choose s odd such that $B = L(M_s^O)$. We claim that $t_s(s) = p \in \mathbb{P}^{\geq 41}$. Otherwise, there exists $x \in K_\vee^{w_s}$ such that $M_s^{w_s}(x)$ definitely accepts. Hence $x \in K_\vee^O$ and $M_s^O(x)$ accepts, which contradicts the assumption $K_\vee^O \cap L(M_s^O) = \emptyset$.

Let $f(\langle u_1, \ldots, u_n \rangle) = \langle u_0, u_1, \ldots, u_n \rangle$, where $u_0 = v_{p^k}$ for the minimal $k \geq 1$ such that $|\langle u_1, \ldots, u_n \rangle| \leq |v_{p^k}|$.

It holds that $f \in \mathrm{FP} \subseteq \mathrm{FP}^O$. We argue that f reduces $K_\vee^O$ to $K_\vee^O \cup B$. If $\langle u_1, \ldots, u_n \rangle \in K_\vee^O$, then $f(\langle u_1, \ldots, u_n \rangle) \in K_\vee^O$.

Assume now $\langle u_1, \dots, u_n \rangle \notin K_\vee^O$. From $t_s(s) = p$ it follows that for all $k \geq 1$, O does not contain elements of length p^k and hence $v_{p^k} \notin K^O$. Therefore, $f(\langle u_1, \dots, u_n \rangle) \notin K_\vee^O$. Moreover, by Claim 18, $f(\langle u_1, \dots, u_n \rangle) \notin L(M_s^O) = B$. ◁

▷ Claim 20. If A is $\leq_{\mathrm{m}}^{\mathrm{p},O}$-complete for NP^O and disjoint to $B \in \mathrm{NP}^O$, then $A \cup B$ is $\leq_{\mathrm{m}}^{\mathrm{p},O}$-complete for NP^O.

Proof. Otherwise, there are counterexamples A and B. Choose $f \in \mathrm{FP}^O$ such that $K_\vee^O \leq_{\mathrm{m}}^{\mathrm{p},O} A$ via f and let $B' = f^{-1}(B)$. Observe $B' \in \mathrm{NP}^O$, $K_\vee^O \cap B' = \emptyset$, and $K_\vee^O \cup B' \leq_{\mathrm{m}}^{\mathrm{p},O} A \cup B$ via f. Hence $K_\vee^O \cup B'$ is not $\leq_{\mathrm{m}}^{\mathrm{p},O}$-complete for NP^O, which contradicts Claim 19. ◁

▷ Claim 21. $\overline{K_\vee^O}$ has P^O-optimal proof systems.

The straightforward proof of this claim is left due to space restrictions. As $\overline{K_\vee^O}$ is $\leq_{\mathrm{m}}^{\mathrm{p},O}$-complete for coNP^O, the first statement of the theorem holds. This finishes the proof of Theorem 12. ◀

Köbler, Messner, and Torán [24] prove the following implications (5) and (6).

$$\mathrm{NEE} \cap \mathrm{TALLY} \subseteq \mathrm{coNEE} \quad \Rightarrow \quad \mathrm{H}_{\mathrm{opps}} \tag{5}$$

$$\mathrm{NEE} \cap \mathrm{TALLY} \subseteq \mathrm{EE} \quad \Rightarrow \quad \exists \text{ P-optimal pps} \tag{6}$$

Relative to the oracle O constructed above, the converses of (5) and (6) fail, i.e., the premises are stronger than the conclusions. This supports the hope that one can weaken the premises in (5) and (6).

4.3 Further Oracles

We briefly discuss two further oracles.

▶ **Theorem 22.** *There exists an oracle O with the following properties.*

1. DisjNP^O *does not have $\leq_{\mathrm{m}}^{\mathrm{pp},O}$-complete pairs (and hence $\neg\mathrm{H}_{\mathrm{opps}}$ relative to O).*
2. *There are disjoint sets A and B that are $\leq_{\mathrm{m}}^{\mathrm{p},O}$-complete for NP^O such that $A \cup B$ is not $\leq_{\mathrm{m}}^{\mathrm{p},O}$-complete for NP^O.*

The construction of this oracle is simpler than the other constructions. In order to achieve statement 1, we proceed similarly as for the oracle in Theorem 11. $\neg\mathrm{H}_{\mathrm{union}}$ can be achieved by a straightforward diagonalization.

The following theorem shows that the implication $\mathrm{H}_{\mathrm{union}} \Rightarrow \mathrm{H}_{\mathrm{cpair}}$ cannot be proven in a relativizable way. Ogiwara and Hemachandra [28] construct an oracle that proves that the converse implication $\mathrm{H}_{\mathrm{cpair}} \Rightarrow \mathrm{H}_{\mathrm{union}}$ cannot be proven relativizably as well.

▶ **Theorem 23.** *There exists an oracle O with the following properties.*

1. DisjNP^O *does not have $\leq_{\mathrm{m}}^{\mathrm{pp},O}$-complete pairs (and hence $\neg\mathrm{H}_{\mathrm{opps}}$ relative to O).*
2. *If A is $\leq_{\mathrm{m}}^{\mathrm{p},O}$-complete for NP^O and disjoint to $B \in \mathrm{NP}^O$, then $A \cup B$ is $\leq_{\mathrm{m}}^{\mathrm{p},O}$-complete for NP^O.*

The construction of this oracle has similarities to the constructions in the Theorems 11 and 12. However, there are less dependencies and thus, the construction is less complicated. Roughly speaking, we achieve $\neg\mathrm{H}_{\mathrm{cpair}}$ in the same way as in Theorem 11 and $\mathrm{H}_{\mathrm{union}}$ can be obtained similarly as in Theorem 12.

Table 1 Summary of oracles and their properties. Each column corresponds to the oracle mentioned in the topmost cell. We say that there exist P-optimal (resp., optimal) pps relative to an oracle, if relative to this oracle, some $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete $A \in \mathrm{coNP}$ has a P-optimal (resp., optimal) proof system (cf. Remark 2). A disjoint NP-pair (A, B) is $\leq^{\mathrm{pp}}_{\mathrm{T}}$-complete, if for every disjoint NP-pair (C, D) and every separator S of (A, B) there exists a separator T of (C, D) such that $T \leq^{\mathrm{p}}_{\mathrm{T}} S$. A disjoint NP-pair (A, B) is $\leq^{\mathrm{pp}}_{\mathrm{T}}$-hard for NP, if for every $C \in \mathrm{NP}$ and every separator S of (A, B) it holds that $C \leq^{\mathrm{p}}_{\mathrm{T}} S$. The double exponential time classes are defined as $\mathrm{EE} = \mathrm{DTIME}(2^{O(2^n)})$ and $\mathrm{NEE} = \mathrm{NTIME}(2^{O(2^n)})$.

	[16, T3.8]	[16, T6.1]	[16, T6.7]	[28, L4.7]	[22, T1]	Thm 11	Thm 12	Thm 22	Thm 23
$\exists$ P-optimal pps	false		false			false	true	false	false
$\exists$ optimal pps / $\mathrm{H}_{\mathrm{opps}}$	false	true	false	true		false	true	false	false
$\mathrm{NPC}^{\mathrm{p}}_{\mathrm{m}}$ closed under disj. union / $\mathrm{H}_{\mathrm{union}}$				false	true		true	false	true
$\exists \leq^{\mathrm{pp}}_{\mathrm{m}}$-complete disjoint NP-pairs / $\mathrm{H}_{\mathrm{cpair}}$	false	true	true	true	true	false	true	false	false
$\exists \leq^{\mathrm{pp}}_{\mathrm{T}}$-complete disjoint NP-pairs	false	true	true	true	true			true	
$\exists$ disj. NP-pairs that are $\leq^{\mathrm{pp}}_{\mathrm{T}}$-hard for NP	false	false	false	true	false				
$\exists$ P-inseparable disjoint NP-pairs	true	true	true	true	false	true		true	true
$\mathrm{P} \neq \mathrm{UP}$					false	false			
$\mathrm{P} \neq \mathrm{NP}$	true	true	true	true	true	true	true	true	true
$\mathrm{UP} \neq \mathrm{NP}$	true	true	true	true	true	true			
$\mathrm{NP} \neq \mathrm{coNP}$	true	true	true	false	true	true	true	true	true
$\mathrm{NP} \cap \mathrm{SPARSE}$ has $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete sets		true	false	true			true		
$\mathrm{E} \neq \mathrm{NE}$	true	true	true			true	true	true	true
$\mathrm{NE} \neq \mathrm{coNE}$	true	false	true	false		true	true	true	true
$\mathrm{NEE} \cap \mathrm{TALLY} \not\subseteq \mathrm{EE}$	true		true			true	true	true	true
$\mathrm{NEE} \cap \mathrm{TALLY} \not\subseteq \mathrm{coNEE}$	true	false	true	false		true	true	true	true

5 Conclusion and Open Questions

The main goal of this paper is to investigate the hypotheses $\mathrm{H}_{\mathrm{union}}$, $\mathrm{H}_{\mathrm{opps}}$, and $\mathrm{H}_{\mathrm{cpair}}$. We have shown that – except for the known implication $\mathrm{H}_{\mathrm{opps}} \Rightarrow \mathrm{H}_{\mathrm{cpair}}$ – each two of these hypotheses are independent under relativizable proofs. But what are the connections between the hypotheses if we consider all three at once? At first glance there are 8 possible situations. As $\mathrm{H}_{\mathrm{opps}}$ implies $\mathrm{H}_{\mathrm{cpair}}$ relative to all oracles, there remain 6 possible situations. Table 1 illustrates that oracles for 4 of the 6 possible situations are known. This leads to the open question: do there also exist oracles for the remaining two situations. More precisely, we ask:

- Does there exist an oracle O_1 with the following properties?
  Relative to O_1, $\neg\mathrm{H}_{\mathrm{opps}} \wedge \mathrm{H}_{\mathrm{union}} \wedge \mathrm{H}_{\mathrm{cpair}}$, i.e., there are no optimal pps, unions of disjoint, $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete NP-sets remain complete, and there are $\leq^{\mathrm{pp}}_{\mathrm{m}}$-complete disjoint NP-pairs.
- Does there exist an oracle O_2 with the following properties?
  Relative to O_2, $\neg\mathrm{H}_{\mathrm{opps}} \wedge \neg\mathrm{H}_{\mathrm{union}} \wedge \mathrm{H}_{\mathrm{cpair}}$, i.e., there is no optimal pps, unions of disjoint $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete NP-sets are not always $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete, and DisjNP has $\leq^{\mathrm{pp}}_{\mathrm{m}}$-complete elements.

Furthermore we receive new insights on problems related to the main topic. On the one hand, we answer an open question by Pudlák [32] who asks for an oracle relative to which neither $\neg\mathrm{H}_{\mathrm{cpair}}$ nor $\neg\mathrm{H}_{\mathrm{opps}}$ implies that UP does not have $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete elements (cf. Theorem 11). On the other hand, we show that the converses of Köbler, Messner, and Torán's [24] implications ($\mathrm{NEE} \cap \mathrm{TALLY} \subseteq \mathrm{coNEE} \Rightarrow \mathrm{H}_{\mathrm{opps}}$) and ($\mathrm{NEE} \cap \mathrm{TALLY} \subseteq \mathrm{EE} \Rightarrow$ there exist P-optimal pps) fail relative to an oracle.

References

1 T. Baker, J. Gill, and R. Solovay. Relativizations of the P=NP problem. *SIAM Journal on Computing*, 4:431–442, 1975.

2 O. Beyersdorff. Representable disjoint NP-pairs. In *Proceedings 24th International Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 3328 of *Lecture Notes in Computer Science*, pages 122–134. Springer, 2004.

3 O. Beyersdorff. Disjoint NP-pairs from propositional proof systems. In *Proceedings of Third International Conference on Theory and Applications of Models of Computation*, volume 3959 of *Lecture Notes in Computer Science*, pages 236–247. Springer, 2006.

4 O. Beyersdorff. Classes of representable disjoint NP-pairs. *Theoretical Computer Science*, 377(1-3):93–109, 2007.

5 O. Beyersdorff. The deduction theorem for strong propositional proof systems. *Theory of Computing Systems*, 47(1):162–178, 2010.

6 S. Cook and R. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44:36–50, 1979.

7 T. Dose. P-optimal proof systems for each set in coNP and no complete problems in NP∩coNP relative to an oracle. *CoRR*, abs/1910.08571, 2019. `arXiv:1910.08571`.

8 T. Dose. P ≠ NP and all sets in NP ∪ coNP have P-optimal proof systems relative to an oracle. *CoRR*, abs/1909.02839, 2019. `arXiv:1909.02839`.

9 T. Dose. An oracle separating conjectures about incompleteness in the finite domain. *Theoret. Comput. Sci.*, 2020. `doi:10.1016/j.tcs.2020.01.003`.

10 T. Dose and C. Glaßer. NP-completeness, proof systems, and disjoint NP-pairs. Technical Report 19-050, Electronic Colloquium on Computational Complexity (ECCC), 2019.

11 S. Even, A. L. Selman, and J. Yacobi. The complexity of promise problems with applications to public-key cryptography. *Information and Control*, 61:159–173, 1984.

12 S. Even and Y. Yacobi. Cryptocomplexity and NP-completeness. In *Proceedings 7th International Colloquium on Automata, Languages and Programming*, volume 85 of *Lecture Notes in Computer Science*, pages 195–207. Springer, 1980.

13 C. Glaßer, J. M. Hitchcock, A. Pavan, and S. Travers. Unions of disjoint NP-complete sets. *ACM Trans. Comput. Theory*, 6(1):3:1–3:10, 2014.

14 C. Glaßer, A. Pavan, A. L. Selman, and S. Sengupta. Properties of NP-complete sets. *SIAM Journal on Computing*, 36(2):516–542, 2006.

15 C. Glaßer, A. L. Selman, and S. Sengupta. Reductions between disjoint NP-pairs. *Information and Computation*, 200:247–267, 2005.

16 C. Glaßer, A. L. Selman, S. Sengupta, and L. Zhang. Disjoint NP-pairs. *SIAM Journal on Computing*, 33(6):1369–1416, 2004.

17 C. Glaßer, A. L. Selman, S. Travers, and K. W. Wagner. The complexity of unions of disjoint sets. *Journal of Computer and System Sciences*, 74(7):1173–1187, 2008.

18 C. Glaßer, A. L. Selman, and L. Zhang. Canonical disjoint NP-pairs of propositional proof systems. *Theoretical Computer Science*, 370:60–73, 2007.

19 C. Glaßer, A. L. Selman, and L. Zhang. The informational content of canonical disjoint NP-pairs. *International Journal of Foundations of Computer Science*, 20(3):501–522, 2009.

20 J. Grollmann and A. L. Selman. Complexity measures for public-key cryptosystems. *SIAM Journal on Computing*, 17(2):309–335, 1988.

21 E. Hemaspaandra, L. A. Hemaspaandra, and H. Hempel. All superlinear inverse schemes are conp-hard. *Theoretical Computer Science*, 345(2-3):345–358, 2005.

22 S. Homer and A. L. Selman. Oracles for structural properties: The isomorphism problem and public-key cryptography. *Journal of Computer and System Sciences*, 44(2):287–301, 1992.

23 Erfan Khaniki. New relations and separations of conjectures about incompleteness in the finite domain. *CoRR*, abs/1904.01362, 2019. `arXiv:1904.01362`.

24 J. Köbler, J. Messner, and J. Torán. Optimal proof systems imply complete sets for promise classes. *Information and Computation*, 184(1):71–92, 2003.

25 J. Krajíček and P. Pudlák. Propositional proof systems, the consistency of first order theories and the complexity of computations. *Journal of Symbolic Logic*, 54:1063–1079, 1989.

26 L. Lovász. On the Shannon capacity of a graph. *IEEE Transactions on Information Theory*, 25(1):1–7, 1979.

27 J. Myhill. Creative sets. *Mathematical Logic Quarterly*, 1(2):97–108, 1955.

28 M. Ogiwara and L. Hemachandra. A complexity theory of feasible closure properties. *Journal of Computer and System Sciences*, 46:295–325, 1993.

29 C. M. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994.

30 P. Pudlák. On the lengths of proofs of consistency. In *Collegium Logicum*, pages 65–86. Springer Vienna, 1996.

31 P. Pudlák. On reducibility and symmetry of disjoint NP pairs. *Theoretical Computer Science*, 295:323–339, 2003.

32 P. Pudlák. Incompleteness in the finite domain. *The Bulletin of Symbolic Logic*, 23(4):405–441, 2017.

33 C. Rackoff. Relativized questions involving probabilistic algorithms. *Journal of the ACM*, 29:261–268, 1982.

34 A. A. Razborov. On provably disjoint NP-pairs. *Electronic Colloquium on Computational Complexity (ECCC)*, 1(6), 1994.

35 H. Rogers Jr. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, New York, 1967.

36 Z. Sadowski. On an optimal propositional proof system and the structure of easy subsets of TAUT. *Theoretical Computer Science*, 288(1):181–193, 2002.

37 A. L. Selman. Natural self-reducible sets. *SIAM Journal on Computing*, 17(5):989–996, 1988.

38 E. Tardos. The gap between monotone and non-monotone circuit complexity is exponential. *Combinatorica*, 8(1):141–142, 1988.

39 S. Travers. *Structural Properties of NP-Hard Sets and Uniform Characterisations of Complexity Classes*. PhD thesis, Julius-Maximilians-Universität Würzburg, 2007.

40 O. V. Verbitskii. Optimal algorithms for coNP-sets and the EXP =? NEXP problem. *Mathematical notes of the Academy of Sciences of the USSR*, 50(2):796–801, August 1991.

String Indexing with Compressed Patterns

Philip Bille
Technical University of Denmark, DTU Compute, Denmark
phbi@dtu.dk

Inge Li Gørtz
Technical University of Denmark, DTU Compute, Denmark
inge@dtu.dk

Teresa Anna Steiner
Technical University of Denmark, DTU Compute, Denmark
terst@dtu.dk

Abstract

Given a string S of length n, the classic string indexing problem is to preprocess S into a compact data structure that supports efficient subsequent pattern queries. In this paper we consider the basic variant where the pattern is given in compressed form and the goal is to achieve query time that is fast in terms of the compressed size of the pattern. This captures the common client-server scenario, where a client submits a query and communicates it in compressed form to a server. Instead of the server decompressing the query before processing it, we consider how to efficiently process the compressed query directly. Our main result is a novel linear space data structure that achieves near-optimal query time for patterns compressed with the classic Lempel-Ziv 1977 (LZ77) compression scheme. Along the way we develop several data structural techniques of independent interest, including a novel data structure that compactly encodes all LZ77 compressed suffixes of a string in linear space and a general decomposition of tries that reduces the search time from logarithmic in the size of the trie to logarithmic in the length of the pattern.

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

Keywords and phrases string indexing, compression, pattern matching

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.10

Related Version `https://arxiv.org/abs/1909.11930`

Funding *Inge Li Gørtz*: Supported by the Danish Research Council (DFF–8021-002498).

1 Introduction

The string indexing problem is to preprocess a string S into a compact data structure that supports efficient subsequent pattern matching queries, that is, given a pattern string P, report all occurrences of P within S. In this paper, we introduce a basic variant of string indexing, called the *string indexing with compressed pattern problem*, where the pattern P is given in compressed form and we want to answer the query without decompressing P. The goal is to obtain a compact structure while achieving fast query times in terms of the compressed size of P.

The string indexing with compressed pattern problem captures the following common client-server scenario: a client submits a query and sends it to a server which processes the query. To minimize communication time and bandwidth the query is sent in compressed form. Naively, the server will then have to decompress the query and then process it. With an efficient solution to the string indexing with compressed pattern problem we can eliminate the overhead decompression and speed up queries by exploiting repetitions in pattern strings.

To the best of our knowledge, no non-trivial solution to the string indexing with compressed pattern problem are known. In contrast, the opposite problem, where the indexed string S is compressed and the pattern P is uncompressed, is well-studied [18, 17, 6, 21, 15, 7, 8, 13, 14, 9, 26, 22, 5, 20, 23, 11, 2, 3] (see also the surveys [26, 24, 25, 12]).

We focus on the classic Lempel-Ziv 1977 (LZ77) [29] compression scheme. Note that since the size of an LZ77 compressed string is a lower bound for many other compression schemes (such as all grammar-based compression schemes) our results can be adapted to such compression schemes by recompressing the pattern string. To state the bounds, let n be the length of S, m be the length of P, and z be the LZ77 compressed length of P. Naively, we can solve the string indexing with compressed pattern problem by using a suffix tree of S as our data structure and answering queries by first decompressing them and then traversing the suffix tree with the uncompressed pattern. This leads to a solution with $O(n)$ space and $O(m + \mathrm{occ})$ query time. At the other extreme, we can store a trie of all the LZ77 compressed suffixes of S together with a simple tabulation, leading to a solution with $O(n^3)$ space and $O(z + \mathrm{occ})$ query time (see discussion in Section 3).

We present the first non-trivial solution to the string indexing with compressed pattern problem achieving the following bound:

▶ **Theorem 1.** *We can solve the string indexing with compressed pattern problem for* LZ77*-compressed patterns in $O(n)$ space and $O(z + \log m + \mathrm{occ})$ time, where n is the length of the indexing string, m is the length of the pattern, and z is the number of phrases in the* LZ77 *compressed pattern.*

Since any solution must use at least $\Omega(z + \mathrm{occ})$ time to read the input and report the occurrences, the time bound in Theorem 1 is optimal within an additive $O(\log m)$ term. In the common case when $z = O(\log m)$ or if we consider LZ77 without *self-references* the time bound is optimal. For simplicity, we focus on reporting queries, but the result is straightforward to extend to also support *existential queries* (decide if the pattern occurs in S) and *counting queries* (count the number of occurrences of the pattern in S) in $O(z + \log m)$ time and the same space.

To achieve Theorem 1 we develop several data structural techniques of independent interest. These include a compact data structure that encodes all LZ77 compressed suffixes of a string in linear space and a general decomposition of tries that reduces the search time from logarithmic in the size of the trie to logarithmic in the length of the pattern.

The paper is organized as follows. In Section 2 we recall basic string data structures and LZ77 compression. In Section 3 we present a simple $O(n^2)$ space and $O(z + \log n + \mathrm{occ})$ time data structure that forms the basis of our solutions in the following sections. In Section 4 we show how to achieve linear space with the same time complexity. Finally, in Section 5 we show how to improve the $\log n$ term to $\log m$.

2 Preliminaries

A string S of length n is a sequence $S[0] \cdots S[n-1]$ of n characters drawn from an alphabet Σ. The string $S[i] \cdots S[j-1]$ denoted $S[i,\ j]$ is called a *substring* of S. The substrings $S[0,\ j]$ and $S[i,\ n]$ are called the j^{th} *prefix* and i^{th} *suffix* of S, respectively. We will sometimes use S_i to denote the i^{th} suffix of S.

Longest Common Prefix

For two strings S and S', the *longest common prefix of* S and S', denoted $\texttt{lcp}(S, S')$, is the maximum $j \in \{0, \ldots, \min(|S|, |S'|)\}$ such that $S[0,\ j] = S'[0,\ j]$.

Given a string S of length n, there is a data structure of size $O(n)$ that answers $\texttt{lcp}$-queries for any two suffixes of S in constant time by storing a suffix tree combined with an efficient nearest common ancestor (NCA) data structure [16, 28].

Compact Trie

A compact trie for a set D of strings $S^1, \ldots, S^l$ is a rooted labeled tree T_D, with the following properties: The label on each edge is a substring of one or more S^i. If the set of strings is prefix free, each root-to-leaf path represents a string in the set (obtained by concatenating the labels on the edges of the path), and for every string there is a leaf corresponding to that string. Common prefixes of two strings share the same path maximally, and all internal vertices have at least two children.

The compact trie has $O(l)$ nodes and edges and a total space complexity of $O\left(\sum_{i=1}^{l} |S^i|\right)$. The position in the trie that corresponds to the maximum longest common prefix of a pattern P of length m and any S^i can be found in $O(m)$ time. For a position p in the tree, which can be either a node or a position within the label of an edge, let $\text{str}(p)$ denote the string obtained by concatenating the labels on the path from the root to p. The locus of a string P in T_D, denoted $\texttt{locus}(P)$, is the deepest position p in the tree such that $\text{str}(p)$ is a prefix of P. A compact trie on the suffixes of a string S is called the *suffix tree* of S and can be stored in linear space [28]. The *suffix array* stores the starting positions of the suffixes in the string in lexicographic order. If at every node in the suffix tree its children are stored in lexicographic order, the order of the suffix array corresponds to the order of the leaves in the suffix tree.

LZ77

Given an input string S of length n, the LZ77 parsing divides S into z substrings $f_1, f_2, \ldots, f_z$, called phrases, in a greedy left-to-right order. The i^{th} phrase f_i, starting at position p_i is either (a) the first occurrence of a character in S or (b) the longest substring that has at least one occurrence starting to the left of p_i. If there are more than one occurrence, we assume that the choice is made in a consistent way. To compress S, we can then replace each phrase f_i of type (b) with a pair (r_i, l_i) such that r_i is the distance from p_i to the start of the previous occurrence, and l_i is the length of the phrase. The occurrence of f_i at position $p_i - r_i$ is called the *source* of the phrase. (This is actually the LZ77-variant of Storer and Szymanski [27]; the original one [29] adds a character to each phrase so that it outputs triples instead of tuples.)

Every LZ77-compressed string is a string over the extended alphabet which consists of all possible LZ77 phrases. For any string T we denote this string by $\text{LZ}(T)$.

3 A Simple Data Structure

In this section we will define a data structure that allows us to solve the string indexing with compressed pattern problem in $O(n^2)$ space and $O(z + \log n + \text{occ})$ time, or $O(n^3)$ space and $O(z + \text{occ})$ time. This data structure forms the basis of our solution.

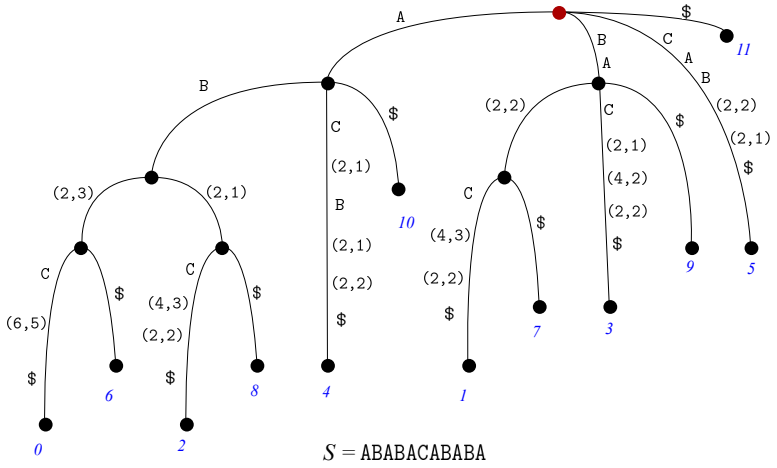


Figure 1 The phrase trie for the string `ABABACABABA$`. In this example, the leaves are sorted according to the lexicographic order of the originial suffixes. For instance the 6^{th} suffix `ABABA$` has the LZ77 parse `A B (2,3) $`, and this string corresponds to the concatenation of labels on the path from the root to the second leaf.

The Phrase Trie

The *phrase trie of a string* S is defined as the compact trie over the set of strings $\{\mathrm{LZ}(S_i\$), i = 0, \ldots, |S| - 1\} \cup \{\$\}$, that is, the LZ77 parses of all suffixes of S appended by a new symbol \$ which is lexicographically greater than any letter in the alphabet. For an example see Figure 1.

The phrase trie for a string S of length n has $n + 1$ leaves, one corresponding to every suffix of $S\$$. Similarly as in the suffix tree, every internal node defines a consecutive range within the suffix array. Since every node has at least 2 children the number of nodes and edges is $O(n)$.

LZ77 has the property that for two strings whose prefixes match up to some position ℓ the LZ77-compression of the two strings will be the same up to (not necessarily including) the phrase that contains position ℓ. As such, we can use the phrase trie to find the suffix S_i of S for which the LZ77-compression of the pattern P agrees with the LZ77-compression of S_i for as long as possible. Assuming they match for $k - 1$ phrases, the longest match of P in S ends within the k^{th} phrase. If we additionally to the phrase trie keep a table for all possible phrases of a pattern that the longest match could end in, encoded as the triple (p, r, l) of starting position, distance to the start of the previous occurance, and length of the phrase, and which for each such phrase stores the solution to the query, we can solve the string indexing with compressed pattern problem in $O(n^3)$ space and $O(z + \mathrm{occ})$ time. Instead, we will store a linear space and constant time `lcp` data structure for S and show that given the first phrase where the suffix S_i and the string P mismatch, we can find the `lcp` of P and S_i by finding the `lcp` of two substrings of S.

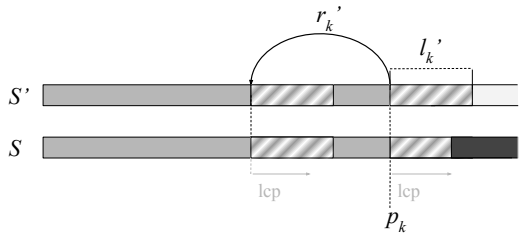


Figure 2 The k^{th} phrase in S' is copied from position $p_k - r'_k$, at which point S and S' are identical; the `lcp` value gives how far p_k and $p_k - r'_k$ match in S.

Longest Common Prefixes in LZ77-Compressed Strings

We will use an intuitive property about LZ77-compressed strings: assuming two strings match up until a certain phrase $k-1$, we can reduce the task of finding the `lcp` of the two strings to the task of finding the longest common prefix between two suffixes of one of the strings. This property is summarized in the following lemma (see also Figure 2):

▶ **Lemma 2.** *Let* $S = f_1\ f_2\ \cdots\ f_z$ *and* $S' = f'_1\ f'_2\ \cdots\ f'_{z'}$ *be two strings parsed into* LZ77 *phrases, where* $f_1 = f'_1$, $f_2 = f'_2, \ldots, f_{k-1} = f'_{k-1}$. *Let* p_k *be the starting position of* f_k *and* f'_k. *If* f'_k *is a phrase represented by a pair* (r'_k, l'_k) *the following holds:*

$$\texttt{lcp}\left(S, S'\right) \geq p_k + \min\left(\texttt{lcp}\left(S\left[p_k,\ n\right], S\left[p_k - r'_k,\ n\right]\right), l'_k\right). \tag{1}$$

Furthermore, if $f_k \neq f'_k$, *equality holds in (1).*

Proof. To prove (1), we will show by induction that for any

$$i \leq \min\left(\texttt{lcp}\left(S\left[p_k,\ n\right], S\left[p_k - r'_k,\ n\right]\right), l'_k\right),$$

we have that $S[p_k + i - 1] = S'[p_k + i - 1]$. For $i = 0$ this is true since S and S' are the same up until position $p_k - 1$. For the induction step assume it is true for all $i_0 < i$. We then have

$$\begin{aligned} S'[p_k + i - 1] &= S'[p_k - r'_k + i - 1] && (2)\\ &= S[p_k - r'_k + i - 1] && (3)\\ &= S[p_k + i - 1], && (4) \end{aligned}$$

where (2) follows from $i \leq l'_k$ and because $p_k - r'_k$ is the source of phrase f'_k, (3) follows from the induction hypothesis, and (4) follows from $i \leq \texttt{lcp}\left(S\left[p_k,\ n\right], S\left[p_k - r'_k,\ n\right]\right)$.

To show equality in the case where $f_k \neq f'_k$, let $t = \min\left(\texttt{lcp}\left(S\left[p_k,\ n\right], S\left[p_k - r'_k,\ n\right]\right), l'_k\right)$. We will show that $S[p_k + t] \neq S'[p_k + t]$. There are two cases:

For $t = \texttt{lcp}\left(S\left[p_k,\ n\right], S\left[p_k - r'_k,\ n\right]\right) < l'_k$, note that $S[p_k - r'_k + t] \neq S[p_k + t]$. From (1) we know that $S'[p_k - r'_k + t] = S[p_k - r'_k + t]$, and therefore we have $S'[p_k + t] = S'[p_k - r'_k + t] = S[p_k - r'_k + t] \neq S[p_k + t]$.

For $l'_k \leq \texttt{lcp}\left(S\left[p_k,\ n\right], S\left[p_k - r'_k,\ n\right]\right)$, note that by (1), we know that S and S' have an `lcp` of length at least $p_k + t$. If $t \geq l_k$, then by the uniqueness of the greedy left-to-right parsing, the k^{th} phrase of S and S' would be the same, contradicting our condition. Otherwise, we have $l_k > t = l'_k$. This together with (1) implies $S[p_k + i] = S[p_k + i - r_k] = S'[p_k + i - r_k]$ for every $i = 0, \ldots, t$, since $r_k \geq 1$. By the greedy parsing property and since $l'_k = t$ we know that $S'[p_k + t - r_k] \neq S'[p_k + t]$ and so $S[p_k + t] \neq S'[p_k + t]$. ◀

3.1 The Data Structure

Additionally to storing the phrase trie of S, we store the suffix array of S, and for every node in the phrase trie, the range of the leaves below it in the suffix array. Finally, we store a linear space and constant time data structure for answering $\texttt{lcp}$-queries for suffixes of S.

3.2 Algorithm

We begin by matching $\mathrm{LZ}(P)$ as far as possible in the phrase trie. Let $v = \texttt{locus}(\mathrm{LZ}(P))$. Let k be the first phrase in $\mathrm{LZ}(P)$ that does not match any of the next phrases in the trie. If v is a node set $w = v$, otherwise let w be the first node below v. We proceed as follows:

- If the k^{th} phrase in P is a single letter, we return p_k as the length of the match and the interval of positions stored at w.
- If the k^{th} phrase is represented by (r_k, l_k) then there are two cases:
  - If v is on an edge, let S_i be the suffix corresponding to any leaf below v. We return
    $$p_k + \min(\texttt{lcp}(S[i + p_k,\ n], S[i + p_k - r_k,\ n]), l_k)$$
    as the length of the match and the interval of positions stored at w.
  - If v is on a node, we do a binary search for the longest match in the range in the suffix array below v. That is, for the suffix S_i corresponding to the middle leaf in the range below v, we compute $\texttt{lcp}(S[i + p_k,\ n], S[i + p_k - r_k,\ n])$. If this is greater than l_k we stop the binary search. Otherwise, we check if the next position in suffix S_i is lexicographically smaller or bigger than the next position in P to see whether we go left or right in the binary search. That is, we compare $S_i[t] = S[i + t]$ with $P[t] = S_i[t - r_k] = S[i + t - r_k]$, and update our search accordingly. We also keep track of the longest match found so far. At the end of the search, we go to longest match, and check left and right in the suffix array to find all occurrences.

3.3 Correctness

The compact trie gives us the longest matching prefix of $\mathrm{LZ}(P) = f_1 \dots f_{z_p}$ in the phrase trie. That is, we find all suffixes $S_i = f'_1 \cdots f'_{z_i}$ for $i = 0, \dots, n-1$ such that $f_1 = f'_1, \dots, f_{k-1} = f'_{k-1}$ and $f_k \neq f'_k$, and k is maximal. By the uniqueness of parsing, the longest prefix of P found in S is the prefix of at least one these suffixes.

Note that by the greedy parsing, the longest match of the k^{th} phrase has to end before the next node in the trie. We argue the different cases:

If the k^{th} phrase in P is a letter, it did not appear in P before. Thus, it never appeared in any of the suffixes we matched so far. Since the next phrase in the phrase trie is different, it is either a copied position, or a different letter. In any case, the next letter of any candidate suffix does not match the next letter in P.

If f_k is represented by (r_k, l_k) there are two subcases. If v is on an edge, recall that S_i is the suffix corresponding to any leaf below the current position v. By Lemma 2 and since $S_i[p] = S[p + i]$ for any p, we have that

$$\begin{aligned}\texttt{lcp}(S_i, P) &= p_k + \min(\texttt{lcp}(S_i[p_k,\ n], S_i[p_k - r_k,\ n]), l_k) \\ &= p_k + \min(\texttt{lcp}(S[i + p_k,\ n], S[i + p_k - r_k,\ n]), l_k).\end{aligned}$$

If v is on a node we have, by the same argument as before,

$$\texttt{lcp}(S_i, P) = p_k + \min(\texttt{lcp}(S[i + p_k,\ n], S[i + p_k - r_k,\ n]), l_k),$$

for every suffix S_i. Further, because of the lexicographic order of the suffix array, we can binary search to find the leaf with the longest match, and by checking the adjacent positions in the suffix array we make sure to find all occurrences.

3.4 Analysis

The suffix array and the `lcp` data structure both use linear space in the size of S. For the phrase trie, we store the LZ77-compressed suffixes of S, which use $O(\sum_{i=1}^{n} z_i) = O(n^2)$ space, where z_i is the number of phrases used to compress suffix S_i.

For the time complexity, we use $O(k) = O(z)$ time for matching the phrases in the trie. In the worst case, that is, when the locus v is on a node, we need $O(\log(\#\texttt{leaves below } v)) = O(\log n)$ constant time `lcp` queries. In total, we have a time complexity of $O(z + \log n + \text{occ})$. In summary, we proved the following lemma.

▶ **Lemma 3.** *The phrase trie solves the string indexing with compressed pattern problem in $O(n^2)$ space and $O(z + \log n + \text{occ})$ time.*

4 Space Efficient Phrase Trie

In this section, we show how to achieve the same functionality as the phrase trie while using linear space. The main idea is to store only one phrase per edge, and use Lemma 2 to navigate along an edge. That is, we no longer store the entire LZ77-compressed suffixes of S.

4.1 The Data Structure

We store a compact form of the phrase trie, which is essentially a blind trie version of the phrase trie. We store the following: We keep the tree structure of the phrase trie, and at each node, we keep a hash table, using perfect hashing [10], where the keys are the first LZ77 phrase of each outgoing edge. For each edge we store as additional information the length of the (uncompressed) substring on that edge and an arbitrarily chosen leaf below it. For an example see Figure 3. As before, we additionally store the suffix array, the range within the suffix array for each node, and a linear-sized `lcp` data structure for S.

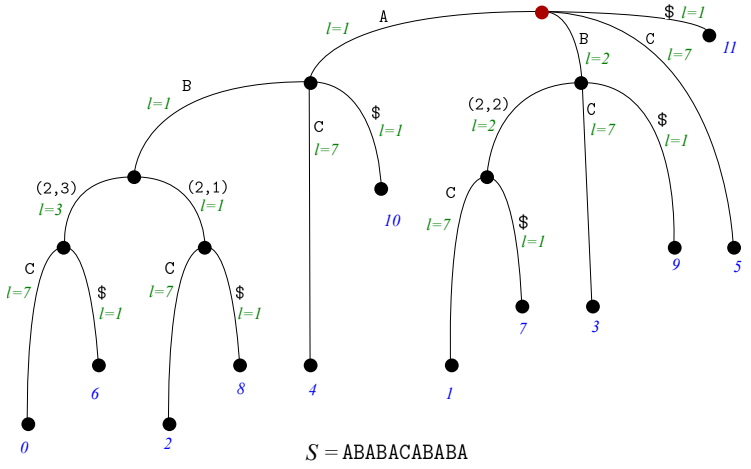


Figure 3 The phrase trie for the string `ABABACABABA` using linear space.

4.2 Algorithm

The algorithm proceeds as follows. We start the search at the root. Assume we have matched $k-1$ phrases of P and the current position in the trie is a node v. To match the next phrase we check if the k^{th} phrase in P is in the hash table of v.

1. If it is not, we proceed exactly as in the previous section in the case where the locus is at a node.
2. If the k^{th} phrase is present, let e be the corresponding edge and let i be the starting index of the leaf stored for e. Set $k = k+1$. We do the following until we reach the end of edge e or get a mismatch. We differentiate between two cases.
   - The k^{th} phrase is a single letter α:
     - If $\alpha = S[i+p_k]$, we set $k = k+1$ and continue with the next phrase.
     - If $\alpha \neq S[i+p_k]$, we stop and return p_k as the length of the match.
   - The k^{th} phrase is represented by (r_k, l_k):
     - If $\min(\texttt{lcp}(S[i+p_k,\ n], S[i+p_k-r_k,\ n]), l_k)) \geq l_k$, we set $k = k+1$ and continue with the next phrase.
     - Otherwise, we return $p_k + \texttt{lcp}(S[i+p_k,\ n], S[i+p_k-r_k,\ n])$ as the length of the match, with the interval of positions stored at the next node.

   If we reach the end of an edge, we go to the next node below and continue in the same way.

Correctness

The correctness follows from the previous section together with Lemma 2, since we always keep the invariant that when we process the k^{th} phrase, we already matched the $k-1$ previous ones.

Analysis

The space complexity is linear since the compact phrase trie has $O(n)$ nodes and edges and stores constant information per node and edge, using perfect hashing.

The time complexity is the same as in the previous section, since for matching full phrases, we use at most one constant time lookup in the hash table and one constant time `lcp` query per phrase in P. As before, the worst case for matching the k^{th} phrase is having to do a binary search, using $O(\log n)$ constant time `lcp` queries. In summary, this gives the following lemma.

▶ **Lemma 4.** *We can solve the string indexing with compressed pattern problem in $O(n)$ space and $O(z + \log n + \mathrm{occ})$ time.*

5 Slice Tree Solution

In this section, we show how to reduce the $O(\log n)$ time overhead to $O(\log m)$. Recall that the additional $O(\log n)$ time originates from the binary search in the case where after matching $k-1$ phrases we arrive at a node, and the k^{th} phrase does not match any of the outgoing edges. In any other case, the solution from the previous section gives $O(z + \mathrm{occ})$ time complexity. We use the solution from the previous section as a basis and show how to speed up the last step of matching the k^{th} phrase. For our solution, we use Karp-Rabin fingerprints and the ART tree decomposition, which we define next.

Karp-Rabin Fingerprints

For a prime p and an $x \leq p$, the *Karp-Rabin fingerprint* [19] of a substring $S[i,\ j]$ is defined as

$$\phi_{p,x}(S[i,\ j]) = \sum_{k=i}^{j-1} S[k]x^{k-i} \mod p.$$

Clearly, we have that if $S[i,\ j] = S'[i',\ j']$, then $\phi_{p,x}(S[i,\ j]) = \phi_{p,x}(S'[i',\ j'])$. Furthermore, the Karp-Rabin fingerprint has the property that for any three strings x, y and z where $z = xy$, given the fingerprint of any two of those strings, the third one can be computed in constant time. It follows that given the fingerprints of all suffixes of a string S, the fingerprint of any substring of S can be computed in constant time.

We assume that p and x are chosen in such a way that $\phi_{p,x}$ is *collision-free* on substrings of S, that is, two distinct substrings of S have different fingerprints. For details on how to construct $\phi_{p,x}$ see for example [4]. We will from now on use the notation $\phi = \phi_{p,x}$.

ART decomposition

The ART decomposition of a tree by Alstrup et al. [1] partitions a tree into a *top tree* and several *bottom trees*. Every vertex v of minimal depth with no more than χ leaves below it is the root of a bottom tree which consists of v and all its descendants. The top tree consists of all vertices that are not in any bottom tree. The following lemma gives a key property of ART trees:

▶ **Lemma 5** (Alstrup et al.[1]). *The ART decomposition with parameter χ for a rooted tree T with n leaves produces a top tree with at most $\frac{n}{\chi+1}$ leaves.*

5.1 The Slice Tree Decomposition

The overall idea is to construct a two level decomposition of the suffix tree. First, we will divide the tree into smaller trees, the *slice trees*, where the heights are powers of two and increase with the depth in the tree. Each of those slice trees is decomposed using an ART decomposition. Together with Karp-Rabin fingerprints stored at the roots of each slice tree, this will allow us to efficiently carry out an approximate search for the longest match, so we can then use the slice trees to find the exact position and length.

In more detail, we store the space efficient phrase trie from the previous section for matching full phrases of the pattern. Additionally, we store the Karp-Rabin fingerprints for each suffix of S, as well as the following *slice tree decomposition* of the suffix tree of S:

- We store the suffix tree together with extra nodes at any position in the suffix tree that corresponds to a string depth that is a power of two. For each node we store the range in the suffix array of the leaves below.
- For each level of string depth 2^i, where $i = 0, \ldots, \lfloor \log n \rfloor$, we store a static hash table with Karp-Rabin fingerprints of the substring in S from the root to every node of string depth 2^i. As in section 4, we use perfect hashing for all hash tables in this solution.
- For each node v at string depth 2^i we define a *slice tree* of order i. The slice tree is the subtree rooted at v, cut off at string depth 2^i, such that the string height of the slice tree is (at most) 2^i.
- We compute an ART decomposition of each slice tree of order i with the parameter χ set to $\chi = 2^i$. For each $1 \leq d < 2^i$, we store a hash table with fingerprints corresponding to the substrings of length d starting at the root of the slice tree and ending in the top tree.

Additionally, for every edge connecting a top tree node to a bottom tree root save the corresponding first letter in the suffix tree. For every leaf in the bottom tree we store the starting position of a leaf below it in the suffix tree.

5.2 Algorithm

To match P, we first match the full phrases in the phrase trie until we find the first phrase f_k which does not match any of the next phrases in the trie. If f_k is just a letter, as before, we are done. Otherwise f_k is represented by (r_k, l_k). Now:

- We find the fingerprint $\phi(P[0,\ p_k]) = \phi(S[i_0,\ i_0 + p_k])$, where i_0 is a leaf below the current position in the phrase trie. Note that since $S[i_0,\ i_0 + p_k]$ is a substring of S and we stored the fingerprints of all suffixes of S we can find its fingerprint in constant time via the fingerprints of the suffixes S_{i_0} and $S_{i_0+p_k}$.
- In order to find the slice tree where the match ends, we do a linear search for the deepest matching fingerprint in the hash tables at the power of 2 levels in the following way:
  - For $j \in \left\{2^{\lceil \log p_k \rceil} - p_k, 2^{\lceil \log p_k \rceil + 1} - p_k, \ldots, 2^{\lfloor \log n \rfloor} - p_k\right\}$ and while $j < l_k$, find the fingerprint of $f_k[0,\ j] = S[i_0 + p_k - r_k,\ i_0 + p_k - r_k + j]$ and look for $\phi(P[0,\ p_k]) + x^{p_k}\phi(f_k[0,\ j]) \mod p$ in the hash table of depth $p_k + j$. If $\phi(P[0,\ p_k]) + x^{p_k}\phi(f_k[0,\ j]) \mod p = \phi(S_i)$ for some i, we check if $\phi(S[i,\ i + p_k]) = \phi(P[0,\ p_k])$ to avoid false positives. We keep doing this until the first level where it is not present or the check fails.
  - For the last level where there is a match, we find the corresponding node the slice tree rooted at that node. Note that this slice tree can be of order at most $\log m$.
- Similarly as the linear search above, we now do an exponential search for fingerprints on the levels in the top tree of the slice tree. For the lowest level in which there is a match in the top tree, find the corresponding position v. If this is an internal node without any off-hanging bottom trees or on an edge in the top tree then $\texttt{locus}(P) = v$. Once we have found $\texttt{locus}(P)$ we can easily find and return the occurrences as before. Otherwise, we check if the next letter in P matches any of the off-hanging bottom trees. Again, we can find this letter in constant time by looking up its source in S. If it matches, we do a binary search for the longest match with the leaves of the bottom tree, which proceeds exactly as in the phrase trie solution, but restricted to the representative leaves stored for each bottom tree leaf. For each bottom tree leaf that has a longest match with P report all suffix tree leaves below it.

5.3 Correctness

The correctness of matching the first $k - 1$ phrases follows from the previous section. Given that k is the first phrase that does not match any of the next phrases in the suffixes, we argue for the linear search in the power of two levels in the suffix tree. We know that the Karp-Rabin fingerprints have no false negatives, so if $P[0,\ p_k + j] = S[i,\ i + p_k + j]$ for some i, then $\phi(P[0,\ p_k]) + x^{p_k}\phi(f_k[0,\ j]) \mod p$ will be present in the hash table of level $p_k + j$. Further, we chose ϕ such that it has no false positives on substrings of S, so by checking $\phi(P[0,\ p_k]) = \phi(S[i,\ i + p_k])$ separately, we make sure that $P[0,\ p_k + j]$ and $S[i,\ i + p_k + j]$ are actually identical. Together, this means that by finding the biggest j such that $p_k + j$ is a power of two and both conditions are fulfilled, we will find the slice tree that contains the end of the longest match.

Next, we argue for the detailed search within the slice tree. The argument for the exponential search is the same as for the linear search. When we end the exponential search, we found the position in the top tree of maximum depth that corresponds to a substring

of S matching a prefix of P. So the longest match either ends there or in a bottom tree that is connected to this position. If there is more than one such bottom tree, the first letter on each edge will uniquely identify the bottom tree that contains the leaf or leaves with the longest match. If the longest match ends in a bottom tree, it is enough to do the binary search with any representative leaf in the suffix tree per leaf in the bottom tree, since for any such leaf the prefix of a given length that ends in the bottom tree is the same.

5.4 Analysis

We use linear space for the phrase trie representation of the previous section and the fingerprints of the suffixes of S. Additionally, we use $O(n \log n)$ space for the extra nodes and hash tables at the power of two levels.

For each slice tree T of order i denote $|T|$ the number of nodes in the slice tree and let $h = 2^i$ be the maximal height of the slice tree. By Lemma 5, the top tree has at most $|T|/h$ leaves. By the definition of the slice tree, each root-to-leaf path has at most h positions. As such, the hash tables for the top tree take up $O(|T|)$ space. Furthermore we use constant space per leaf in the bottom tree. Each bottom tree leaf is a node in the suffix tree or an extra node, and each such node is a leaf in at most one bottom tree. So the total space for all slice trees is $\sum_{T \text{ is slice tree}} O(|T|) = O(\#\texttt{nodes in suffix tree} + \texttt{extra nodes}) = O(n \log n)$.

For the time complexity, as before, we use $O(z)$ for matching in the phrase trie. Since we stored the fingerprints of all suffixes of S, the fingerprint of any substring of S can be found in constant time.

For the linear search of fingerprints in the suffix tree, note that the last phrase of P is at most m long. This means we stop the search after checking at most $\log m$ power of 2 levels, and a check can be done in constant time.

After the linear search we end up in a slice tree of order at most $\log m$, which means $h \leq m$. It follows that the exponential search in the top tree uses time at most $O(\log h) = O(\log m)$. Further, by the definition of the ART decomposition, every bottom tree has no more than $h \leq m$ leaves, and as such the binary search in the bottom tree uses no more than $O(\log m)$ operations.

In total, this gives us a time complexity of $O(z + \log m + \text{occ})$. We arrive at the following result:

▶ **Lemma 6.** *The slice tree solution solves the string indexing with compressed pattern problem in $O(n \log n)$ space and $O(z + \log m + \text{occ})$ time.*

6 Saving Space

For the solution above, we constructed $O(n \log n)$ slice trees. By the way we defined them, note that any internal node in a slice tree has to be an original node from the suffix tree. Since there are only $O(n)$ such nodes, we conclude that many of the slice trees consist of a single edge. We will show that by removing those, we can define a linear space solution that gives the same time complexity as in Lemma 6.

6.1 The Data Structure

We start with the slice tree solution. Call every edge that contains two or more extra nodes a *long edge*. For every long edge, delete every extra node except the first and last, which we call v_{first} and v_{last}. For every deleted node also delete the additional information stored for their slice trees, and their corresponding entries in the power of two hash tables. For each long edge, store at the hash table position of v_{first} additionally the information that it is on a long edge, how long that edge is, and a leaf below it.

6.2 Algorithm

The algorithm proceeds almost as before. The only change is that in the linear search of power of two levels, when we match with a node that is v_{first} of a long edge, jump directly to the last power of two level that is before the end of the edge. If the fingerprint is present, proceed normally, otherwise, the longest match ends on that edge and we do a single `lcp` query between the source of the phrase in S and the stored leaf to find its length.

6.3 Correctness

If we do not encounter any long edges, nothing changes. If a long edge is entirely contained in the match, we will first find v_{first} and then jump directly to the last power of two level on that edge, where we will find v_{last}, and then continue as before. If the longest match ends on a long edge, there are two cases:

1. The longest match ends before v_{first} or after v_{last}: this means that by doing the linear search we find the slice tree that the longest match ends in, thus everything follows as before.
2. The longest match ends between v_{first} and v_{last}: In this case, we will find a matching fingerprint at the level corresponding to v_{first} but no matching fingerprint at the level corresponding to v_{last}, which means we will use `lcp` to find the longest match with a leaf below v_{first}. Since the match ends on that edge, this gives us the correct length and position.

6.4 Analysis

For space complexity, note that we only keep original nodes from the suffix tree, plus at most two extra nodes per edge, so a linear number of nodes in total. Since the space used for the slice trees and power of two hash tables is linear in the number of nodes, the total space consumption is linear. The time complexity does not change. This concludes the proof of Theorem 1.

References

1 Stephen Alstrup, Thore Husfeldt, and Theis Rauhe. Marked ancestor problems. In *Proc. 39th FOCS*, pages 534–543, 1998.

2 Djamal Belazzougui and Gonzalo Navarro. Alphabet-independent compressed text indexing. *ACM Trans. Algorithms*, 10(4):23, 2014.

3 Philip Bille, Mikko Berggren Ettienne, Inge Li Gørtz, and Hjalte Wedel Vildhøj. Time–space trade-offs for lempel–Ziv compressed indexing. *Theoret. Comput. Sci.*, 713:66–77, 2018.

4 Philip Bille, Inge Li Gørtz, Mathias Bæk Tejs Knudsen, Moshe Lewenstein, and Hjalte Wedel Vildhøj. Longest common extensions in sublinear space. In *Proc. 26th CPM*, pages 65–76, 2015.

5 Francisco Claude and Gonzalo Navarro. Improved grammar-based compressed indexes. In *Proc. 19th SPIRE*, pages 180–192, 2012.

6 Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proc. 41st FOCS*, pages 390–398, 2000.

7 Paolo Ferragina and Giovanni Manzini. An experimental study of an opportunistic index. In *Proc. 12th SODA*, pages 269–278, 2001.

8 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005.

9 Paolo Ferragina, Giovanni Manzini, Veli Mäkinen, and Gonzalo Navarro. Compressed representations of sequences and full-text indexes. *ACM Trans. Algorithms*, 3(2):20, 2007.

10 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, 1984.

11 Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J Puglisi. LZ77-based self-indexing with faster pattern matching. In *Proc. 11th LATIN*, pages 731–742, 2014.

12 Travis Gagie and Simon J Puglisi. Searching and indexing genomic databases via kernelization. *Front. Bioeng. Biotechnol.*, 3:12, 2015.

13 Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proc. 14th SODA*, pages 841–850, 2003.

14 Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. When indexing equals compression: Experiments with compressing suffix arrays and applications. In *Proc. 15th SODA*, pages 636–645, 2004.

15 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM J. Comput.*, 35(2):378–407, 2005.

16 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

17 Juha Kärkkäinen and Erkki Sutinen. Lempel-Ziv index for q-grams. *Algorithmica*, 21(1):137–154, 1998.

18 Juha Kärkkäinen and Esko Ukkonen. Lempel-Ziv parsing and sublinear-size index structures for string matching. In *Proc. 3rd WSP*, pages 141–155, 1996.

19 Richard M Karp and Michael O Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev*, 31(2):249–260, 1987.

20 Sebastian Kreft and Gonzalo Navarro. On compressing and indexing repetitive sequences. *Theoret. Comp. Sci.*, 483:115–133, 2013.

21 Veli Mäkinen. Compact suffix array. In *Proc. 11th CPM*, pages 305–319, 2000.

22 Veli Mäkinen, Gonzalo Navarro, Jouni Sirén, and Niko Välimäki. Storage and retrieval of highly repetitive sequence collections. *J. Comput. Bio.*, 17(3):281–308, 2010.

23 Shirou Maruyama, Masaya Nakahara, Naoya Kishiue, and Hiroshi Sakamoto. ESP-index: A compressed index based on edit-sensitive parsing. *J. Discrete Algorithms*, 18:100–112, 2013.

24 Gonzalo Navarro. Indexing highly repetitive collections. In *Proc. 23rd IWOCA*, pages 274–279, 2012.

25 Gonzalo Navarro. *Compact data structures: A practical approach*. Cambridge University Press, 2016.

26 Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 39(1):2, 2007.

27 James A Storer and Thomas G Szymanski. Data compression via textual substitution. *J. ACM*, 29(4):928–951, 1982.

28 Peter Weiner. Linear pattern matching algorithms. In *Proc. 14th FOCS*, pages 1–11, 1973.

29 Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inform. Theory*, 23(3):337–343, 1977.

An FPT Algorithm for Minimum Additive Spanner Problem

Yusuke Kobayashi
Research Institute for Mathematical Sciences, Kyoto University, Japan
yusuke@kurims.kyoto-u.ac.jp

Abstract

For a positive integer t and a graph G, an additive t-spanner of G is a spanning subgraph in which the distance between every pair of vertices is at most the original distance plus t. The Minimum Additive t-Spanner Problem is to find an additive t-spanner with the minimum number of edges in a given graph, which is known to be NP-hard. Since we need to care about global properties of graphs when we deal with additive t-spanners, the Minimum Additive t-Spanner Problem is hard to handle and hence only few results are known for it. In this paper, we study the Minimum Additive t-Spanner Problem from the viewpoint of parameterized complexity. We formulate a parameterized version of the problem in which the number of removed edges is regarded as a parameter, and give a fixed-parameter algorithm for it. We also extend our result to the case with both a multiplicative approximation factor α and an additive approximation parameter β, which we call (α, β)-spanners.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Graph algorithms, Fixed-parameter tractability, Graph spanner

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.11

Related Version https://arxiv.org/abs/1903.01047

Funding *Yusuke Kobayashi*: Supported by JST ACT-I Grant Number JPMJPR17UB, and JSPS KAKENHI Grant Numbers JP16K16010, 16H03118, JP18H05291, and JP19H05485, Japan.

1 Introduction

1.1 Spanners

A *spanner* of a graph G is a spanning subgraph of G that approximately preserves the distance between every pair of vertices in G. Spanners were introduced in [4, 40, 41] in the context of synchronization in networks. Since then, spanners have been studied with applications to several areas such as space efficient routing tables [19, 42], computation of approximate shortest paths [17, 18, 26], distance oracles [6, 45], and so on.

A main topic on spanners is trade-offs between the sparsity (i.e., the number of edges) of a spanner and its quality of approximation of the distance, and there are several ways to measure the approximation quality. In the early studies, the approximation quality of spanners was measured by a multiplicative factor, i.e., the ratio between the distance in the spanner and the original distance. Formally, for a positive integer t and a graph G, a spanning subgraph H of G is said to be a *multiplicative t-spanner* if $\mathrm{dist}_H(u, v) \leq t \cdot \mathrm{dist}_G(u, v)$ holds for any pair of vertices u and v. Here, $\mathrm{dist}_G(u, v)$ (resp. $\mathrm{dist}_H(u, v)$) denotes the distance between u and v in G (resp. in H). Note that we deal with only graphs with unit length edges, and hence the distance is defined as the number of edges on a shortest path. A well-known trade-off between the sparsity and the multiplicative factor is as follows: for any positive integer t and any graph G, there exists a $(2t-1)$-spanner with $O(n^{1+1/t})$ edges [3], where n denotes the number of vertices in G. This bound is conjectured to be tight based on the popular Girth Conjecture of Erdős [30].

Another natural measure of the approximation quality is the difference between the distance in the spanner and the original distance. For a positive integer t and a graph G, a spanning subgraph H of G is said to be an *additive t-spanner* if $\mathrm{dist}_H(u,v) \leq \mathrm{dist}_G(u,v) + t$ holds for any pair of vertices u and v. Since an additive spanner was introduced in [36, 37], trade-offs between the sparsity and the additive term have been actively studied. It is shown in [2, 25] that every graph has an additive 2-spanner with $O(n^{3/2})$ edges. In addition, every graph has an additive 4-spanner with $O(n^{7/5}\mathrm{poly}(\log n))$ edges [14], and every graph has an additive 6-spanner with $O(n^{4/3})$ edges [7]. On the negative side, it is shown in [1] that these bounds cannot be improved to $O(n^{4/3-\epsilon})$ for any $\epsilon > 0$.

As a common generalization of these two concepts, (α, β)-spanners have also been studied in the literature. For $\alpha \geq 1$, $\beta \geq 0$, and a graph G, a spanning subgraph H of G is said to be an *(α, β)-spanner* if $\mathrm{dist}_H(u,v) \leq \alpha \cdot \mathrm{dist}_G(u,v) + \beta$ holds for any pair of vertices u and v. See [9, 27, 32, 43, 44, 46, 48, 49] for other results on trade-offs between the sparsity of a spanner and its approximation quality.

In this paper, we consider a classical but natural and important problem that finds a spanner of minimum size. In particular, we focus on additive t-spanners and consider the following problem for a positive integer t.

Minimum Additive t-Spanner Problem
Instance. A graph $G = (V, E)$.
Question. Find an additive t-spanner $H = (V, E_H)$ of G that minimizes $|E_H|$.

The Minimum Multiplicative t-Spanner Problem and the Minimum (α, β)-Spanner Problem are defined in the same way. Such a problem is sometimes called the Sparsest Spanner Problem.

Although additive t-spanners have attracted attention as described above, there are only few results on the Minimum Additive t-Spanner Problem. For any positive integer t, the Minimum Additive t-Spanner Problem is shown to be NP-hard in [37]. It is shown by Chlamtáč et al. [16] that there is no polynomial-time $2^{(\log^{1-\epsilon} n)/t^3}$-approximation algorithm for any $\epsilon > 0$ under a standard complexity assumption. In [16], an $O(n^{3/5+\epsilon})$-approximation algorithm is proposed for any $\epsilon > 0$ for a more general problem. We can obtain algorithms for some special cases as consequences of known results. Every connected interval graph has an additive 2-spanner that is a spanning tree [35], which implies that the Minimum Additive t-Spanner Problem in interval graphs can be solved in polynomial time for $t \geq 2$. The same result holds for AT-free graphs [35]. It is shown in [15] that every chordal graph has an additive 4-spanner with at most $2n - 2$ edges, which implies that there exists a 2-approximation algorithm for the Minimum Additive 4-Spanner Problem in chordal graphs. To the best of our knowledge, no other results exist for the Minimum Additive t-Spanner Problem, which is in contrast to the fact that the Minimum Multiplicative t-Spanner Problem has been actively studied from the viewpoints of graph classes and approximation algorithms (see Section 1.3).

We make a remark on a difference between multiplicative t-spanners and additive t-spanners. As in [13, 38, 33], multiplicative t-spanners can be characterized as follows: a subgraph $H = (V, E_H)$ of $G = (V, E)$ is a multiplicative t-spanner if and only if $\mathrm{dist}_H(u,v) \leq t$ holds for any $uv \in E \setminus E_H$. This characterization means that if two edges in $E \setminus E_H$ are far from each other, then they do not interfere with each other. Thus, we only need to care about local properties of graphs when we deal with multiplicative t-spanners. In contrast, for additive t-spanners, no such characterization exists, and hence we have to consider global properties of graphs. In this sense, handling the Minimum Additive t-Spanner Problem is much harder than the Minimum Multiplicative t-Spanner Problem, which might be a reason why only few results exist for the Minimum Additive t-Spanner Problem.

1.2 Our Results

In this paper, we consider the Minimum Additive t-Spanner Problem from the viewpoint of fixed-parameter tractability and give the first fixed-parameter algorithm parameterized by the number of deleted edges for it. A parameterized version of the Minimum Multiplicative t-Spanner Problem is studied in [33]. Since an additive (or multiplicative) t-spanner of a connected graph contains $\Omega(|V|)$ edges, the number of edges of a minimum additive (or multiplicative) t-spanner is not an appropriate parameter. Therefore, as in [33], a parameter is defined as the number of edges that are removed to obtain an additive (or multiplicative) t-spanner. This parameterization has a meaning when we remove a small number of edges, and such a situation might appear in a subroutine of other algorithms, e.g., in order to obtain a small additive spanner, we can consider a heuristic algorithm that removes a small number of edges repeatedly. Furthermore, the number of removed edges is a solution size in a certain sense. For these reasons, it will be the most natural parameter when we deal with spanners. Note that the same parameterization is also adopted in [5] for another network design problem. Our problem is formulated as follows.

Parameterized Minimum Additive t-Spanner Problem

Instance. A graph $G = (V, E)$.

Parameter. A positive integer k.

Question. Find an edge set $E' \subseteq E$ with $|E'| \geq k$ such that $H = (V, E \setminus E')$ is an additive t-spanner of G or conclude that such E' does not exist.

Note that if there exists a solution of size at least k, then its subset of size k is also a solution, which means that we can replace the condition $|E'| \geq k$ with $|E'| = k$ in the problem. In this paper, we show that there exists a fixed-parameter algorithm for this problem, where an algorithm is called a *fixed-parameter algorithm* (or an *FPT algorithm*) if its running time is bounded by $f(k)(|V| + |E|)^{O(1)}$ for some computable function f. See [20, 31, 39] for more details.

Formally, our result is stated as follows.

▶ **Theorem 1.** *For a positive integer t, there exists a fixed-parameter algorithm for the Parameterized Minimum Additive t-Spanner Problem that runs in $(t+1)^{O(k^2+tk)}|V||E|$ time. In particular, the running time is $2^{O(k^2)}|V||E|$ if t is fixed.*

This result implies that there exists a fixed-parameter algorithm for the problem even when $t + k$ is the parameter. As described in Section 1.1, the Minimum Additive t-Spanner Problem is a really hard problem and only few results were previously known for it. Therefore, this result may be a starting point for further research on the problem. A technical key ingredient of our algorithm is Lemma 5 that constructs a sequence of edge-disjoint cycles satisfying a certain condition, which is of independent interest.

By using almost the same argument, we can show that a parameterized version of the Minimum (α, β)-Spanner Problem is also fixed-parameter tractable. We define the Parameterized Minimum (α, β)-Spanner Problem in the same way as the Parameterized Minimum Additive t-Spanner Problem.

▶ **Theorem 2.** *For real numbers $\alpha \geq 1$ and $\beta \geq 0$, there exists a fixed-parameter algorithm for the Parameterized Minimum (α, β)-Spanner Problem that runs in $(\alpha + \beta)^{O(k^2+(\alpha+\beta)k)}|V||E|$ time.*

1.3 Related Work: Minimum Multiplicative Spanner Problem

As mentioned in Section 1.1, there are a lot of results on the Minimum Multiplicative t-Spanner Problem, whereas only few results are known for the Minimum Additive t-Spanner Problem.

The Minimum Multiplicative t-Spanner Problem is NP-hard for any $t \geq 2$ in general graphs [11, 40], and there are several results on the problem for some graph classes. It is NP-hard even when the input graph is restricted to be planar [10, 33]. Cai and Keil [13] showed that the Minimum Multiplicative 2-Spanner Problem can be solved in linear time if the maximum degree of the input graph is at most 4, whereas this problem is NP-hard even if the maximum degree is at most 9. Venkatesan et al. [47] revealed the complexity of the Minimum Multiplicative t-Spanner Problem for several graph classes such as chordal graphs, convex bipartite graphs, and split graphs. For the weighted version of the problem in which each edge has a positive integer length, Cai and Corneil [12] showed the NP-hardness of the Minimum Multiplicative t-Spanner Problem for $t > 1$.

Another direction of research is to design approximation algorithms for the Minimum Multiplicative t-Spanner Problem. Kortsarz [34] gave an $O(\log n)$-approximation algorithm for $t = 2$, and Elkin and Peleg [28] gave an $O(n^{2/(t+1)})$-approximation algorithm for $t > 2$. The approximation ratio was improved to $O(n^{1/3} \log n)$ for $t = 3$ by Berman et al. [8] and to $O(n^{1/3}\mathrm{poly}(\log n))$ for $t = 4$ by Dinitz and Zhang [22]. On the negative side, for any $t \geq 2$, it is shown in [29] that no $o(\log n)$-approximation algorithm exists unless $P = NP$. This lower bound was improved to $2^{(\log^{1-\epsilon} n)/k}$ for any $\epsilon > 0$ in [21] under a standard complexity assumption. Dragan et al. [23] gave an EPTAS for the problem in planar graphs. When the input graph is a 4-connected planar triangulation, a PTAS is proposed for the Minimum Multiplicative 2-Spanner Problem in [24].

A parameterized version of the Minimum Multiplicative t-Spanner Problem is introduced in [33], where the parameter is the number of deleted edges, and a fixed-parameter algorithm for it is presented in the same paper.

1.4 Organization

The remainder of this paper is organized as follows. In Section 2, we give some preliminaries. In Section 3, we give an FPT algorithm for the Parameterized Minimum Additive t-Spanner Problem and prove Theorem 1. In Section 4, we extend the argument in Section 3 to the Parameterized Minimum (α, β)-Spanner Problem and prove Theorem 2. Finally, in Section 5, we conclude the paper with a summary.

2 Preliminaries

In this paper, we deal with only undirected graphs with unit length edges. Since we can remove all the parallel edges and self-loops when we consider spanners, we assume that all the graphs in this paper are simple. Let $G = (V, E)$ be a graph. For $u, v \in V$, an edge connecting u and v is denoted by uv. For a subgraph H of G, the set of vertices and the set of edges in H are denoted by $V(H)$ and $E(H)$, respectively. For an edge $e \in E$, let $G - e$ denote the subgraph $G' = (V, E \setminus \{e\})$. We say that an edge set $F \subseteq E$ *contains* a path P if $E(P) \subseteq F$. For a path P and for two vertices $u, v \in V(P)$, let $P[u, v]$ denote the subpath of P between u and v. For $u, v \in V$, let $\mathrm{dist}_G(u, v)$ denote the length of the shortest path between u and v in G. Note that the length of a path is the number of edges in it. If G is clear from the context, $\mathrm{dist}_G(u, v)$ is simply denoted by $\mathrm{dist}(u, v)$. For a positive integer t, a subgraph

$H = (V, E_H)$ of $G = (V, E)$ is said to be an *additive t-spanner* if $\mathrm{dist}_H(u, v) \leq \mathrm{dist}_G(u, v) + t$ or $\mathrm{dist}_G(u, v) = +\infty$ holds for any $u, v \in V$. For real numbers $\alpha \geq 1$ and $\beta \geq 0$, a subgraph $H = (V, E_H)$ of $G = (V, E)$ is said to be an *(α, β)-spanner* if $\mathrm{dist}_H(u, v) \leq \alpha \cdot \mathrm{dist}_G(u, v) + \beta$ or $\mathrm{dist}_G(u, v) = +\infty$ holds for any $u, v \in V$. In what follows, we may assume that the input graph $G = (V, E)$ is connected and $\mathrm{dist}_G(u, v)$ is finite for any $u, v \in V$, since we can deal with each connected component separately. For a positive integer p, let $[p] := \{1, \ldots, p\}$.

3 Proof of Theorem 1

3.1 Outline

In this subsection, we show an outline of our proof of Theorem 1.

Define $F \subseteq E$ as the set of all edges contained in cycles of length at most $t + 2$. In other words, an edge $e = uv \in E$ is in F if and only if $G - e$ contains a u-v path of length at most $t + 1$. By definition, if $H = (V, E \setminus E')$ is an additive t-spanner of G, then $\mathrm{dist}_{G-e}(u, v) \leq \mathrm{dist}_H(u, v) \leq \mathrm{dist}_G(u, v) + t = t + 1$ holds for each $e = uv \in E'$, which implies that $E' \subseteq F$. Thus, if $|F|$ is small, then we can solve the Parameterized Minimum Additive t-Spanner Problem by checking whether $H = (V, E \setminus E')$ is an additive t-spanner of G or not for every subset E' of F with $|E'| = k$.

If $|F|$ is sufficiently large (as a function of t and k), then there exist many cycles of length at most $t+2$. In what follows, we show that if G has many cycles of length at most $t+2$, then there always exists $E' \subseteq E$ with $|E'| = k$ such that $H = (V, E \setminus E')$ is an additive t-spanner of G. To this end, we prove the following statements in Sections 3.2–3.4, respectively.

- If there are many cycles of length at most $t+2$, then we can find either many edge-disjoint cycles of length at most $t + 2$ or a desired set $E' \subseteq E$ (Section 3.2).
- If there are many edge-disjoint cycles of length at most $t + 2$, then we can construct a sequence $(C_1, \ldots, C_p)$ of edge-disjoint cycles with a certain condition (Section 3.3). Roughly speaking, this condition means that if $h < i < j$, then removing edges in $E(C_j)$ does not affect the distance between C_h and C_i.
- If we have a sequence of edge-disjoint cycles with the above condition, then we can construct a desired set $E' \subseteq E$ (Section 3.4).

Finally, in Section 3.5, we put them together and describe our entire algorithm.

3.2 Finding Edge-disjoint Cycles

The objective of this subsection is to show that if there are many cycles of length at most $t + 2$, then we can find either many edge-disjoint cycles of length at most $t + 2$ or a desired set $E' \subseteq E$. We first show the following lemma.

▶ **Lemma 3.** *For positive integers r and ℓ, there exists an integer $f_1(r, \ell) = (r\ell)^{O(\ell)}$ satisfying the following condition. For any pair of distinct vertices $u, v \in V$ in a graph $G = (V, E)$, if there exists a set $\mathcal{P}$ of u-v paths of length at most ℓ with $|\mathcal{P}| \geq f_1(r, \ell)$, then G contains two distinct vertices $u', v' \in V$ and r edge-disjoint u'-v' paths of length at most $\ell - \mathrm{dist}(u, u') - \mathrm{dist}(v, v')$.*

Proof. We show that $f_1(r, \ell) := 2(r\ell^3)^{\ell-1}$ satisfies the condition by induction on ℓ. The claim is obvious when $\ell = 1$, because $|\mathcal{P}| \leq 1$ holds as G is simple and $f_1(r, 1) = 2$. Thus, it suffices to consider the case of $\ell \geq 2$. Let $\mathcal{P}$ be a set of u-v paths of length at most ℓ with $|\mathcal{P}| \geq f_1(r, \ell)$. We consider the following two cases separately.

We first consider the case when $|\{P \in \mathcal{P} \mid e \in E(P)\}| < \frac{f_1(r,\ell)}{r\ell}$ holds for every $e \in E$. In this case, $|\{Q \in \mathcal{P} \mid E(P) \cap E(Q) \neq \emptyset\}| < \frac{f_1(r,\ell)}{r}$ for every $P \in \mathcal{P}$. This shows that we can take r edge-disjoint u-v paths in $\mathcal{P}$ by a greedy algorithm (i.e., repeatedly taking a u-v path P in $\mathcal{P}$ and removing all the paths sharing an edge with P). They form a desired set of paths in which $u' = u$ and $v' = v$.

We next consider the case when there exists an edge $e = xy \in E$ such that $|\{P \in \mathcal{P} \mid e \in E(P)\}| \geq \frac{f_1(r,\ell)}{r\ell} = 2\ell^2(r\ell^3)^{\ell-2}$. Since $\{x, y\} \neq \{u, v\}$, by changing the roles of x and y if necessary, we may assume that $x \notin \{u, v\}$. For $i = 1, \ldots, \ell - 1$, let $\mathcal{P}^i_{ux}$ be the set of all u-x paths of length i and $\mathcal{P}^i_{xv}$ be the set of all x-v paths of length i. Then, since each path $P \in \mathcal{P}$ containing e can be divided into a u-x path and an x-v path, we obtain

$$\sum_{i+j\leq\ell} |\mathcal{P}^i_{ux}| \cdot |\mathcal{P}^j_{xv}| \geq |\{P \in \mathcal{P} \mid e \in E(P)\}| \geq 2\ell^2(r\ell^3)^{\ell-2}.$$

Since the number of pairs (i, j) with $i + j \leq \ell$ is at most $\frac{\ell(\ell-1)}{2} < \frac{\ell^2}{2}$, there exist $i, j \in [\ell - 1]$ with $i + j \leq \ell$ such that

$$|\mathcal{P}^i_{ux}| \cdot |\mathcal{P}^j_{xv}| \geq 2\ell^2(r\ell^3)^{\ell-2} \cdot \frac{2}{\ell^2} \geq 2(r\ell^3)^{i-1} \cdot 2(r\ell^3)^{j-1} \geq f_1(r, i) \cdot f_1(r, j).$$

Then, we have either (i) $|\mathcal{P}^i_{ux}| \geq f_1(r, i)$ and $|\mathcal{P}^j_{xv}| \geq 1$, or (ii) $|\mathcal{P}^j_{xv}| \geq f_1(r, j)$ and $|\mathcal{P}^i_{ux}| \geq 1$. Suppose that (i) holds. By induction hypothesis, $|\mathcal{P}^i_{ux}| \geq f_1(r, i)$ implies that there exist $u', v' \in V$ and r edge-disjoint u'-v' paths of length at most

$$\begin{aligned}
&i - \mathrm{dist}(u, u') - \mathrm{dist}(x, v') \\
&\quad \leq \ell - j - \mathrm{dist}(u, u') - \mathrm{dist}(x, v') && \text{(by } i + j \leq \ell\text{)} \\
&\quad \leq \ell - \mathrm{dist}(x, v) - \mathrm{dist}(u, u') - \mathrm{dist}(x, v') && \text{(by } |\mathcal{P}^j_{xv}| \geq 1\text{)} \\
&\quad \leq \ell - \mathrm{dist}(u, u') - \mathrm{dist}(v, v'). && \text{(by the triangle inequality)}
\end{aligned}$$

Thus, they form a desired set of paths. The same argument can be applied when (ii) holds. ◀

By using this lemma, we obtain the following proposition.

▶ **Proposition 4.** *Let $G = (V, E)$ be a graph and $\mathcal{C}$ be a set of cycles of length at most $t+2$. Let N be a positive integer and f_1 be a function as in Lemma 3. If $|\mathcal{C}| \geq N(t+2)f_1(k+t+1, t+1)$, then we have one of the following.*

- *There exist N edge-disjoint cycles in $\mathcal{C}$.*
- *There exists $E' \subseteq \bigcup_{C\in\mathcal{C}} E(C)$ with $|E'| = k$ such that $H = (V, E \setminus E')$ is an additive t-spanner of G.*

Proof. For each edge $e \in E$, let $\mathcal{C}_e := \{C \in \mathcal{C} \mid e \in E(C)\}$. We first consider the case when $|\mathcal{C}_e| < f_1(k + t + 1, t + 1)$ holds for every $e \in E$. In this case, $|\{C' \in \mathcal{C} \mid E(C) \cap E(C') \neq \emptyset\}| < (t+2)f_1(k+t+1, t+1)$ for every $C \in \mathcal{C}$. This shows that we can take N edge-disjoint cycles in $\mathcal{C}$ by a greedy algorithm (i.e., repeatedly taking a cycle C in $\mathcal{C}$ and removing all the cycles sharing an edge with C), because $|\mathcal{C}| \geq N(t + 2)f_1(k + t + 1, t + 1)$.

We next consider the case when there exists an edge $e = uv \in E$ such that $|\mathcal{C}_e| \geq f_1(k + t + 1, t + 1)$. Since $\mathcal{P} := \{C - e \mid C \in \mathcal{C}_e\}$ consists of u-v paths of length at most $t + 1$, by Lemma 3, G contains two vertices $u', v' \in V$ and a set $\mathcal{P}'$ of $k + t + 1$ edge-disjoint u'-v' paths of length at most $t' := t + 1 - \mathrm{dist}_G(u, u') - \mathrm{dist}_G(v, v')$. Let Q_u and Q_v be a shortest u-u' path and a shortest v-v' path, respectively. Since $|E(Q_u)|+|E(Q_v)|+1 = t+2-t' \leq t+1$, there exists $\mathcal{P}'' \subseteq \mathcal{P}'$ with $|\mathcal{P}''| = |\mathcal{P}'|-(t+1) = k$ such that each path in $\mathcal{P}''$ does not contain

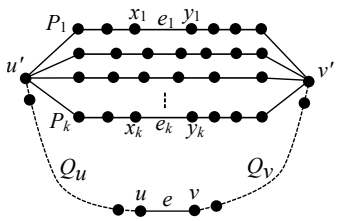


Figure 1 Definition of $e_1, \ldots, e_k$ in Proposition 4.

edges in $E(Q_u) \cup E(Q_v) \cup \{e\}$. Let $P_1, \ldots, P_k$ denote the paths in $\mathcal{P}''$. For $i = 1, \ldots, k$, let e_i be the middle edge of P_i (see Fig. 1). Formally, we take $e_i = x_i y_i$ so that $P_i[u', x_i]$ contains $\lfloor \frac{|E(P_i)|-1}{2} \rfloor \leq \lfloor \frac{t'-1}{2} \rfloor$ edges and $P_i[y_i, v']$ contains $\lceil \frac{|E(P_i)|-1}{2} \rceil \leq \lceil \frac{t'-1}{2} \rceil$ edges. Define $E' := \{e_1, \ldots, e_k\}$ and consider the graph $H = (V, E \setminus E')$. Then, for any $i, j \in [k]$ we can see that

$$\mathrm{dist}_H(x_i, x_j) \leq |E(P_i[u', x_i])| + |E(P_j[u', x_j])| \leq t' \leq t + 1, \tag{1}$$

$$\mathrm{dist}_H(y_i, y_j) \leq |E(P_i[y_i, v'])| + |E(P_j[y_j, v'])| \leq t' \leq t + 1, \tag{2}$$

$$\begin{aligned} \mathrm{dist}_H(x_i, y_j) &\leq |E(P_i[u', x_i])| + |E(Q_u) \cup E(Q_v) \cup \{e\}| + |E(P_j[y_j, v'])| \\ &\leq \left\lfloor \frac{t'-1}{2} \right\rfloor + (t + 2 - t') + \left\lceil \frac{t'-1}{2} \right\rceil \leq t + 1. \end{aligned} \tag{3}$$

We now show that H is an additive t-spanner of G. Let x and y be distinct vertices in V and let P be a shortest x-y path in G. If $E(P) \cap E' = \emptyset$, then it is obvious that $\mathrm{dist}_H(x, y) = \mathrm{dist}_G(x, y)$. If $E(P) \cap E' \neq \emptyset$, then let $P[z, z']$ be the unique minimal subpath of P that contains all edges in $E(P) \cap E'$, where x, z, z', and y appear in this order along P. Since $z, z' \in \{x_1, y_1, \ldots, x_k, y_k\}$, we have $\mathrm{dist}_H(z, z') \leq t + 1$ by (1)–(3). Therefore,

$$\begin{aligned} \mathrm{dist}_H(x, y) &\leq \mathrm{dist}_H(x, z) + \mathrm{dist}_H(z, z') + \mathrm{dist}_H(z', y) && \text{(by the triangle inequality)} \\ &\leq |E(P[x, z])| + t + 1 + |E(P[z', y])| && \text{(by (1)–(3))} \\ &= |E(P)| - |E(P[z, z'])| + t + 1 \\ &\leq \mathrm{dist}_G(x, y) + t, && \text{(by } |E(P)| = \mathrm{dist}_G(x, y)) \end{aligned}$$

which shows that H is an additive t-spanner of G. ◀

3.3 Finding a Good Sequence of Cycles

In this subsection, we construct a sequence of edge-disjoint cycles with a certain condition when we are given many edge-disjoint cycles.

Let $\mathcal{C}$ be a set of edge-disjoint cycles of length at most $t + 2$. For each cycle $C \in \mathcal{C}$, we apply the breadth-first search from $V(C)$ and obtain a shortest path $P(v, C)$ between $V(C)$ and each vertex $v \in V$. That is, $|E(P(v, C))| = \min\{|E(P)| \mid u \in V(C),\ P \text{ is a } u\text{-}v \text{ path}\}$. Then, $\bigcup_{v \in V} E(P(v, C))$ forms a forest for each cycle $C \in \mathcal{C}$. The objective of this subsection is to find a sequence $(C_1, \ldots, C_p)$ of distinct p cycles $C_1, \ldots, C_p \in \mathcal{C}$ satisfying the following condition:

(⋆) For any $h, i, j \in [p]$ with $h < i < j$ and for any vertex $v \in V(C_h)$, it holds that $E(P(v, C_i)) \cap E(C_j) = \emptyset$.

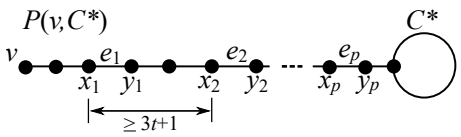


Figure 2 Definition of $e_1, \ldots, e_p$.

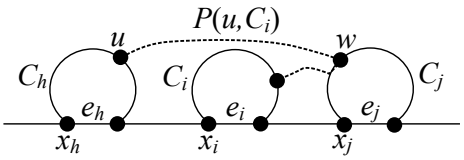


Figure 3 Definition of w.

Roughly speaking, this condition means that if $h < i < j$, then removing edges in $E(C_j)$ does not affect the distance between C_h and C_i.

▶ **Lemma 5.** *For any positive integers t and p, there exists an integer $f_2(t,p) = O(t^2p^4)$ satisfying the following condition. If $\mathcal{C}$ is a set of $f_2(t,p)$ edge-disjoint cycles of length at most $t+2$, then there exists a sequence $(C_1, \ldots, C_p)$ of distinct p cycles $C_1, \ldots, C_p \in \mathcal{C}$ satisfying the condition $(\star)$.*

Proof. We show that $f_2(t,p) := 27(t+2)(3t+1)p^4$ satisfies the condition in the lemma. Let $\mathcal{C}$ be a set of $f_2(t,p)$ edge-disjoint cycles of length at most $t+2$. We consider the following two cases separately.

Case 1. Suppose that there exist a vertex $v \in V$ and a cycle $C^* \in \mathcal{C}$ such that $|E(P(v,C^*)) \cap \bigcup_{C\in\mathcal{C}} E(C)| \geq (3t+1)p$. In this case, we can take edges $e_1, \ldots, e_p$ in $E(P(v,C^*)) \cap \bigcup_{C\in\mathcal{C}} E(C)$ such that $e_1 = x_1y_1, e_2 = x_2y_2, \ldots, e_p = x_py_p$ appear in this order along $P(v,C^*)$ and the subpath of $P(v,C^*)$ between x_i and x_{i+1} contains at least $3t+1$ edges for $i = 1, \ldots, p-1$ (see Fig. 2). For $i = 1, \ldots, p$, let $C_i \in \mathcal{C}$ be the cycle containing e_i. Note that C_i and C_j are distinct if $i \neq j$, since $\mathrm{dist}_G(x_i, x_j) \geq 3t+1 > |E(C_i)|$.

We now show that $(C_1, \ldots, C_p)$ satisfies the condition $(\star)$. Assume to the contrary that there exist indices $h, i, j \in [p]$ with $h < i < j$ and a vertex $u \in V(C_h)$ such that $E(P(u,C_i)) \cap E(C_j) \neq \emptyset$. Let w be the first vertex in $V(C_j)$ when we traverse $P(u,C_i)$ from u to $V(C_i)$ (see Fig. 3). Then, by using

$$\mathrm{dist}(u,x_h) \leq \left\lfloor \frac{|E(C_h)|}{2} \right\rfloor \leq t \quad \text{and} \quad \mathrm{dist}(x_j,w) \leq \left\lfloor \frac{|E(C_j)|}{2} \right\rfloor \leq t, \tag{4}$$

we obtain

$$\begin{aligned}
&\mathrm{dist}(x_h,x_i) + t \\
&\quad \geq \mathrm{dist}(x_h,x_i) + \mathrm{dist}(u,x_h) && \text{(by (4))} \\
&\quad \geq \mathrm{dist}(u,x_i) \geq |E(P(u,C_i))| \geq \mathrm{dist}(u,w) \\
&\quad \geq \mathrm{dist}(x_h,x_j) - \mathrm{dist}(x_h,u) - \mathrm{dist}(w,x_j) && \text{(by the triangle inequality)} \\
&\quad \geq |E(P[x_h,x_j])| - 2t && \text{(by (4) and } \mathrm{dist}(x_h,x_j) = |E(P[x_h,x_j])|) \\
&\quad \geq (|E(P[x_h,x_i])| + 3t + 1) - 2t && \text{(by } |E(P[x_i,x_j])| \geq 3t+1) \\
&\quad = \mathrm{dist}(x_h,x_i) + t + 1, && \text{(by } \mathrm{dist}(x_h,x_i) = |E(P[x_h,x_i])|)
\end{aligned}$$

which is a contradiction. Therefore, $(C_1, \ldots, C_p)$ satisfies the condition $(\star)$.

Case 2. Suppose that $|E(P(v,C^*)) \cap \bigcup_{C\in\mathcal{C}} E(C)| < (3t+1)p$ holds for every $v \in V$ and $C^* \in \mathcal{C}$, which implies that $|\{C \in \mathcal{C} \mid E(P(v,C^*)) \cap E(C) \neq \emptyset\}| < (3t+1)p$ as $\mathcal{C}$ is a set of edge-disjoint cycles. We define $\mathcal{F}_3 \subseteq \mathcal{C}^3$ by

$$\mathcal{F}_3 := \{(C_h,C_i,C_j) \mid C_h,C_i,C_j \in \mathcal{C},\ E(P(v,C_i)) \cap E(C_j) \neq \emptyset \text{ for some } v \in V(C_h)\}.$$

Then, it holds that

$$\begin{aligned}|\mathcal{F}_3| &= \sum_{C_h\in\mathcal{C}}\sum_{C_i\in\mathcal{C}} |\{C_j\in\mathcal{C} \mid E(P(v,C_i))\cap E(C_j)\neq\emptyset \text{ for some } v\in V(C_h)\}| \\ &\leq \sum_{C_h\in\mathcal{C}}\sum_{C_i\in\mathcal{C}}\sum_{v\in V(C_h)} |\{C_j\in\mathcal{C} \mid E(P(v,C_i))\cap E(C_j)\neq\emptyset\}| \\ &< \sum_{C_h\in\mathcal{C}}\sum_{C_i\in\mathcal{C}}\sum_{v\in V(C_h)} (3t+1)p \\ &\leq (t+2)(3t+1)p|\mathcal{C}|^2. \end{aligned} \tag{5}$$

We note that $(C_1,\dots,C_p)$ satisfies the condition $(\star)$ if and only if $(C_h,C_i,C_j)\notin\mathcal{F}_3$ holds for any $h,i,j\in[p]$ with $h<i<j$. That is, $\mathcal{F}_3$ represents the set of forbidden orderings of three cycles. We define $\mathcal{F}_2\subseteq\mathcal{C}^2$ and $\mathcal{F}_1\subseteq\mathcal{C}$ by

$$\begin{aligned}\mathcal{F}_2 &:= \left\{(C_h,C_i)\in\mathcal{C}^2 \;\middle|\; |\{C\in\mathcal{C}\mid (C_h,C_i,C)\in\mathcal{F}_3\}|\geq\frac{|\mathcal{C}|}{3p^2}\right\}, \\ \mathcal{F}_1 &:= \left\{C_h\in\mathcal{C} \;\middle|\; |\{C\in\mathcal{C}\mid (C_h,C)\in\mathcal{F}_2\}|\geq\frac{|\mathcal{C}|}{3p}\right\}.\end{aligned}$$

By (5), we have

$$\begin{aligned}|\mathcal{F}_2| &\leq |\mathcal{F}_3|\cdot\frac{3p^2}{|\mathcal{C}|} < 3(t+2)(3t+1)p^3|\mathcal{C}|, \\ |\mathcal{F}_1| &\leq |\mathcal{F}_2|\cdot\frac{3p}{|\mathcal{C}|} < 9(t+2)(3t+1)p^4 \leq \frac{|\mathcal{C}|}{3}.\end{aligned} \tag{6}$$

In order to obtain $(C_1,\dots,C_p)$ satisfying the condition $(\star)$, we construct a sequence of cycles satisfying additional conditions.

▷ Claim 6. For each $q\in[p]$, there exists a sequence $(C_1,\dots,C_q)$ of q distinct cycles $C_1,\dots,C_q\in\mathcal{C}$ satisfying the following conditions:

- $C_h\notin\mathcal{F}_1$ for any $h\in[q]$,
- $(C_h,C_i)\notin\mathcal{F}_2$ for any $h,i\in[q]$ with $h<i$, and
- $(C_h,C_i,C_j)\notin\mathcal{F}_3$ for any $h,i,j\in[q]$ with $h<i<j$.

Proof. We show the claim by induction on q. When $q=1$, we can choose $C_1\in\mathcal{C}\setminus\mathcal{F}_1$ arbitrarily. Suppose that we have $C_1,\dots,C_q\in\mathcal{C}$ satisfying the conditions in the claim, where $q\leq p-1$. We evaluate the number of cycles that cannot be chosen as C_{q+1}. By the definitions of $\mathcal{F}_1$ and $\mathcal{F}_2$, we have that

$$N_2 := |\{C\in\mathcal{C}\mid (C_h,C)\in\mathcal{F}_2 \text{ for some } h\in[q]\}| \leq q\cdot\frac{|\mathcal{C}|}{3p} \leq \left(1-\frac{1}{p}\right)\cdot\frac{|\mathcal{C}|}{3} < \frac{|\mathcal{C}|}{3}-p, \tag{7}$$

$$N_3 := |\{C\in\mathcal{C}\mid (C_h,C_i,C)\in\mathcal{F}_3 \text{ for some } h,i\in[q] \text{ with } h<i\}| \leq q^2\cdot\frac{|\mathcal{C}|}{3p^2} < \frac{|\mathcal{C}|}{3}, \tag{8}$$

where we use $|\mathcal{C}|>3p^2$ to obtain (7). Since $|\mathcal{C}|-|\mathcal{F}_1|-N_2-N_3>p\geq q+1$ by (6)–(8), there exists a cycle $C_{q+1}\in\mathcal{C}$ that is different from $C_1,\dots,C_q$ such that $(C_1,\dots,C_q,C_{q+1})$ satisfies the conditions in the claim. This shows the claim by induction on q. ◁

By this claim, there exists a sequence $(C_1,\dots,C_p)$ of p distinct cycles $C_1,\dots,C_p\in\mathcal{C}$ such that $(C_h,C_i,C_j)\notin\mathcal{F}_3$ for any $h,i,j\in[p]$ with $h<i<j$, which means that $(C_1,\dots,C_p)$ satisfies the condition $(\star)$. ◀

3.4 Constructing an Additive t-Spanner

In this subsection, we show that we can construct an additive t-spanner of G by using a sequence of edge-disjoint cycles satisfying the condition $(\star)$.

▶ **Lemma 7.** *For any positive integers t and k, there exists an integer $f_3(t,k) = (t+2)^{O(k)}$ satisfying the following condition. If there exists a sequence $(C_1, \ldots, C_p)$ of $p = f_3(t,k)$ edge-disjoint cycles of length at most $t+2$ satisfying the condition $(\star)$, then there exists an edge set $E' \subseteq \bigcup_{i\in[p]} E(C_i)$ with $|E'| = k$ such that $H = (V, E \setminus E')$ is an additive t-spanner of G.*

Proof. We show that $p = f_3(t,k) := k(t+2)^{k-1}$ satisfies the condition. For each edge $e \in E$, define

$$I(e) := \{i \in [p] \mid e \notin \bigcup_{v\in V} E(P(v, C_i))\}.$$

Since $\bigcup_{v\in V} E(P(v,C_i))$ forms a forest for each $i \in [p]$, for any cycle C there exists an edge $e \in E(C)$ such that $i \in I(e)$. In other words, $\bigcup_{e\in E(C)} I(e) = [p]$ for any cycle C. We prove the lemma by showing that Algorithm 1 always finds an edge set $E' \subseteq \bigcup_{i\in[p]} E(C_i)$ with $|E'| = k$ such that $H = (V, E \setminus E')$ is an additive t-spanner of G.

Algorithm 1 Constructing an additive t-spanner from a sequence with $(\star)$.

Input : A sequence $(C_1, \ldots, C_p)$ of edge-disjoint cycles of length at most $t+2$ with the condition $(\star)$

Output : An edge set $E' \subseteq \bigcup_{i\in[p]} E(C_i)$ with $|E'| = k$ such that $H = (V, E \setminus E')$ is an additive t-spanner

```
I_0 := [p]
for i = 1, ..., k do
    Let ind(i) be the minimum index in I_{i-1}
    C'_i := C_{ind(i)}
    Choose an edge e_i ∈ E(C'_i) that maximizes |(I_{i-1} \ {ind(i)}) ∩ I(e_i)|
    I_i := (I_{i-1} \ {ind(i)}) ∩ I(e_i)
end
Return E' := {e_1, ..., e_k}
```

We first show that the algorithm returns a set of k edges. For $i = 1, \ldots, k-1$, since $\bigcup_{e\in E(C'_i)} I(e) = [p]$ and $|E(C'_i)| \le t+2$, we have that $|I_i| \ge \frac{|I_{i-1}\setminus\{\mathrm{ind}(i)\}|}{|E(C'_i)|} \ge \frac{|I_{i-1}|-1}{t+2}$. By combining this with $|I_0| = k(t+2)^{k-1}$, we see that $|I_i| \ge (k-i)(t+2)^{k-i-1}$ for each i by induction, because

$$|I_i| \ge \frac{|I_{i-1}|-1}{t+2} \ge \frac{(k-i+1)(t+2)^{k-i}-1}{t+2} \ge (k-i)(t+2)^{k-i-1}.$$

In particular, $|I_{k-1}| \ge 1$ holds, and hence the algorithm returns a set $E' = \{e_1, \ldots, e_k\}$.

We next show that $H = (V, E \setminus E')$ is an additive t-spanner. Let x and y be distinct vertices in V and let P be a shortest x-y path in G. If $E(P) \cap E' = \emptyset$, then it is obvious that $\mathrm{dist}_H(x,y) = \mathrm{dist}_G(x,y)$. If $E(P) \cap E' = \{e_i\}$ for some $i \in \{1, \ldots, k\}$, then $(E(P) \setminus \{e_i\}) \cup (E(C'_i) \setminus \{e_i\})$ contains an x-y path, and hence we obtain $\mathrm{dist}_H(x,y) \le |(E(P) \setminus \{e_i\}) \cup (E(C'_i) \setminus \{e_i\})| \le \mathrm{dist}_G(x,y) + t$.

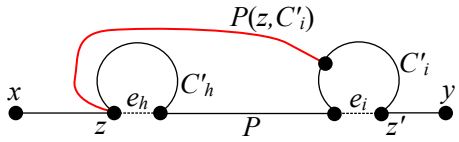


Figure 4 Proof of Lemma 7.

Thus, it suffices to consider the case when $|E(P) \cap E'| \geq 2$. Let $P[z, z']$ be the unique minimal subpath of P that contains all edges in $E(P) \cap E'$, where x, z, z', and y appear in this order along P. Suppose that z and z' are endpoints of edges e_h and e_i in $E(P) \cap E'$, respectively. We may assume that $h < i$ by changing the roles of x and y if necessarily. We now observe the following properties of $P(z, C'_i)$.

- Since $(C_1, \ldots, C_p)$ satisfies $(\star)$, $(C'_1, \ldots, C'_k)$ also satisfies $(\star)$. It follows that $P(z, C'_i)$ does not contain edges in $E(C'_j)$ for any $j > i$, because $z \in V(C'_h)$ and $h < i$. In particular, $P(z, C'_i)$ does not contain e_j for any $j > i$.
- Since $\mathrm{ind}(i) \in I_{i-1} \subseteq I(e_1) \cap I(e_2) \cap \cdots \cap I(e_{i-1})$ by the algorithm, $P(z, C'_i)$ does not contain e_j for any $j < i$.
- It is obvious that $P(z, C'_i)$ does not contain e_i by the definition of $P(z, C'_i)$.

By these observations, $P(z, C'_i)$ does not contain edges in E', which means that $P(z, C'_i)$ is a path in H (see Fig. 4). Since $C'_i - e_i$ contains a path connecting an endpoint of $P(z, C'_i)$ and z', $E(P(z, C'_i)) \cup (E(C'_i) \setminus \{e_i\})$ contains a path between z and z', and hence we have that

$$\mathrm{dist}_H(z, z') \leq |E(P(z, C'_i))| + |E(C'_i) \setminus \{e_i\}| \leq |E(P(z, C'_i))| + t + 1. \tag{9}$$

Since $P[z, z'] - e_i$ forms a path from z to C'_i, we obtain

$$|E(P(z, C'_i))| \leq |E(P[z, z']) \setminus \{e_i\}|. \tag{10}$$

By (9) and (10), we have that

$$\begin{aligned}
&\mathrm{dist}_H(x, y) \\
&\quad \leq \mathrm{dist}_H(x, z) + \mathrm{dist}_H(z, z') + \mathrm{dist}_H(z', y) && \text{(by the triangle inequality)} \\
&\quad \leq \mathrm{dist}_H(x, z) + |E(P(z, C'_i))| + t + 1 + \mathrm{dist}_H(z', y) && \text{(by (9))} \\
&\quad \leq \mathrm{dist}_H(x, z) + |E(P[z, z']) \setminus \{e_i\}| + t + 1 + \mathrm{dist}_H(z', y) && \text{(by (10))} \\
&\quad = |E(P[x, z])| + |E(P[z, z'])| + t + |E(P[z', y])| \\
&\quad = |E(P[x, y])| + t \\
&\quad = \mathrm{dist}_G(x, y) + t.
\end{aligned}$$

Therefore, H is an additive t-spanner of G. ◀

3.5 The Entire Algorithm

In this subsection, we describe our entire algorithm for the Parameterized Minimum Additive t-Spanner Problem and prove Theorem 1 by using Proposition 4 and Lemmas 5 and 7. Define

$$p := f_3(t, k), \qquad N := f_2(t, p), \qquad f_4(t, k) := N(t+2)^2 f_1(k + t + 1, t + 1),$$

where f_1, f_2, and f_3 are as in Lemmas 3, 5, and 7, respectively. Then, $N = (t+2)^{O(k)}$ and $f_1(k+t+1, t+1) = (kt)^{O(t)}$, and hence

$$f_4(t,k) = (t+2)^{O(k)} \cdot (kt)^{O(t)}. \tag{11}$$

When $t \geq k$, (11) is bounded by $(t+2)^{O(t)} \cdot (t^2)^{O(t)} = (t+1)^{O(t)}$. When $t \leq k$, (11) is bounded by $(t+2)^{O(k)} \cdot (k^2)^{O(t)} = (t+1)^{O(k)}$. By combining them, we obtain $f_4(t,k) = (t+1)^{O(k+t)}$. Note that we can simply denote $f_4(t,k) = t^{O(k+t)}$ unless $t = 1$.

In our algorithm, we first compute the set $F \subseteq E$ of all edges contained in cycles of length at most $t+2$. Note that we can do it in $O(|V||E|)$ time by applying the breadth-first search from each vertex.

As described in Section 3.1, if $H = (V, E \setminus E')$ is an additive t-spanner of G for $E' \subseteq E$, then $E' \subseteq F$ holds. Thus, if $|F| \leq f_4(t,k)$, then we can solve the Parameterized Minimum Additive t-Spanner Problem in $O(f_4(t,k)^k|V||E|)$ time by checking whether $H = (V, E \setminus E')$ is an additive t-spanner of G or not for every subset E' of F with $|E'| = k$.

Otherwise, we have $|F| \geq f_4(t,k) = N(t+2)^2 f_1(k+t+1, t+1)$. Since there exist at least $\frac{|F|}{t+2} \geq N(t+2) f_1(k+t+1, t+1)$ cycles of length at most $t+2$ by the definition of F, we can take a set $\mathcal{C}$ of $N(t+2)f_1(k+t+1, t+1)$ cycles of length at most $t+2$ by a greedy algorithm. The procedure is formally described as follows: for $i = 1, 2, \ldots, N(t+2)f_1(k+t+1, t+1)$, we pick up an edge $e_i \in F$, find a cycle C_i of length at most $t+2$ that contains e_i, and remove $E(C_i)$ from F. Then, define $\mathcal{C} := \{C_1, C_2, \ldots, C_{N(t+2)f_1(k+t+1,t+1)}\}$.

By Proposition 4 and Lemmas 5 and 7, there always exists a set $E' \subseteq \bigcup_{C \in \mathcal{C}} E(C)$ with $|E'| = k$ such that $H = (V, E \setminus E')$ is an additive t-spanner of G. Furthermore, such E' can be found in $O(((t+2)|\mathcal{C}|)^k|V||E|) = O(f_4(t,k)^k|V||E|)$ time by checking all the edge sets of size k in $\bigcup_{C \in \mathcal{C}} E(C)$. Note that it will be possible to improve the running time of this part by following the proofs of Proposition 4 and Lemmas 5 and 7. However, we do not do it in this paper, because it does not improve the total running time.

Overall, our algorithm solves the Parameterized Minimum Additive t-Spanner Problem in $O(f_4(t,k)^k|V||E|) = (t+1)^{O(k^2+tk)}|V||E|$ time, and hence we obtain Theorem 1. The entire algorithm is shown in Algorithm 2.

4 Extension to (α, β)-Spanners

In this section, we extend the argument in the previous section to (α, β)-spanners and give a proof of Theorem 2.

Let $t := \lfloor \alpha + \beta \rfloor - 1$. We compute the set $F \subseteq E$ of all edges contained in cycles of length at most $t+2 = \lfloor \alpha + \beta \rfloor + 1$. If $H = (V, E \setminus E')$ is an (α, β)-spanner of G for $E' \subseteq E$, then $\mathrm{dist}_H(u,v) \leq \alpha \cdot \mathrm{dist}_G(u,v) + \beta \leq \alpha + \beta$ for each $uv \in E'$. By integrality, $\mathrm{dist}_H(u,v) \leq \lfloor \alpha + \beta \rfloor$ for each $uv \in E'$, which shows that $E' \subseteq F$ holds. This implies that the problem is trivial if $t = 0$. Thus, we consider the case when $t \geq 1$ and define $f_4(t,k)$ as in Section 3.5. If $|F| \leq f_4(t,k)$, then we can solve the Parameterized Minimum (α, β)-Spanner Problem in $O(f_4(t,k)^k|V||E|)$ time by checking whether $H = (V, E \setminus E')$ is an (α, β)-spanner of G or not for every subset E' of F with $|E'| = k$.

Otherwise, by the argument in Section 3.5, in $O(f_4(t,k)^k|V||E|)$ time, we can find an edge set E' with $|E'| = k$ such that $H = (V, E \setminus E')$ is an additive t-spanner. Then, H is also an (α, β)-spanner, because

$$\begin{aligned} \mathrm{dist}_H(u,v) &\leq \mathrm{dist}_G(u,v) + t \leq (\mathrm{dist}_G(u,v) - 1) + \alpha + \beta \\ &\leq \alpha \cdot (\mathrm{dist}_G(u,v) - 1) + \alpha + \beta = \alpha \cdot \mathrm{dist}_G(u,v) + \beta \end{aligned}$$

for every pair of vertices u and v. Therefore, it suffices to return the obtained set E'. This completes the proof of Theorem 2.

Algorithm 2 Entire Algorithm.

Input : A graph $G = (V, E)$
Output : An edge set $E' \subseteq E$ with $|E'| = k$ such that $H = (V, E \setminus E')$ is an additive t-spanner (or conclude that such E' does not exist)

Compute $F := \{e \in E \mid e \text{ is contained in some cycle of length at most } t + 2\}$
if $|F| \leq f_4(t, k)$ **then**
  for *each* $E' \subseteq F$ *with* $|E'| = k$ **do**
    if $H = (V, E \setminus E')$ *is an additive* t*-spanner of* G **then**
      Return E'
    end
  end
  Conclude that such E' does not exist
end
else
  for $i = 1, 2, \ldots, N(t + 2)f_1(k + t + 1, t + 1)$ **do**
    Find a cycle C_i of length at most $t + 2$ that contains $e_i \in F$
    $F := F \setminus E(C_i)$
  end
  $\mathcal{C} := \{C_1, C_2, \ldots, C_{N(t+2)f_1(k+t+1,t+1)}\}$
  for *each* $E' \subseteq \bigcup_{C \in \mathcal{C}} E(C)$ *with* $|E'| = k$ **do**
    if $H = (V, E \setminus E')$ *is an additive* t*-spanner of* G **then**
      Return E'
    end
  end
end

5 Conclusion

In this paper, we studied the Minimum Additive t-Spanner Problem from the viewpoint of fixed-parameter tractability. We formulated a parameterized version of the Minimum Additive t-Spanner Problem in which the number of removed edges is regarded as a parameter, and gave a fixed-parameter algorithm for it. We also extended our result to the Minimum (α, β)-Spanner Problem.

As described in the last paragraph in Section 1.1, handling the Minimum Additive t-Spanner Problem is much harder than the Minimum Multiplicative t-Spanner Problem, because we have to care about global properties of graphs. Since only few results were previously known for the Minimum Additive t-Spanner Problem, this work may be a starting point for further research on the problem.

References

1 Amir Abboud and Greg Bodwin. The 4/3 additive spanner exponent is tight. *J. ACM*, 64(4):28:1–28:20, 2017. doi:10.1145/3088511.

2 D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999. doi:10.1137/S0097539796303421.

3 Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993. doi:10.1007/BF02189308.

4 Baruch Awerbuch. Complexity of network synchronization. *J. ACM*, 32(4):804–823, 1985. doi:10.1145/4221.4227.

5 Jørgen Bang-Jensen, Manu Basavaraju, Kristine Vitting Klinkby, Pranabendu Misra, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Parameterized algorithms for survivable network design with uniform demands. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*, pages 2838–2850, 2018. doi:10.1137/1.9781611975031.180.

6 Surender Baswana and Telikepalli Kavitha. Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In *Proceedings of the Forty-seventh Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 591–602, 2006. doi:10.1109/FOCS.2006.29.

7 Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. Additive spanners and (α, β)-spanners. *ACM Trans. Algorithms*, 7(1):5:1–5:26, 2010. doi:10.1145/1868237.1868242.

8 Piotr Berman, Arnab Bhattacharyya, Konstantin Makarychev, Sofya Raskhodnikova, and Grigory Yaroslavtsev. Approximation algorithms for spanner problems and directed steiner forest. *Information and Computation*, 222:93–107, 2013. doi:10.1016/j.ic.2012.10.007.

9 Béla Bollobás, Don Coppersmith, and Michael Elkin. Sparse distance preservers and additive spanners. *SIAM J. Discret. Math.*, 19(4):1029–1055, 2005. doi:10.1137/S0895480103431046.

10 Ulrik Brandes and Dagmar Handke. NP-completeness results for minimum planar spanners. In *Proceedings of the 23rd International Workshop Graph-Theoretic Concepts in Computer Science (WG'97)*, pages 85–99, 1997. doi:10.1007/BFb0024490.

11 Leizhen Cai. NP-completeness of minimum spanner problems. *Discrete Applied Mathematics*, 48(2):187–194, 1994. doi:10.1016/0166-218X(94)90073-6.

12 Leizhen Cai and Derek G. Corneil. Tree spanners. *SIAM J. Discrete Math.*, 8:359–387, 1995. doi:10.1137/S0895480192237403.

13 Leizhen Cai and Mark Keil. Spanners in graphs of bounded degree. *Networks*, 24(4):233–249, 1994. doi:10.1002/net.3230240406.

14 Shiri Chechik. New additive spanners. In *Proceedings of the Twenty-fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'13)*, pages 498–512, 2013. doi:10.1137/1.9781611973105.36.

15 Victor D. Chepoi, Feodor F. Dragan, and Chenyu Yan. Additive sparse spanners for graphs with bounded length of largest induced cycle. *Theoretical Computer Science*, 347(1):54–75, 2005. doi:10.1016/j.tcs.2005.05.017.

16 Eden Chlamtáč, Michael Dinitz, Guy Kortsarz, and Bundit Laekhanukit. Approximating spanners and directed steiner forest: Upper and lower bounds. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'17)*, pages 534–553, 2017. doi:10.1137/1.9781611974782.34.

17 Edith Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM Journal on Computing*, 28(1):210–236, 1998. doi:10.1137/S0097539794261295.

18 Edith Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *J. ACM*, 47(1):132–166, 2000. doi:10.1145/331605.331610.

19 Lenore J. Cowen and Christopher G. Wagner. Compact roundtrip routing in directed networks. *Journal of Algorithms*, 50(1):79–95, 2004. doi:10.1016/j.jalgor.2003.08.001.

20 M. Cygan, F.V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

21 Michael Dinitz, Guy Kortsarz, and Ran Raz. Label cover instances with large girth and the hardness of approximating basick-spanner. *ACM Transactions on Algorithms*, 12(2):1–16, 2016. doi:10.1145/2818375.

22 Michael Dinitz and Zeyu Zhang. Approximating low-stretch spanners. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'16)*, pages 821–840, 2016. doi:10.1137/1.9781611974331.ch59.

23 Feodor F. Dragan, Fedor V. Fomin, and Petr A. Golovach. Approximation of minimum weight spanners for sparse graphs. *Theoretical Computer Science*, 412(8):846–852, 2011. doi:10.1016/j.tcs.2010.11.034.

24 William Duckworth, Nicholas C Wormald, and Michele Zito. A PTAS for the sparsest 2-spanner of 4-connected planar triangulations. *Journal of Discrete Algorithms*, 1(1):67–76, 2003. doi:10.1016/S1570-8667(03)00007-8.

25 M. Elkin and D. Peleg. $(1+\epsilon, \beta)$-spanner constructions for general graphs. *SIAM Journal on Computing*, 33(3):608–631, 2004. doi:10.1137/S0097539701393384.

26 Michael Elkin. Computing almost shortest paths. *ACM Trans. Algorithms*, 1(2):283–323, 2005. doi:10.1145/1103963.1103968.

27 Michael Elkin and Ofer Neiman. Efficient algorithms for constructing very sparse spanners and emulators. *ACM Trans. Algorithms*, 15(1):4:1–4:29, 2018. doi:10.1145/3274651.

28 Michael Elkin and David Peleg. Approximating k-spanner problems for $k > 2$. *Theoretical Computer Science*, 337(1):249–277, 2005. doi:10.1016/j.tcs.2004.11.022.

29 Michael Elkin and David Peleg. The hardness of approximating spanner problems. *Theory of Computing Systems*, 41(4):691–729, 2007. doi:10.1007/s00224-006-1266-2.

30 P. Erdős. Extremal problems in graph theory. In *Theory of Graphs and Its Applications*, pages 29–36, 1964.

31 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.

32 Mathias Bæk Tejs Knudsen. Additive spanners: A simple construction. In *Proceedings of the 14th Scandinavian Workshop on Algorithm Theory (SWAT'14)*, pages 277–281, 2014. doi:10.1007/978-3-319-08404-6_24.

33 Yusuke Kobayashi. NP-hardness and fixed-parameter tractability of the minimum spanner problem. *Theoretical Computer Science*, 746:88–97, 2018. doi:10.1016/j.tcs.2018.06.031.

34 Guy Kortsarz and David Peleg. Generating sparse 2-spanners. *Journal of Algorithms*, 17(2):222–236, 1994. doi:10.1006/jagm.1994.1032.

35 Dieter Kratsch, Hoàng-Oanh Le, Haiko Müller, Erich Prisner, and Dorothea Wagner. Additive tree spanners. *SIAM Journal on Discrete Mathematics*, 17(2):332–340, 2003. doi:10.1137/s0895480195295471.

36 Arthur Liestman and Thomas Shermer. Additive spanners for hypercubes. *Parallel Processing Letters*, 1:35–42, September 1991. doi:10.1142/S0129626491000197.

37 Arthur L. Liestman and Thomas C. Shermer. Additive graph spanners. *Networks*, 23:343–363, 1993. doi:10.1002/net.3230230417.

38 M.S. Madanlal, G. Venkatesan, and C. Pandu Rangan. Tree 3-spanners on interval, permutation and regular bipartite graphs. *Information Processing Letters*, 59(2):97–102, 1996. doi:10.1016/0020-0190(96)00078-6.

39 R. Niedermeier. *Invitation to Fixed Parameter Algorithms*. Oxford University Press, 2006.

40 David Peleg and Alejandro A. Schäffer. Graph spanners. *J. Graph Theory*, 13(1):99–116, 1989. doi:10.1002/jgt.3190130114.

41 David Peleg and Jeffrey D. Ullman. An optimal synchronizer for the hypercube. *SIAM J. Computing*, 18(4):740–747, 1989. doi:10.1137/0218050.

42 David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *J. ACM*, 36(3):510–530, 1989. doi:10.1145/65950.65953.

43 Seth Pettie. Low distortion spanners. *ACM Trans. Algorithms*, 6(1):7:1–7:22, 2009. doi:10.1145/1644015.1644022.

44 Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In *Proceedings of the 32nd International Conference on Automata, Languages and Programming (ICALP'05)*, pages 261–272, 2005. doi:10.1007/11523468_22.

45 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005. doi:10.1145/1044731.1044732.

46 Mikkel Thorup and Uri Zwick. Spanners and emulators with sublinear distance errors. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm (SODA'06)*, pages 802–809, 2006. doi:10.1145/1109557.1109645.

47 G. Venkatesan, U. Rotics, M.S. Madanlal, J.A. Makowsky, and C.Pandu Rangan. Restrictions of minimum spanner problems. *Information and Computation*, 136(2):143–164, 1997. doi:10.1006/inco.1997.2641.

48 David P. Woodruff. Lower bounds for additive spanners, emulators, and more. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 389–398, 2006. doi:10.1109/FOCS.2006.45.

49 David P. Woodruff. Additive spanners in nearly quadratic time. In *Proceedings of the 37th International Colloquium Conference on Automata, Languages and Programming (ICALP'10)*, pages 463–474, 2010. doi:10.1007/978-3-642-14165-2_40.

New Bounds for Randomized List Update in the Paid Exchange Model

Susanne Albers
Department of Computer Science, Technical University of Munich, Germany
albers@in.tum.de

Maximilian Janke
Department of Computer Science, Technical University of Munich, Germany
maximilian.janke@in.tum.de

Abstract

We study the fundamental list update problem in the paid exchange model P^d. This cost model was introduced by Manasse, McGeoch and Sleator [18] and Reingold, Westbrook and Sleator [24]. Here the given list of items may only be rearranged using paid exchanges; each swap of two adjacent items in the list incurs a cost of d. Free exchanges of items are not allowed. The model is motivated by the fact that, when executing search operations on a data structure, key comparisons are less expensive than item swaps.

We develop a new randomized online algorithm that achieves an improved competitive ratio against oblivious adversaries. For large d, the competitiveness tends to 2.2442. Technically, the analysis of the algorithm relies on a new approach of partitioning request sequences and charging expected cost. Furthermore, we devise lower bounds on the competitiveness of randomized algorithms against oblivious adversaries. No such lower bounds were known before. Specifically, we prove that no randomized online algorithm can achieve a competitive ratio smaller than 2 in the partial cost model, where an access to the i-th item in the current list incurs a cost of $i - 1$ rather than i. All algorithms proposed in the literature attain their competitiveness in the partial cost model. Furthermore, we show that no randomized online algorithm can achieve a competitive ratio smaller than 1.8654 in the standard full cost model. Again the lower bounds hold for large d.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases self-organizing lists, online algorithm, competitive analysis, lower bound

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.12

Funding Work supported by the European Research Council, Grant Agreement No. 691672, project APEG.

1 Introduction

In this paper we revisit the *list update problem*, one of the most basic problems in the theory of online algorithms [7, 26]. The goal is to maintain an unsorted linear linked list of items so that a sequence of accesses to these items can be served with low total cost. Unsorted linear lists are sensible when one has to store a small dictionary consisting of a few dozen items. Moreover, they have interesting applications in data compression. In fact, the standard compression program `bzip2` relies on a combination of the Burrows-Wheeler transform and linear list encoding [9, 10, 19, 25].

Early work on the list update problem dates back to the 1960s [21]. Over the past decades an extensive body of literature has been developed, see e.g. [1, 2, 5, 6, 7, 8, 11, 12, 14, 24, 26] and references therein. List update in the *standard model* is well understood. In this setting the cost of an access is equal to the depth of the referenced item in the current list. Immediately after an access the requested item may be moved at no extra cost to any position closer to the front of the list (free exchanges). Any other swap of two adjacent items

in the list incurs a cost of 1 and is called a *paid exchange*. During the last years, research on the list update problem has explored (1) alternative cost models [13, 15, 20, 22], (2) refined input models capturing locality of reference [2, 6, 11], and (3) the value of algorithmic service abilities [8, 15, 17].

We investigate the list update problem in the P^d *model*, i.e. the paid exchange model, introduced by Manasse, McGeoch and Sleator [18] as well as Reingold, Westbrook and Sleator [24]. In this model there are no free exchanges and each paid exchange, swapping a pair of adjacent items in the list, incurs a cost of d, where $d \geq 1$ is a real-valued constant. The model is motivated by the fact that the execution time of a program swapping a pair of adjacent items in the list is typically much higher than that of the program doing one iteration of the search loop. Moreover, bringing a referenced element closer to the front of the list does incur cost. As main result we develop nearly tight upper and lower bounds on the competitive ratio achieved by randomized online algorithms.

Problem formulation

In the list update problem we are given an unsorted linear linked list L of n items. An algorithm is presented with a sequence $\sigma = \sigma(1), \ldots, \sigma(m)$ of requests that must be served in the order of occurrence. Each request $\sigma(t)$, $1 \leq t \leq m$, specifies an item in the list. In order to serve a request, an algorithm has to access the requested item, i.e. it has to start at the front of the list and search linearly through the items until the referenced item is found. Hence an access to the i-th item in the list incurs a cost of i. In the *standard model*, immediately after a request, the referenced item may be moved at no extra cost to any position closer to the front of the list. These exchanges are called free exchanges. Moreover, at any time, two adjacent items in the list may be swapped at a cost of 1. Such exchanges are called paid exchanges.

In contrast, in the *paid exchange model* P^d, there are no free exchanges. The list can only be rearranged using paid exchanges. Each paid exchange incurs a cost of d, where $d \geq 1$ is an arbitrary, real-valued constant. Following Reingold, Westbrook and Sleator [24] we assume that the service of a request $\sigma(t)$, $1 \leq t \leq m$, proceeds as follows: First an algorithm performs a number of paid exchanges; then the item referenced by $\sigma(t)$ is accessed. In general, in both the standard and the P^d model, the goal is to serve a request sequence so that the total cost is as small as possible.

An online algorithm has to serve each request without knowledge of any future requests. Given a request sequence σ, let $C_A(\sigma)$ and $C_{\mathrm{OPT}}(\sigma)$ denote the costs incurred by an online algorithm A and an optimal offline algorithm OPT in serving σ. A deterministic online algorithm A is called *c-competitive* if there exists an α such that $C_A(\sigma) \leq c \cdot C_{\mathrm{OPT}}(\sigma) + \alpha$ holds for all request sequences σ. The value α must be independent of the input σ but may depend on the list length n. A randomized online algorithm A is *c-competitive against oblivious adversaries* if there exists an α such that $\mathbf{E}[C_A(\sigma)] \leq c \cdot C_{\mathrm{OPT}}(\sigma) + \alpha$ holds for all σ. Here the expectation is taken over the random choices made by A.

Previous work

Due to the wealth of results on the list update problem, we only mention the most important contributions relevant to our work. First we focus on the standard model. In their seminal paper [26], Sleator and Tarjan showed that the deterministic MOVE-TO-FRONT algorithm is 2-competitive. This is the best competitiveness a deterministic online strategy can attain [16]. Subsequent research developed randomized online algorithms against oblivious adversaries.

Irani [12] gave a SPLIT algorithm that is 1.9375-competitive. Reingold et al. [24] devised a family of COUNTER algorithms and showed that the best of these achieve a competitive ratio of $85/49 \approx 1.7346$. In the same paper a generalized RANDOM RESET algorithm attains a competitive ratio of $\sqrt{3} \approx 1.7321$. A family of TIMESTAMP algorithms was developed in [1]. They attain a competitiveness equal to the Golden Ratio $(1+\sqrt{5})/2 \approx 1.6180$. The best randomized algorithm currently known is 1.6-competitive [3]. The algorithm is a combination of specific instances of the COUNTER and TIMESTAMP families.

As for lower bounds, Teia [27] showed that no randomized online algorithm can be better than 1.5-competitive against oblivious adversaries. Ambühl et al. [5] showed that no projective randomized online algorithm can be better than 1.6-competitive against oblivious adversaries in the partial cost model. In the *partial cost model* an access to the i-th item in the list incurs a cost of $i-1$ rather than i. Moreover, an algorithm is projective if it suffices to consider pairs of items. Specifically, the relative order of any two items in the list only depends on the previous requests to those elements. All the algorithms mentioned above, except for SPLIT, are projective.

Recent research on the standard model has proposed models capturing locality of reference in request sequences [2, 6, 11]. Furthermore, Lopez-Ortiz et al. [17] analyzed the value of paid exchanges. They showed a lower bound of 12/11 on the worst-case ratio between the performance of an offline algorithm that uses only free exchanges and that of an offline algorithm that uses both paid and free exchanges. The optimal offline algorithm does not need to use free exchanges [23]. Boyar et al. [8] analyzed the list update problem with advice, where an online algorithm has partial information on future requests.

We next consider the P^d model. So far only COUNTER and RANDOM RESET algorithms have been studied. The best deterministic online algorithm currently known achieves a competitive ratio of $(5+\sqrt{17})/2 \approx 4.5616$ [4]. No deterministic online algorithm can be better than 3-competitive [24]. Reingold et al. [24] gave a randomized COUNTER algorithm. For $d=1$, the algorithm is 2.75-competitive against oblivious adversaries. For increasing d, the competitiveness decreases and tends to $(5+\sqrt{17})/4 \approx 2.2808$. For small values of d, their RANDOM RESET algorithm achieves competitive ratios that are slightly smaller than those of the COUNTER algorithm. No lower bounds on the performance of randomized online algorithms against oblivious adversaries are known.

As for other cost models, Munro [22] and Kamali et al. [13] proposed settings that allow rearranging large parts of a list at lower costs. Specifically, after an access to the i-th item in the list, all items up to position i can be rearranged at a cost proportional to i. In an even stronger model of Kamali et al. [13], which is interesting in data compression applications, the whole list can be rearranged free of charge, while accessing an item in the ith position only incurs cost $\Theta(\lg i)$.

Our contribution

We present a comprehensive study of the list update problem in the P^d model. First in Section 3 we develop a new randomized online algorithm TIMESTAMP(l,p), which is defined for any positive integer l and any probability p, $0 \le p \le 1$. The strategy incorporates features of the TIMESTAMP algorithm in the standard model and the COUNTER algorithm in the P^d model. In the P^d model one cannot afford to move a referenced item closer to the front of the list on every request to that item. Therefore, for every item in the list, TIMESTAMP(l,p) maintains a mod l counter. These counters are initialized independently and uniformly at random. When an item is requested, its counter value is decremented by 1. If the counter switches from 0 to $l-1$, we say that a *master request* to that item occurs. On a master

request to x, with probability p, item x is moved to the front of the list. Otherwise, with probability $1-p$, item x is inserted in front of the first item y in the list such that (a) at most one master request to y occurred since the last master request to x and (b) the possible master request to y was also handled according to this latter policy, i.e. y was not moved to the front.

TIMESTAMP(l,p) achieves an improved competitive ratio of $c < 2.2442$ against oblivious adversaries, as d grows. A main contribution of this paper is a new analysis technique. TIMESTAMP(l,p) is projective so that it can be analyzed on pairs of items. However in the P^d model, in contrast to the standard model, it is hard to keep track of the optimal offline algorithm. Specifically, it does not hold true anymore that after two consecutive requests to the same item, that item precedes the other item in the optimum list. Therefore we define a more general phase partitioning of request sequences. A phase ends whenever the optimum offline algorithm OPT changes the relative order of the two considered items in the list. Hence the analysis is guided by OPT's moves. In particular, each phase can be assigned the cost of one paid exchange performed by OPT. The challenge is to estimate the cost of TIMESTAMP(l,p) without making any assumptions on the request pattern at the end of a phase. In order to analyze any phase, we also devise a new framework to charge expected service cost.

In Section 4 we develop lower bounds. We prove that, against oblivious adversaries and as d goes to infinity, no randomized online algorithm can achieve a competitive ratio smaller than 1.8654. Furthermore, we show that no randomized online algorithm can attain a competitiveness smaller than $2-1/(2d)$ against oblivious adversaries in the partial cost model, for general d. No lower bound against oblivious adversaries was known before.

In order to establish our lower bounds, we devise a probability distribution on request sequences: The items of a given list are requested in cyclic order. The number of consecutive requests to the same item is distributed geometrically with mean $2d$. We then compare expected online and offline cost. The analysis of the expected cost incurred by OPT is quite involved. In the partial cost model we partition a request sequence into phases, each ending with a certain surplus of requests to the same item, so that OPT will move that item to the front of its list. As for the lower bound in the full cost model, we prove that we may restrict ourselves to the partial cost model and request sequences referencing two items, provided that we consider projective offline algorithms. Unfortunately, OPT is not projective. Therefore, we define a family of projective offline algorithms and analyze their cost using a Markov chain.

Although the competitive ratio of $c < 2.2442$ achieved by TIMESTAMP(l,p) is a relatively small improvement over the previous best bound of 2.2808, our work – together with the lower bounds – significantly tightens the gap on the best competitiveness of randomized online algorithms in the P^d model.

2 Preliminaries

Given any algorithm A for list update in the P^d model, let $C_A(\sigma)$ denote its costs incurred in serving a request sequence σ. We will consider both the *full cost model* and the *partial cost model*. Again, in the former model, an access to the i-th item in the current list incurs a cost of i. In the latter one the access cost is $i-1$ only. Observe that for every algorithm the total full cost exceeds the total partial cost by precisely $m = |\sigma|$, the length of σ. Hence, if an online algorithm is c-competitive in the partial cost model, it is also c-competitive in the full cost model, too. Therefore, we will analyze TIMESTAMP(l,p) in the partial cost model.

We will prove in Section 3 that TIMESTAMP(l,p) is projective so that it can be analyzed using item pairs. An algorithm is projective if, for any request sequences σ and any pair x, y of items, the relative order of x and y in the list is always the same as if only references to x and y were served on the respective two-item list. Formally consider an algorithm A, a list L and two distinct items $x, y \in L$. Let σ be an arbitrary request sequence. Starting from an initial list configuration $L(0)$, let $L_A(\sigma)$ be the list state immediately after A has served σ. Let $L_A(\sigma)_{xy}$ be the list obtained from $L_A(\sigma)$ by deleting all items other than x and y. Next consider the projected request sequence σ_{xy}, obtained from σ by deleting all requests that are neither to x nor to y. Moreover, let $L(0)_{xy}$ be the list derived from $L(0)$ by removing all items, except for x and y. Finally, starting with $L(0)_{xy}$, let $L_A(\sigma_{xy})$ be the list immediately after A has served σ_{xy}. If A is a randomized algorithm, its random choices on σ_{xy} are identical to those on the requests to x and y in σ.

▶ **Definition 1.** *An algorithm A is projective if and only if, for any request sequence σ, starting list L and any pair $x, y \in L$ of distinct items, $L_A(\sigma)_{xy} = L_A(\sigma_{xy})$.*

This property is desirable since it reduces the analysis to two-item lists. Formally, the following holds:

▶ **Proposition 2.** *Consider the partial cost model. A projective online algorithm A is c-competitive if and only if it is c-competitive on request sequences referencing only two items, which are served on a two-item list.*

Since this proposition is well known in the literature, we give the proof in the full version of the paper. We call an algorithm A *strictly c-competitive on σ* if we have $C_A(\sigma) \leq c \cdot C_{\mathrm{OPT}}(\sigma)$. We will also apply this notion to subsequences $\lambda = \sigma(t) \dots \sigma(t')$ of σ. It is an obvious but very useful fact that A is strictly c-competitive on a sequence σ if we can divide σ into subsequences $\sigma = \lambda_1 \dots \lambda_h$ such that A is strictly c-competitive for each λ_i, $1 \leq i \leq h$. Here we assume that OPT serves the entire sequence σ and evaluate OPT's cost on each subsequence $\lambda_1, \dots, \lambda_h$.

3 The TIMESTAMP(l,p)-algorithm

3.1 The algorithm

Our new algorithm TIMESTAMP(l,p), we refer to it by TS(l,p) or TS for short, is the generalization of TIMESTAMP(p) for the standard cost model [1]. In the P^d model item exchanges are expensive and one can afford them only once in a while. Therefore, for every item x in the given list L, our algorithm maintains a mod l counter $c(x)$, taking values in $\{0, \dots, l-1\}$, for some positive integer l. The counter is initialized uniformly at random and independently of other items.

Consider a request $\sigma(t)$, referencing item x. There are two cases. If $c(x) > 0$ before the request, then $c(x)$ is decremented by 1 and the position of x remains unchanged in the current list. On the other hand, if $c(x) = 0$, then a *master request* occurs. TIMESTAMP(l,p) resets $c(x)$ to $l-1$ and moves x closer to the front of the list, choosing among two policies. With probability p, item x is simply moved to the front of the list (Policy 1). With probability $1-p$, item x is moved more reluctantly (Policy 2). Specifically, the algorithm determines the longest suffix $\lambda(t)$ of the request sequence ending with $\sigma(t)$ such that $\lambda(t)$ contains exactly one master request to x, namely the one at $\sigma(t)$. The algorithm then identifies the first item z in the current list such that at most one master request to z occurs in $\lambda(t)$ and additionally the possible master request, if existent, was served using Policy 2. The algorithm

TIMESTAMP(l,p) moves x in front of z in the list. Observe that x satisfies the conditions formulated for item z. If $z = x$ the item is not moved. In particular, x does never move backward in the list, which is sensible. The intuition of Policy 2 is to pass items whose request frequency, measured in terms of master requests, is not higher than that of x. A pseudo-code description of TIMESTAMP(l,p) is given in Algorithm 1.

Note that, for any item x, two master requests are separated by $l-1$ regular requests to x so that the item is not moved too often. Since $c(x)$ is initialized uniformly at random, the cycles consisting of a master request followed by $l-1$ regular requests to x are shifted in a random fashion in σ. In particular, with probability $1/l$ a request to x happens to be a master request.

Algorithm 1 TIMESTAMP(l,p).

Let $\sigma(t) = x$;
if $c(x) > 0$ **then**
    $c(x) \leftarrow c(x) - 1$;
else // $\sigma(t)$ is a master request
    $c(x) \leftarrow l - 1$;
    With probability p, serve $\sigma(t)$ using **Policy 1** and with probability $1-p$ serve it using **Policy 2**;
    Policy 1: Move x to the front of the list.
    Policy 2: Let $\lambda(t)$ be the longest suffix of the sequence ending with $\sigma(t)$ in which exactly one master request to x occurs. Let z be the first item in the current list for which at most one master request occurs in $\lambda(t)$ and the possible master request was served using Policy 2. If $z \neq x$ move x in front of z in the list.

Theorem 3 gives the competitive ratio of TS(l,p), which is the maximum of six expressions. Nonetheless, the maximum can be determined exactly and truly optimal, algebraic values for p, l and hence the competitive ratio c can be computed. Details are given in the full paper. By plugging in $p = 0.45787$ and $\varphi = l/d = 1.19390$, the reader can verify that indeed $c < 2.2442$. For the optimal choice of p and φ, we have $c_1 = c_2 = c_3$ while the other ratios are smaller.

▶ **Theorem 3.** *Let $\varphi = \frac{l}{d}$. TS(l,p) is c-competitive, where c is the maximum of the following expressions:*

$$c_1 = 1 + \left(\tfrac{1}{2} + \max\{1, 2p\}(1-p)\right)\varphi \quad c_2 = \tfrac{7-3p}{4} + \tfrac{1}{\varphi} \quad c_3 = 1 + \tfrac{3p}{2} - p^2 + \tfrac{2p}{\varphi}$$
$$c_4 = \tfrac{3-p+p^2}{2} + \tfrac{2(1-2p+2p^2)}{\varphi} \quad c_5 = \tfrac{3+p-p^2}{2} + \tfrac{2p^2}{\varphi} \quad c_6 = 2 - p + \tfrac{1-p}{\varphi}.$$

As d goes to infinity, $c < 2.2442$, when choosing $p \approx 0.45787$ and l such that $\varphi \approx 1.19390$.

Figure 1 depicts the max-function we wish to minimize, considering two ranges for p and φ each. A thorough analysis shows indeed that it is identical to the function $\max\{c_1, \ldots, c_4\}$.

3.2 The analysis

We will prove that TS(l,p) is c-competitive in the partial cost model, for the ratio c stated in Theorem 3. As explained in Section 2 this immediately implies c-competitiveness in the full cost model. In [1] it is shown that TS(l,p) is projective for $l = 1$. This is easily generalized to the case of arbitrary l. A proof of the following proposition is contained in the full paper.

▶ **Proposition 4.** *The algorithm TS(l,p) is projective.*

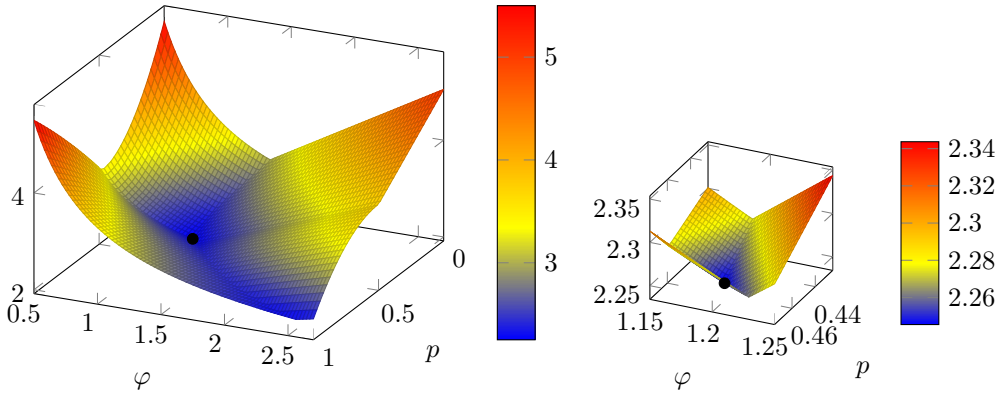


Figure 1 The function of Theorem 3. (The plot is colored.)

Therefore, we may consider an arbitrary request sequence σ, referencing two items x and y that is served on a two-item list. We will compare the expected cost incurred by TS(l, p) to the cost of OPT on σ.

A simple observation is that at any time while TS(l, p) serves σ, the counter value of any item is uniformly distributed over $\{0, \ldots, l-1\}$, independently of the counters of other items. This holds true because the counter value of an item, at any time, is equal to its initial value minus the number of requests to that item served so far modulo l. We will use this fact repeatedly. Of course, care needs to be taken since the choices of TS(l, p) are highly correlated with the counter values.

The analysis of TS(l, p) crucially relies on a new phase partitioning of σ, together with a sophisticated cost charging scheme. More precisely, the service cost of a request to an item is paid at the next master request to that item, provided it occurs in the current phase. Extra care needs to be taken about items where this master request belongs to the next phase. The cost will then be further redistributed so that the expected service cost within a phase can be analyzed independently of counter values at the beginning or during the phase.

Phases and pre-refined cost

Phase partitioning

Given an arbitrary request sequence σ, we partition it into phases. The first phase starts with the first request in σ. Whenever OPT exchanges the two items x and y, the current phase ends and a new phase starts with the request to the item just moved forward. Recall that in response to a request, an algorithm may first exchange items. Then the request is served. Hence, when OPT swaps x and y, the most recent request served is the final one of the current phase. The upcoming request is the first one in the new phase. The last phase ends with the last request in σ.

Suppose that σ has been partitioned into phases $\lambda_1, \ldots, \lambda_k$. We will prove that, for any phase λ_i, TS(l, p)'s expected cost is bounded from above by c times the cost paid by OPT, where c is the ratio given in Theorem 3. This establishes the theorem. In analyzing a phase, we charge OPT a cost of d for the item swap at the end of the phase, in addition to the service costs. We will charge this cost of d also in the last phase λ_k. This way we overestimate OPT's cost on σ by d, which does not affect the competitive ratio.

Now consider an arbitrary phase $\lambda = \lambda_i$. In what follows, y denotes the item stored at the first position in OPT's list. Item x is the one stored at the second position, behind y.

Thus OPT incurs a service cost of 1 for each reference to x. Requests to y cost 0. Item x will be the *good item* because its service cost can be used to balance TS(l,p)'s cost. Item y will be the *bad item*.

Pre-refined cost

We wish to evaluate the algorithms' cost in λ without considering neighboring phases and by focusing on master requests. Therefore, in a series of two steps, we will pass over to the *refined cost setting*. First, we define the *pre-refined cost* for TS(l,p). The service cost incurred by TS(l,p) on a request to an item $z \in \{x,y\}$ is charged at the next master request to z in the phase, if it exists. This charging scheme is applied even if the reference to z itself is a master request. We still have to take care of service cost incurred on requests that are not followed by a master request in the phase.

For the item y, we simply append l requests to y at the end of the phase, right before OPT moves the element y to the back of its list. Then an additional master request to y occurs at the end of the phase and the service cost of previous, unpaid requests to y is covered. Indeed we will add $2l$ requests to y so that any phase ends with two master requests to y. This will be convenient in the further phase analysis. The service cost of OPT does not increase by the addition of requests to y. We remark that in analyzing a phase, we will make no assumptions regarding the end of the preceding phase. In particular, the analysis will not assume that $2l$ requests to x (or y) where appended to the preceding phase as this would impact the behavior of TS(l,p).

As for the requests to x that are not followed by a master request to x in the phase, we have to be more careful. Adding additional requests to x at the end of the phase is infeasible, since it would increase the costs of OPT. Therefore, regarding requests to x that are not followed by a master request to x in a given phase, we ignore their cost. Instead, at the first master request to x in the phase, if it exists, we charge TS(l,p) a cost of l no matter how many non-master requests have occurred so far. We show in the full paper that TS(l,p)'s expected cost does not decrease when changing over to the pre-refined cost setting. For further reference, the following definition summarizes this cost charging scheme.

▶ **Definition 5.** *In the pre-refined cost setting, $2l$ requests to the bad item y are appended at the end of a given phase.* TS(l,p)*'s cost incurred at a request to $z \in \{x,y\}$ is charged to the next master request to z in the phase, if such a request exists. At the first master request to the good item x in the phase, if it exists, a cost of l is charged to* TS(l,p).

▶ **Lemma 6.** TS(l,p)*'s expected cost in a phase does not decrease when passing over to the pre-refined cost setting.*

The proof is given in the full paper. The main idea is to show that, for item x, the expected extra cost charged at the first master request covers the expected cost ignored at the end of the phase. Note that there might be no master request to x in a phase consisting of less than l requests to x. In this case every request to x is a master request with probability $1/l$ and thus pessimistically charged cost $1/l \cdot l = 1$ in expectation. From now on we evaluate TS(l,p)'s cost in the pre-refined cost setting.

Counter fixing for x and refined cost

Given a phase λ, we split OPT's cost so that OPT also incurs service cost at the master requests to x, in addition to the cost for the paid exchange. In particular, this will allow us to fix the counter value of x at the beginning of λ and compare the cost of TS(l,p) to

that of OPT. Let $\mathbf{E}[C_{\mathrm{TS}}(\lambda)]$ denote TS(l,p)'s expected cost in λ. Moreover, let c_x be the counter value of x at the beginning of λ. Finally $\mathbf{E}[C^c_{\mathrm{TS}}(\lambda)]$ denotes TS(l,p)'s expected cost conditioned on $c_x = c$. Since c_x takes any value in $\{0,\ldots,l-1\}$ with equal probability $1/l$, $\mathbf{E}[C_{\mathrm{TS}}(\lambda)] = 1/l \cdot \sum_{c=0}^{l-1} \mathbf{E}[C^c_{\mathrm{TS}}(\lambda)]$.

Assume that λ contains $kl+j$ requests to x, for some $k \geq 0$ and $0 \leq j \leq l-1$. Then OPT's cost in λ is equal to $C_{\mathrm{OPT}}(\lambda) = d + kl + j$. Let k_c be the number of master requests to x in λ conditioned on $c_x = c$. If $c < j$, then $k_c = k+1$; otherwise $k_c = k$.

Define $C^c_{\mathrm{OPT}}(\lambda) := d + k_c l$. Then $1/l \cdot \sum_{c=0}^{l-1} C^c_{\mathrm{OPT}}(\lambda) = d + \sum_{c=0}^{l-1} k_c = d + j(k+1) + (l-j)k = d + kl + j = C_{\mathrm{OPT}}(\lambda)$. This implies

$$\frac{\mathbf{E}[C_{\mathrm{TS}}(\lambda)]}{C_{\mathrm{OPT}}(\lambda)} = \frac{1/l \cdot \sum_{c=0}^{l-1} \mathbf{E}[C^c_{\mathrm{TS}}(\lambda)]}{1/l \cdot \sum_{c=0}^{l-1} C^c_{\mathrm{OPT}}(\lambda)} \leq \max_c \left\{ \frac{\mathbf{E}[C^c_{\mathrm{TS}}(\lambda)]}{C^c_{\mathrm{OPT}}(\lambda)} \right\}.$$

Hence in the following we consider a fixed $c_x = c$ and upper bound $\mathbf{E}[C^c_{\mathrm{TS}}(\lambda)]/C^c_{\mathrm{OPT}}(\lambda)$. We emphasize that $C^c_{\mathrm{OPT}}(\lambda)$ charges service cost l to every master request to x.

We next partition a given phase λ into a *prephase* and a *postphase*, i.e. $\lambda = \lambda_{\mathrm{pre}}\lambda_{\mathrm{post}}$. The prephase λ_{pre} starts at the first request in λ and ends right before the first master request to x in the phase. If no master request to x exists in λ, then λ_{pre} ends with the last request in λ and the postphase is empty. Note that λ_{pre} cannot be empty, since the first request of the phase must go to y. The cost of d the algorithm OPT incurs due to the paid exchange made at the beginning of the phase accounts for the prephase, while the cost of l the optimum algorithm pays at any master request to x is charged in the postphase.

The remainder of this section is devoted to evaluating TS(l,p)'s expected cost on λ_{pre} and λ_{post}. For the analysis on λ_{post}, in a second step, we change over to a refined cost setting that applies in λ_{post}. In this framework TS(l,p) is charged a pessimistic cost of l at every master request to x if item x is not at the front of the list when the request occurs. If after a master request to x algorithm TS(l,p) has item x at the front of the list, then the request is charged an additional cost of $l/2$ to pay the service cost of references to y until the next master request to y occurs. A master request to y is assigned a cost of l if item y has been at the back of TS(l,p)'s list since the last master request to y. This covers the remaining cost for requests to y. Note in this case the preceding master request to y must have been treated using Policy 2. Lemma 8 below shows that TS(l,p)'s expected cost does not decrease when passing over to the refined cost.

▶ **Definition 7.** *In the refined cost setting for* TS(l,p) *in* λ_{post}*, a master request to* x *is charged a* base cost *of* l *if the master request is the first one to* x *in* λ_{post} *(and hence* λ*) or if* x *is not at the front of* TS(l,p)*'s list when the request is presented. An* additional cost *of* $l/2$ *is charged at the master request to* x *if, after service of the request,* x *is at the front of* TS(l,p)*'s list. A master request to* y *in* λ_{post} *is charged a* bad cost *of* l *if* y *has been at the back of* TS(l,p)*'s list since the last master request to* y*.*

▶ **Lemma 8.** *The expected cost of* TS(l,p) *in* λ_{post} *does not decrease when passing over to the refined cost.*

Proof. First consider a master request to either x or y, assuming that the referenced item is at the front of TS(l,p)'s list when the request occurs. Then the item must have been at the front of the list since the last master request to the item. Hence, no cost needs to be charged.

Next consider a master request to x and suppose that x is not at the front of TS(l,p)'s list when the request occurs. The refined cost framework charges the maximum possible service cost for l references to x. This reasoning also applies to the first master request to x in λ_{post}.

We next examine a master request Y to item y, assuming that y is not at the front of TS(l,p)'s list when Y occurs. If y has been at the back of TS(l,p)'s list since the last master request to y, the refined cost framework charges a cost of l to Y, which is equal to the true service cost for the last l requests to y. An overpayment might occur if the last master request to y was in the previous phase.

The interesting case is the one where y was at the front of TS(l,p)'s list after the previous master request to y was served. Then there must exist a master request X to item x where TS(l,p) moved x to the front of the list. We now assign the cost to be paid at Y to X instead. Let c_y^X denote the counter value of y at request X. There are c_y^X non-master requests to y, each incurring a cost of 1, that need to be paid at Y. This cost is now assigned to X. In the following we prove that the expected cost assigned to X, over TS(l,p)'s random choices and the counter value of y at X, is bounded above by $l/2$ times the probability that x is at the front of TS(l,p)'s list after X has been served. We remark that we analyze a slightly more pessimistic cost charging scheme. Any master request X to x is charged a cost of c_y^X if x resides at the front of TS(l,p)'s list after the service of X, independently of whether or not x was just exchanged with y.

Given any master request X to x, let $q(c) = q^X(c)$ be the random variable denoting the probability that item x is at the front of TS(l,p)'s list after X has been served, conditioned on $c_y^X = c$. For simplicity we denote the counter c_y^X of y at X by c_y. Then the expected cost charged at X is exactly

$$\mathbf{E}_{c_y}\left[q(c_y)c_y\right] = \sum_{c=0}^{l-1} \mathbf{P}\left[c_y = c\right] \mathbf{P}\left[x \text{ is at the front after serving } X \mid c_y = c\right] c_y.$$

The probability of x being at the front of TS's list after the service of X is

$$\mathbf{E}_{c_y}\left[q(c_y)\right] = \sum_{c=0}^{l-1} \mathbf{P}\left[c_y = c\right] \mathbf{P}\left[x \text{ is at the front after serving } X \mid c_y = c\right].$$

In the refined cost framework, the expected additional cost charged at X is $\mathbf{E}_{c_y}\left[q(c_y)\right] \cdot l/2 \geq \mathbf{E}_{c_y}\left[q(c_y)\right] \mathbf{E}_{c_y}\left[c_y\right]$. Hence it remains to prove that $\mathbf{E}_{c_y}[q(c_y)c_y] \leq \mathbf{E}_{c_y}[q(c_y)]\mathbf{E}_{c_y}[c_y]$. The last inequality holds if and only if the covariance of the random variables c_y and $q(c_y)$ is not positive. The latter property in turn holds if $q(c_y)$ is (non-strictly) decreasing in c_y. We verify this property by giving a complete characterization of the function $q(c_y)$. Let r denote the number of requests to y between X and the master request to x before that.

- If $c_y < l - r$, then the master request preceding X is to x and the probability of x being at the front of the list after X is 1.
- If $l - r \leq c_y < 2l - r$, then the two master requests preceding X are xy (in that order). The element x is moved to the front if and only if either X is served with Policy 1 or the preceding master request to y is served with Policy 2. Hence, the probability of x being at the front of the list is $q(c_y) = p + (1-p)^2$.
- If $2l - r \leq c_y$, then the two master requests preceding X both go to y and the probability of x being at the front of the list after the service of X is exactly p, the probability that X was handled by TS(l,p) using Policy 1.

In particular the function $q(c_y)$ is decreasing. ◀

The cost analysis of a phase

Consider an arbitrary phase λ. Let λ_{post} be its postphase, which starts with a master request to x and ends with at least two master requests to y, according to the phase adjustment we did when changing over to the pre-refined cost. We next modify λ_{post} so that we can partition it into subphases, each ending with at least two master requests to y. For this

purpose consider two consecutive master requests to x that are preceded by a single master request to y. The next proposition shows that we can insert l new requests to y, thereby generating a new master request to y, without decreasing the strict competitiveness on λ_{post}. The proof is given in the full version of the paper.

▶ **Proposition 9.** *Consider a subsequence of consecutive master requests $x_0y_1x_1x_2$ in λ_{post} where master request z_i goes to z, $z \in \{x, y\}$. Add l new requests to y before x_1.* TS(l, p)*'s expected refined cost on the resulting sequence of master requests $x_0y_1y_2x_1x_2$ is at least as high as that on the former sequence.* OPT*'s cost does not change.*

Hence in λ_{post}, whenever two master requests to x are preceded by exactly one master request to y, we insert l new requests to y. Again, this creates a second master request to y. We then partition λ_{post} into subphases, each ending with two master requests to y. More precisely, the first subphase starts with the first master request in λ_{post}. A subphase ends immediately before a master request to x that is preceded by at least two master requests to y. Observe that whenever at least two consecutive master requests to x occur, they start a new subphase. We obtain two types of subphases, specified by their master requests.

- Type 1: $x(yx)^k y^{2+i}$ for some $i, k \geq 0$.
- Type 2: $x^{2+j}(yx)^k y^{2+i}$ for some $i, j, k \geq 0$.

In the following four lemmas we analyze TS(l, p) on any subphase. If the subphase is the first one in λ_{post}, we analyze it jointly with the preceding prephase λ_{pre}. Together the four lemmas imply Theorem 3. The cost ratios c_i, $1 \leq i \leq 6$, stated in the lemmas are identical to those of Theorem 3. In Lemma 10 we first consider Type 2 subphases as TS(l, p)'s performance ratio on those phases can be bounded independent of the preceding master requests. Then Lemmas 11, 12 and 13 address Type 1 subphases with the preceding master requests. Lemma 11 considers the case that a Type 1 subphase is preceded by two master requests to y. Note that this is, in particular, the case for any Type 1 subphase that is not the first subphase in the given λ. Lemmas 12 and 13 address the special cases where a Type 1 subphase is preceded (a) by master requests to x and y, in this order, or (b) by a master request to x. In both cases such a subphase must be the first one in λ_{post}. Moreover, λ_{pre} consists of less than two master requests to y in these special cases. In fact in case (b), there is no master request in λ_{pre}. Proofs of all the lemmas are presented in the full paper.

▶ **Lemma 10.** TS(l, p) *is strictly* $\max\{c_1, c_2, c_4\}$*-competitive on any subphase of Type 2, including a possible preceding prephase.*

▶ **Lemma 11.** TS(l, p) *is strictly* $\max\{c_1, c_3, c_4\}$*-competitive on any subphase of Type 1, including a possible prephase, if the subphase is preceded by two master requests to y.*

▶ **Lemma 12.** TS(l, p) *is strictly* $\max\{c_1, c_4, c_5\}$*-competitive on any subphase of Type 1 and its leading prephase if the subphase is preceded by master requests to x and y in this order.*

▶ **Lemma 13.** TS(l, p) *is strictly* $\max\{c_1, c_4, c_6\}$*-competitive on any subphase of Type 1 and its leading prephase if the subphase is preceded by a master request to x.*

4 Lower bounds

We develop lower bounds in the partial cost and the full cost P^d models.

▶ **Theorem 14.** *Let A be a randomized online algorithm for list update in the partial cost P^d model. If A is c-competitive against oblivious adversaries, then $c \geq 2 - \frac{1}{2d}$. This holds even for request sequences referencing only two items.*

We conjecture that a lower bound of $2 - \frac{1}{2d}$ also holds for the full cost P^d model. The difficulty is to bound the cost of OPT from above on request sequences referencing a general set of n items. We can establish the following lower bound in the full cost P^d model.

▶ **Theorem 15.** *Let A be a randomized online algorithm for list update in the full cost P^d model. If A is c-competitive against oblivious adversaries, then c is at least*

$$\frac{1}{1+2W\left(\frac{-1}{2e}\right)} - O\left(\frac{1}{d}\right) \approx 1.8654.$$

Here W is the upper branch of the Lambert-W-function, i.e. $W\left(\frac{-1}{2e}\right)$ is largest value x satisfying $2xe^{x+1} = -1$.

Table 1 presents the values of the lower bound of Theorem 15, for small values of d, up to $d = 100$. For $d = 1$, i.e. the standard cost model, our bound matches the one by Teia [27].

Table 1 The lower bound in the full cost P^d model for various d.

d	$c(d)$
1	1.5
2	1.8036
3	1.8270
4	1.8337
5	1.8420

d	$c(d)$
6	1.8438
7	1.8485
10	1.8531
20	1.8594
100	1.8642

A probability distribution on request sequences

For the proof of the two theorems above, we use Yao's minimax principle [28]. We define a probability distribution on request sequences and compare the expected cost incurred by any (deterministic) online algorithm A to that of OPT. For the definition of our probability distribution, we describe how to sample a request sequence according to it. Let $L(0) = [x_0 \dots x_{n-1}]$ be a starting list of n items; x_i precedes x_{i+1} for $i = 0, \dots, n-2$. Throughout the sampling process the list is static, i.e. no rearrangement of items is done. The items will be requested in a cyclic fashion, in decreasing order of index. Each item will be referenced a certain number of times.

Formally, in addition to $L(0)$, the sampling process takes as input a number $N \in \mathbb{N}$ and a starting item $x \in L(0)$. Typically, the starting item is equal to the last item x_{n-1} in the list but the process is defined for any $x \in L(0)$. The sampling process produces a request sequence consisting of N segments. Throughout this section a *segment* is a maximal subsequence of requests to the same item. Moreover, to simplify notation, we set $x_i = x_{i \bmod n}$, for all $i \in \mathbb{Z}$.

Initially, the request sequence σ to be produced is equal to the empty string. In each step of the sampling procedure, if $N > 0$, a request to the current item x is appended at the end of σ. Then with probability $p = 1/(2d)$, the value N is decremented and $x = x_i$ is replaced by x_{i-1}. Hence in this event the segment of requests to x_i ends and a segment of references to x_{i-1} starts. Note that the length of a segment is geometrically distributed with $p = 1/(2d)$ and thus equal to $2d$ in expectation. The process stops when $N = 0$. A pseudo-code description of the sampling process is given in Algorithm 2. Let $\mathcal{S}_N[x]$ denote the resulting probability distribution on request sequences with starting item x. Furthermore, let $\mathcal{S}_N = \mathcal{S}_N[x_{n-1}]$.

The lower bound construction in the full cost model will work with sequences generated according to this particular process. In the partial cost model we will have to extend the process so that the request sequences admit a phase partitioning and end with a complete phase.

Algorithm 2 SampleRequestSequence.

Input: $L(0) = [x_0 \dots x_{n-1}]$, $N \in \mathbb{N}$ and $x \in L(0)$.
$\sigma :=$ empty string;
while $N > 0$ **do**
    Append x to σ;
    With probability $p = \frac{1}{2d}$, decrement N and replace $x = x_i$ by x_{i-1};
Return σ;

We next introduce the notion of an algorithm being *uncompetitive*. It will be particularly useful when deriving our lower bound in the full cost model. Nonetheless, the notion applies to both the partial and the full cost models and it will always be clear from the context which model is used.

▶ **Definition 16.** *Let $c \geq 1$. An online algorithm A is* c-uncompetitive against an offline algorithm B *if, for every $\varepsilon > 0$, there exists an initial list $L(0)$ such that*

$$\lim_{N\to\infty} \frac{\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_A(\sigma)]}{\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_B(\sigma)]} \geq c - \varepsilon.$$

If $B = \mathrm{OPT}$, algorithm A is simply c-uncompetitive. *Finally, A is* c-uncompetitive on two-item sequences (against B) *if the above condition holds with $\varepsilon = 0$, for lists consisting of two items.*

The terminology is relevant due to the following obvious fact.

▶ **Lemma 17.** *If a (possibly randomized) online algorithm A is c-uncompetitive, then its competitive ratio is not smaller than c.*

In a first step we bound the expected cost incurred by any online algorithm A on request sequences generated according to $\mathcal{S}_N$ from below. In Theorem 14, we consider the partial cost model and request sequences referencing two items. In the proof of Theorem 15, we show that we may restrict ourselves to the partial cost model and, again, may focus on two-item request sequences if we consider a projective offline algorithm. Therefore, the following Lemma 18 will be essential in the proofs of Theorems 14 and 15. Let $L(0) = [xy]$ be a list consisting of two items x and y; where x is initially stored in front of y. Consider the distribution $\mathcal{S}_N = \mathcal{S}_N[y]$.

▶ **Lemma 18.** *Let $L(0) = [xy]$. For every online algorithm A and every N, we have in the partial cost model*

$$\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_A(\sigma)] \geq dN.$$

Proof. It suffices to prove the lemma for deterministic algorithms because any randomized algorithm is a probability distribution over deterministic ones. Hence, let us assume for the sake of a contradiction that there were a deterministic online algorithm A and an $N \in \mathbb{N}$ such that

$$C = \mathbf{E}_{\sigma\sim\mathcal{S}_N}\left[C_A^{\mathrm{part}}(\sigma)\right] < dN.$$

Among all possible choices, we choose such a pair (A, N) with N minimal and, in case of ties, C minimal. We clearly have $N > 0$. Given a sequence σ we denote by $\alpha(\sigma)$ the number of leading requests to y and by $\beta(\sigma)$ the number of requests to x following those.

We show that the algorithm A does not immediately move the element y to the front of its list. Indeed, let us write $\sigma = y^{\alpha(\sigma)}\sigma'$. If we choose $\sigma \sim \mathcal{S}_N$ and pass over to σ', it is the same as choosing $\sigma' \sim S_{N-1}[x]$. Now if A were to move the item y immediately to the front, it would incur exactly cost d on the sequence $y^{\alpha(\sigma)}$. Then, by the minimality of N, it will incur an expected cost of at least $d(N-1)$ on σ' and hence a total expected cost of at least dN. This is a contradiction to the assumption that $C < dN$.

Hence we may assume that A does not immediately move y to the front of its list. For $\sigma \sim \mathcal{S}_N$ we consider two cases. With probability $\frac{1}{2d}$, the sequence σ starts with a single request to y, i.e. we have $\alpha(\sigma) = 1$. Then $\sigma = yx^{\beta(\sigma)}\sigma''$. Note that we have $\sigma" \sim S_{N-2}$ if we pick $\sigma \sim \mathcal{S}_N|_{\alpha(\sigma)=1}$. On the sequence $yx^{\beta(\sigma)}$ the algorithm A incurs cost of exactly 1 at the first request. By the minimality of N we have for the remainder $\sigma"$ of the sequence σ

$$\mathbf{E}_{\sigma"}\left[C_A(\sigma")\right] \geq d(N-2).$$

Note that this is obviously true if $N - 2 \leq 0$ holds.

On the other hand, with probability $1 - \frac{1}{2d}$, we have $\alpha(\sigma) > 1$. In this case the algorithm A incurs cost 1 on the first request to y. We consider the algorithm B which behaves like the algorithm A after having read a request to y, i.e. on the input sequence σ it behaves like A would on the corresponding suffix of $y\sigma$. By the minimality of C we have $\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_B(\sigma)] \geq C$. Note that sampling σ from $\mathcal{S}_N$ conditioned on it starting with at least two requests to y is the same as sampling σ from $\mathcal{S}_N$ and appending a request to y to its front. Hence we get

$$\mathbf{E}_{\sigma\sim\mathcal{S}_N}\left[C_A(\sigma) \mid \alpha(\sigma) > 1\right] = 1 + \mathbf{E}_{\sigma\sim\mathcal{S}_N}\left[C_B(\sigma)\right] \geq 1 + C.$$

In total we have

$$\begin{aligned} C &= \tfrac{1}{2d}\mathbf{E}_{\sigma\sim\mathcal{S}_N}\left[C_A(\sigma) \mid \alpha(\sigma) = 1\right] + \left(1 - \tfrac{1}{2d}\right)\mathbf{E}_{\sigma\sim\mathcal{S}_N}\left[C_A(\sigma) \mid \alpha(\sigma) > 1\right] \\ &\geq \tfrac{d(N-2)+1}{2d} + \left(1 - \tfrac{1}{2d}\right)(C+1) \\ &= \tfrac{dN}{2d} + \left(1 - \tfrac{1}{2d}\right)C. \end{aligned}$$

The last inequality implies $C \geq dN$. Again, we have reached the desired contradiction. ◀

The challenging part in the proofs of Theorems 14 and 15 is to bound the expected cost incurred by OPT on sequences drawn according to $\mathcal{S}_N$. In the following we sketch some of the main ideas. Full analyses are presented in the full paper.

Proof sketch for Theorem 14

We focus on request sequences referencing two items x and y, and hence on sequences $\sigma \sim \mathcal{S}_N$ with initial list $L(0) = [xy]$. Such sequences consist of N segments that in turn reference y and x. Given a sequence σ' and an item $z \in \{x, y\}$, let $|\sigma'|_z$ be the number of requests to z in σ'.

The optimal algorithm for two-item sequences is the following: We consider the case where y is at the front of the list of OPT and x is at the back. The opposite case works symmetrically. If y is requested next, OPT will obviously not move it to the back, but wait, until x is requested. If x is requested next, OPT needs to decide whether to move x to the front of its list. It does so if and only if there is a prefix σ' of future requests

with $|\sigma'|_x = |\sigma'|_y + 2d$ and no (non-empty) prefix σ'' of σ' satisfies $|\sigma''|_x \leq |\sigma''|_y$. There is a special case if prefix σ' comprises the entire sequence of future requests. Then we only require in the first condition that $|\sigma'|_x \geq |\sigma'|_y + d$ holds true. In the following we will analyze a simpler algorithm O, which omits this special case. While O is only close-to-optimal, it still gives a good upper bound on OPT. Additionally, we will consider the algorithm $\bar{O}$ that always keeps its list in opposite order, compared to the list of O. On each request in a given sequence, exactly one of the two algorithms has a service cost of 1.

Given any sequence σ, let $\tilde{C}_O(\sigma)$ and $\tilde{C}_{\bar{O}}(\sigma)$ be the pure service cost of O and $\bar{O}$. Furthermore, let $D_O(\sigma)$ be the cost incurred by O for paid exchanges. Define $K(\sigma) = \tilde{C}_{\bar{O}}(\sigma) - \tilde{C}_O(\sigma) - 2D_O(\sigma)$. For $\sigma \sim \mathcal{S}_N$, $K(\sigma)$ is a random variable, which we denote by E_N. We will show that $\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_O(\sigma)] = dN - \mathbf{E}[E_N]/2$. Thus, by Lemma 18, the value $\mathbf{E}[E_N]/2$ is a lower bound for the cost the algorithm O saves compared to any online algorithm. The heart of the analysis is to estimate that $\lim_{N\to\infty} \mathbf{E}[E_N]/N \geq 2d(2d-1)/(4d-1)$. Together with Lemmas 18, this implies that any online algorithm A is c-uncompetitive against OPT with $c \geq dN/(dN - \frac{N}{2}\frac{2d(2d-1)}{4d-1}) = 2 - \frac{1}{2d}$. In particular, by Lemma 17 its competitive ratio is at least $2 - 1/(2d)$.

For the analysis of $\mathbf{E}[E_N]$, we partition a request sequence $\sigma \sim \mathcal{S}_N$ into phases. Each phase λ consists of a series of subphases $\mu_1, \ldots, \mu_l$. We describe how to obtain μ_1. At the beginning of λ, let $z \in \{x, y\}$ be the current item in the sampling process, i.e. the first segment in λ consists of requests to z. Let $z' \in \{x, y\}$, $z' \neq z$, be the other item. Starting at the beginning of λ we scan the generated requests, adding them to μ_1 until one of the following events occurs. (1) $|\mu_1|_z = |\mu_1|_{z'}$ or (2) $|\mu_1|_z = |\mu_1|_{z'} + 2d$. In case (1) we call μ_1 a *zero-subphase*; in case (2) μ_1 is an *up-subphase*. When event (1) or (2) occurs, the remaining requests of the current segment are appended to μ_1. These requests form the *post-subphase*. Then μ_1 ends. Each following subphase μ_i, for $i > 1$, is obtained in the same way, starting at the end of μ_{i-1}. Phase λ ends when, for the first time, an up-subphase is obtained. Formally, algorithm O moves item z to the front of the list right before the up-subphase.

We extend the sampling process so as to obtain sequences ending with a complete phase. This induces a slightly related probability distribution of request sequences, which is not critical though. For any λ, let $C = K(\lambda)$ and let $R = R(\lambda)$ be the number of segments in λ. At the heart of the analysis, using a recurrence relation, we prove $\lim_{N\to\infty} \mathbf{E}[E_N]/N \geq \mathbf{E}[C]/\mathbf{E}[R]$. We argue that $\mathbf{E}[C]$ is the total length of all post-subphases in λ. Then we show $\mathbf{E}[C] = (2d-1)/(1-P_0)$ and $\mathbf{E}[R] = (4d-1)/(2d(1-P_0))$, where P_0 is the probability that a subphase happens to be a zero-subphase. This yields the desired bound.

Proof sketch for Theorem 15

In this setting we are given a list $L = L(0)$ with n items. Again, we focus on request sequences generated according to $\mathcal{S}_N$ (Algorithm 2). We first prove that we may restrict ourselves to the partial cost model and two-item request sequences if we focus on *projective* offline algorithms: If every deterministic online algorithm is c-uncompetitive on two-item sequences against a projective offline algorithm B in the partial cost model, then every deterministic online algorithm is c-uncompetitive against B in the full cost model. Nonetheless, the analysis of the offline algorithm O developed for the proof of Theorem 15 cannot be employed because O and OPT are not projective.

Therefore, we define a family of projective offline algorithms B_h, for $0 < h < 2d$. Consider a request sequence to be served on a list L consisting of n items. When presented with any request, B_h works as follows: If the next h requests reference the same element $z \in L$, algorithm B_h moves z to the front of its list. Otherwise it does not change the list. Algorithm

B_h is projective, for sequences σ generated by $\mathcal{S}_N$, because items are requested in cyclic order in σ and a projection to item pairs does not create longer subsequences of the same item.

Given the result for projective offline algorithms stated above, it suffices to analyze B_h on two-item sequences. Let $L(0) = [xy]$ be the starting list. Technically, the main step is to show that, for any $\sigma \sim \mathcal{S}_N$, the expected cost incurred by B_h is

$$\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_{B_h}(\sigma)] = \frac{\left(\frac{1-(1-p)^h}{p} - h(1-p)^{h-1}\right) + (1-p)^{h-1}d}{2-(1-p)^{h-1}}N + o(N), \tag{1}$$

where $p = 1/(2d)$. In order to establish this equation, given $\sigma \sim \mathcal{S}_N$, we write $\sigma = z_1^{\alpha_1} z_2^{\alpha_2} \dots z_N^{\alpha_N}$, where $z_1 = y$ and $z_2 = x$. If we consider the algorithm B_h at the beginning of the subsequence $z_t^{\alpha_t}$, there can be three cases.

- If z_t is at the back of the list of B_h and $\alpha_t \geq h$ holds, we say the sequence $z_t^{\alpha_t}$ is of *type* h. The algorithm B_h will incur a cost d on it because it immediately moves z_t to the front.
- If z_t is at the back of the list of B_h and $\alpha_t = j < h$, we say the sequence $z_t^{\alpha_t}$ is of *type* $(j, 1)$. The algorithm B_h will incur cost j on it.
- If z_t is at the front of the list of B_h, the algorithm will incur no cost on it. Further we have $t > 1$ and $\alpha_{t-1} = j < h$, for some j. We say the sequence $z_t^{\alpha_t}$ is of *type* $(j, 2)$.

For a type $w \in W_h = \{1, \dots, h-1\} \times \{1, 2\} \cup \{h\}$, let $P(t)_w$ be the probability that the sequence $z_t^{\alpha_t}$ is of this type if we sample $\sigma \sim \mathcal{S}_N$. The process forms a Markov chain: From type $(j, 1)$, for $j < h$, we pass to the type $(j, 2)$ with probability 1. From every other type w we pass to type $(j, 1)$, for $j < h$, with probability $p(1-p)^{j-1}$ and to type h with probability $(1-p)^{h-1}$. Using basic Markov analysis we evaluate $\lim_{t\to\infty} P(t)_w$, for $w = (j, 1)$, $w = (j, 2)$ and $w = h$. Using the respective expressions, we obtain (1). Using this expression, we can immediately verify the competitive ratios in table 1 using the following optimal choices of h.

Table 2 The best choices of h, for small values of d.

d	$c(d)$	h
1	1.5	1
2	1.8036	3
3	1.8270	5
4	1.8337	6
5	1.8420	8

d	$c(d)$	h
6	1.8438	10
7	1.8485	11
10	1.8531	16
20	1.8594	31
100	1.8642	154

To obtain the lower bound for $d \to \infty$ we set $h = \lfloor 2\tilde{h}d \rfloor$ for some $\tilde{h} > 0$ to be determined later. Then $\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_{B_h}(\sigma)]$ is of the form $(1 + \frac{2\tilde{h}e^{-\tilde{h}}}{e^{-\tilde{h}}-2} + O\left(\frac{1}{d}\right))dN + o(N)$. Lemma 18 implies that every online algorithm A is c-uncompetitive against the algorithm B_h in the full cost model, where c is at least $(1 + 2\tilde{h}e^{-\tilde{h}}/(e^{-\tilde{h}} - 2))^{-1}$ as $d \to \infty$. Lemma 17 ensures that the competitive ratio of A is not smaller than the above expression. Theorem 15 follows if we set $\tilde{h} = W(-1/(2e)) + 1$.

References

1 S. Albers. Improved randomized on-line algorithms for the list update problem. *SIAM J. Comput.*, 27(3):682–693, 1998.

2 S. Albers and S. Lauer. On list update with locality of reference. *J. Comput. Syst. Sci.*, 82(5):627–653, 2016.

3 S. Albers, B. von Stengel, and R. Werchner. A combined BIT and TIMESTAMP algorithm for the list update problem. *Inf. Process. Lett.*, 56(3):135–139, 1995.
4 S. Albers and J. Westbrook. Self-organizing data structures. In *Online Algorithms, The State of the Art*, Springer LNCS 1442, pages 13–51, 1998.
5 C. Ambühl, B. Gärtner, and B. von Stengel. Optimal lower bounds for projective list update algorithms. *ACM Trans. Algorithms*, 9(4):31:1–31:18, 2013.
6 S. Angelopoulos and P. Schweitzer. Paging and list update under bijective analysis. *J. ACM*, 60(2):7:1–7:18, 2013.
7 A. Borodin and R. El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press, 1998.
8 J. Boyar, S. Kamali, K.S. Larsen, and A. López-Ortiz. On the list update problem with advice. *Inf. Comput.*, 253:411–423, 2017.
9 M. Burrows and D. Wheeler. A block sorting lossless data compression algorithm. *Technical Report 124*, 1994.
10 M. Crochemore, R. Grossi, J. Kärkkäinen, and G.M. Landau. Computing the Burrows-Wheeler transform in place and in small space. *J. Discrete Algorithms*, 32:44–52, 2015.
11 R. Dorrigiv, M.R. Ehmsen, and A. López-Ortiz. Parameterized analysis of paging and list update algorithms. *Algorithmica*, 71(2):330–353, 2015.
12 S. Irani. Two results on the list update problem. *Inf. Process. Lett.*, 38(6):301–306, 1991.
13 S. Kamali, S. Ladra, A. López-Ortiz, and D. Seco. Context-based algorithms for the list-update problem under alternative cost models. In *Proc. 2013 Data Compression Conference (DCC)*, IEEE, pages 361–370, 2013.
14 S. Kamali and A. López-Ortiz. A survey of algorithms and models for list update. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of his 66th Birthday*, pages 251–266, 2013.
15 S. Kamali and A. López-Ortiz. Better compression through better list update algorithms. In *Proc. 2014 Data Compression Conference (DCC)*, IEEE, pages 372–381, 2014.
16 R. Karp and P. Raghavan. Personal communication cited in [24].
17 A. López-Ortiz, M.P. Renault, and A. Rosén. Paid exchanges are worth the price. In *Proc. 32nd International Symposium on Theoretical Aspects of Computer Science (STACS15)*, LIPIcs 30, pages 636–648, 2015.
18 M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive algorithms for on-line problems. In *Proc. 20th Annual ACM Symposium on Theory of Computing*, pages 322–333, 1988.
19 G. Manzini. An analysis of the Burrows-Wheeler transform. *J. ACM*, 48(3):407–430, 2001.
20 C. Martínez and S. Roura. On the competitiveness of the move-to-front rule. *Theor. Comput. Sci.*, 242(1-2):313–325, 2000.
21 J. McCabe. On serial files with relocatable records. *Operations Research*, 12:609–618, 1965.
22 J.I. Munro. On the competitiveness of linear search. In *Proc. 8th Annual European Symposium on Algorithms (ESA00)*, Springer LNCS 1879, pages 338–345, 2000.
23 N. Reingold and J. Westbrook. Off-line algorithms for the list update problem. *Inf. Process. Lett.*, 60(2):75–80, 1996.
24 N. Reingold, J. Westbrook, and D.D. Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11(1):15–32, 1994.
25 J. Sirén. Burrows-Wheeler transform for terabases. In *Proc. 2016 Data Compression Conference (DCC)*, IEEE, pages 211–220, 2016.
26 D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.
27 B. Teia. A lower bound for randomized list update algorithms. *IPL*, 47(1):5–9, 1993.
28 A.C.C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proc. 18th IEEE Annual Symposium on Foundations of Computer Science*, pages 222–227, 1977.

On Covering Segments with Unit Intervals

Dan Bergren
School of Computing, DePaul University, Chicago, USA
bergren2@gmail.com

Eduard Eiben
Department of Computer Science, Royal Holloway, University of London, Egham, UK
eduard.eiben@rhul.ac.uk

Robert Ganian
Algorithms and Complexity Group, Vienna University of Technology, Vienna, Austria
rganian@gmail.com

Iyad Kanj
School of Computing, DePaul University, Chicago, USA
ikanj@cs.depaul.edu

Abstract

We study the problem of covering a set of segments on a line with the minimum number of unit-length intervals, where an interval covers a segment if at least one of the two endpoints of the segment falls in the unit interval. We also study several variants of this problem.

We show that the restrictions of the aforementioned problems to the set of instances in which all the segments have the same length are NP-hard. This result implies several NP-hardness results in the literature for variants and generalizations of the problems under consideration.

We then study the parameterized complexity of the aforementioned problems. We provide tight results for most of them by showing that they are fixed-parameter tractable for the restrictions in which all the segments have the same length, and are W[1]-complete otherwise.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Computational geometry

Keywords and phrases Segment covering, unit intervals, NP-completeness, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.13

Funding *Robert Ganian*: Robert Ganian acknowledges support by the Austrian Science Fund (FWF, project P31336).

1 Introduction

Problem Definition and Motivation. The problem of covering a set of points on the (real) line with the minimum number of closed unit-length intervals is a classical problem that can be solved in polynomial time by a simple greedy algorithm (e.g., see exercise 16.2-5 in [8] and exercise 5 in chapter 4 of [18]). A generalization of the above problem to that of covering a set of segments on the real line with the minimum number of unit intervals, where an interval *covers* a segment if at least one endpoint of the segment is in the interval, has been studied in several works [2, 3, 4]. For clarity, throughout the paper, we distinguish the entities to be covered from those used for covering, by referring to the former as *segments* and the latter as *intervals*. It is easy to see that the greedy algorithm – referred to above – no longer works for this generalization. In fact, this generalization turns out to be NP-hard, even though a straightforward (polynomial-time) greedy algorithm works for the restriction in which all segments have length at most 1 (unit).

Recently, several variants and generalizations of the above segment covering problem have been considered (as discussed below). Two natural variants arise based on whether the unit intervals can be arbitrarily chosen on the line, or are restricted to a given input set; the former version has been referred to as the *continuous* version as opposed to the latter *discrete* version. Moreover, a more restricted notion of covering has been studied as well, that we refer to henceforth as *exact covering*, in which exactly one endpoint from each segment must be covered by the unit intervals.

In this paper, we study the classical and parameterized complexity of the variants of segment covering by unit intervals discussed above, in most cases providing tight characterizations. The problem variants we study are: Continuous Segment Covering (Cont-SC); Discrete Segment Covering (Disc-SC); Continuous Exact Segment Covering (Cont-Exact-SC); and Discrete Exact Segment Covering (Disc-Exact-SC).

Related Work. Arkin et al. [2, 4] studied the exact covering problem of a set of *color classes*, where each color class contains two points on the real line of the same color, with the minimum number of intervals; here an interval covers a color class if it covers exactly one point from the color class. This is precisely the notion of exact segment covering, in which each color class corresponds to a segment whose endpoints are the two points in the class. It was shown in [2] that the aforementioned problem is NP-hard, and that the case in which the intervals are restricted to be unit intervals is NP-hard as well.

Arkin et al. [3, 4] also studied the problem of finding a *conflict-free* covering, where a color class can have both points covered, but it is not allowed to use an interval that covers both points of any color class. They showed that both the discrete and continuous versions of the aforementioned problem are NP-hard, and gave approximation algorithms of ratios 3 and 2, respectively, for them. They also studied a problem variant in which each color class consists of a horizontal or a vertical unit-length segment in the plane, and the goal is to compute a minimum-cardinality set of axes-parallel unit squares such that exactly one point from each segment is covered by the unit squares. They showed that this variant is NP-hard, and gave an approximation algorithm of ratio 6 for it. Achaaryya et al. [1] studied several variants of covering segments with axes-parallel unit squares in the plane. They obtained approximation algorithms and showed the NP-hardness of the variant in which all segments are horizontal unit segments.

The Cont-Exact-SC and Disc-Exact-SC problems under consideration are also related to an NP-hard combinatorial problem, referred to as the "Paintshop" problem [5, 15], that has applications in automotive industry. Other applications of covering line segments (referred to as "stabbing") with geometric objects (such as unit disks/squares) are in the area of networks security (see [1, 19]).

Finally, we mention that there is a vast amount of literature on other notions of covering and stabbing of geometric objects [7, 12, 13, 20, 21, 22].

Our Results. Our results for the Cont-SC, Disc-SC, Cont-Exact-SC, and Disc-Exact-SC problems can be summarized as follows (recall that the covering elements in all these segment covering variants are unit intervals):

(i) The restrictions of Cont-SC, Disc-SC, Cont-Exact-SC, and Disc-Exact-SC to instances in which all segments have the same length are NP-hard.
This NP-hardness result has important implications. First, it strengthens and implies several NP-hardness results in the literature about segment covering. The NP-hardness of Cont-Exact-SC implies the NP-hardness result stated in Theorem 6 of [2]. Moreover,

since we (can) assume that the uniform segment length in these restrictions of Cont-SC and Disc-SC is more than 1 (otherwise, the problem is polynomial-time solvable by a simple greedy algorithm), our NP-hardness result for Disc-SC implies the NP-hardness result in Theorem 1 of [3] (since the segment length is more than 1 unit and the intervals are unit intervals, the covering obtained is automatically a conflict-free covering). Second, the NP-hardness results for Cont-SC, Disc-SC, Cont-Exact-SC, and Disc-Exact-SC refine the complexity of these problems. For Cont-SC and Disc-SC, we already know that the slices of these problems consisting of instances in which each segment has length at most 1 unit are solvable in polynomial time by a greedy algorithm. (Note that we do not know if the same holds for Cont-Exact-SC and Disc-Exact-SC, as we do not know the complexity of their restrictions to instances in which each segment has length at most 1.) The above result shows that the slices of Cont-SC, Disc-SC, Cont-Exact-SC, and Disc-Exact-SC, consisting of instances in which all segments have the same length, are NP-hard.

The crucial insight required for our NP-hardness results is that, while the problems are one-dimensional, instances where the length of all segments differs significantly from the length of all intervals in fact behave like two-dimensional objects. We employ this in our proof by devising a series of two reductions, where we begin by considering a 2-dimensional segment covering problem whose instances are "nicely" embedded on a grid. We show that this aforementioned problem is NP-hard via a reduction from the restriction of Planar Vertex Cover to instances that are also "nicely" embedded on a grid. It is worth noting that, while the idea of proving NP-hardness by reducing from a problem with nice embedding properties (e.g., Planar 3-SAT) has been used in previous work [1, 4], the presented reduction stands out due to requiring complex "modularly constructed gadgets". We compose the above reduction with a second one that maps the segment covering problem on the grid to our 1-dimensional segment covering problems.

(ii) We show that the restrictions of Cont-SC, Disc-SC, Cont-Exact-SC, and Disc-Exact-SC to instances in which all segments have the same length are fixed-parameter tractable (FPT). The FPT algorithm for Cont-SC combines several algorithmic ideas. (The other FPT algorithms are similar.) It starts by computing an approximate solution of ratio 3 whose intervals contain *all* (input) segment endpoints. The algorithm then branches on all possibilities to determine how the approximate solution "interacts" with an optimal solution. Based on the determined interaction, the algorithm identifies endpoints of segments in the approximate solution, called *anchors*, around which the intervals in an optimal solution are *anchored* (i.e., placed). The goal then becomes to assign the endpoints of the segments to the anchors, where assigning an endpoint to an anchor means that the endpoint (and hence the associated segment) is covered by the interval in the optimal solution placed around that anchor. The algorithm then exploits the restriction that all segments have the same length, to define a domination relation among the anchors affecting each segment, which is then revealed through further branching. With these domination relations revealed, the resulting problem can be modeled as an instance of 2-Sat, which is solvable in polynomial time.

(iii) We show that Disc-SC and Cont-SC are W[1]-complete. Membership in W[1] is proved using the characterization of W[1] by Chen et al. [6], whereas the W[1]-hardness is proved via an FPT-reduction from the Multicolored Clique problem.

This reduction is quite involved, requiring gadget constructions that extend beyond the standard toolkit used in conventional W[1]-hardness reductions from Multicolored

Clique. The W[1]-completeness results, in conjunction with the results in (ii) above, provide tight results for the parameterized complexity of Disc-SC and Cont-SC. Note that, while the restrictions of Disc-SC and Cont-SC to instances in which all segments have equal length have the same classical complexity as their general counterparts, these restrictions exhibit a different behavior in terms of their parameterized complexity. The parameterized complexity of Cont-Exact-SC and Disc-Exact-SC remains open.

2 Preliminaries

We assume familiarity with the basic notation and terminology used in graph theory and parameterized complexity. We refer the reader to the standard books [10, 11] for more information on these subjects. The asymptotic notation $\mathcal{O}^*$ suppresses a polynomial factor in the input length. For $r \in \mathbb{N}$, we write $[r]$ as shorthand for the set $\{1, \ldots, r\}$.

We provide a brief overview of the basic parameterized complexity terminology used throughout the paper. A *parameterized problem* Q is a subset of $\Omega^* \times \mathbb{N}$, where Ω is a fixed alphabet. Each instance of Q is a pair (x, k), where $k \in \mathbb{N}$ is called the *parameter*. The problem Q is called *fixed-parameter tractable* (FPT) if it can be solved in time $f(k) \cdot |x|^{\mathcal{O}(1)}$, where f is a computable function. On the other hand, showing that a parameterized problem Q is hard for the parameterized complexity class W[1] provides strong conditional evidence that Q is not FPT. This is usually done by obtaining a suitable *FPT-reduction*, i.e., a reduction which runs in time $f(k) \cdot |x|^{\mathcal{O}(1)}$ and where the parameter of the output instance is upper-bounded by a function of the parameter of the input instance. We refer to the books by Downey and Fellows [11] and Cygan et al. [9] for an in-depth introduction to parameterized complexity.

2.1 Segment Covering Problems

The following problem will serve as a baseline for our problem definitions.

Discrete Segment Covering (Disc-SC)
Given: A set Γ of n intervals (called *segments* from here on out) on the rational line; a set $\mathcal{I}$ of unit-intervals on the rational line; $k \in \mathbb{N}$.
Parameter: k.
Question: Can the segments in Γ be covered by at most k intervals from $\mathcal{I}$?

Recall that an interval I *covers* a segment S if at least one endpoint of S lies in I. The Continuous Segment Covering problem (Cont-SC) is defined analogously to Disc-SC, with the sole distinction that the intervals can be chosen arbitrarily (i.e., there is no set $\mathcal{I}$ restricting which intervals may be chosen). For both of these problems, we also consider their *exact* versions, where we require that each segment also has an endpoint that is not contained in any interval (i.e., each segment must have precisely one "covered endpoint"); we call the associated problems Cont-Exact-SC and Disc-Exact-SC.

Finally, we denote by Disc-Equal-SC, Cont-Equal-SC, Disc-Equal-Exact-SC and Cont-Equal-Exact-SC the restrictions of Disc-SC, Cont-SC, Disc-Exact-SC and Cont-Exact-SC, respectively, to instances in which all segments in Γ have the same length. The restrictions of Cont-Equal-SC and Disc-Equal-SC to instances in which the length of the segments is at most 1 unit can be easily solved in polynomial time using a greedy approach; therefore, we will assume throughout this paper that the length of the segments in the instances of Disc-Equal-SC and Cont-Equal-SC is more than 1 unit.

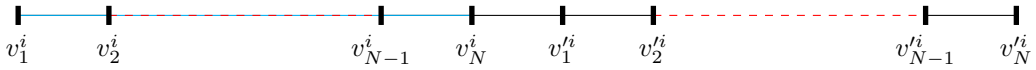


Figure 1 Illustration for the vertex-gadget construction. The segment $S_i = [v_1^i, v_N^i]$ is shown in cyan color.

▶ **Remark 1.** *There are polynomial-time FPT-reductions from* Cont-SC *and* Cont-Exact-SC *to* Disc-SC *and* Disc-Exact-SC*, respectively. The reduction from* Cont-SC *to* Disc-SC *follows from the fact that, for an instance of* Cont-SC*, we can always assume that the left endpoint of each covering unit-interval is an endpoint of a segment; if this is not the case for a covering unit-interval, we can shift it to the right until its left endpoint coincides with a segment's endpoint.*

3 Parameterized Complexity of Disc-SC and Cont-SC

In this section, we give a very high-level sketch of the W[1]-completeness proofs for Disc-SC and Cont-SC. Membership in W[1] is proved using the characterization of W[1] by Chen et al. [6]. The W[1]-hardness of Disc-SC is easier to explain as the set of covering unit-intervals is restricted; the proof can then be modified in order to lift this restriction, and obtain a W[1]-hardness proof for Cont-SC as well.

We show the W[1]-hardness of Disc-SC via an FPT-reduction from the W[1]-hard problem Multi-Colored Clique: Given a graph G with a proper k-coloring of its vertices, where each color class has cardinality N, decide if there exists a clique $Q \subseteq V(G)$ of size k [16, 9]; the parameter is k.

The reduction involves constructing three types of gadgets: vertex-selection gadgets, edge-verification gadgets, and edge-synchronization gadgets. Vertex-selection gadgets encode that k colorful vertices (i.e., no two vertices have the same color) in G are selected, and the edge-verification and edge-synchronization gadgets encode that the selected vertices form a clique. All the intervals and segments in the construction lie on the same (horizontal) line. The vertex and edge gadgets are placed far apart on the line, such that for any two gadgets, no two of their intervals overlap.

Vertex Gadgets. For each color class $C_i = \{v_1^i, \ldots, v_N^i\}, i \in [k]$, we place a sequence $\mathcal{S}^i$ of N interleaved unit intervals $[v_r^i, v_r'^i]$, $r \in [N]$, on the line, where the starting point of each interval is separated from the starting point of the next by a distance of $1/N$. We add the intervals in $\mathcal{S}^i$, for $i \in [k]$, to $\mathcal{I}$ as covering unit-intervals. For each sequence $\mathcal{S}^i$, we add the segment $S_i = [v_1^i, v_N^i]$ to Γ, which ensures that any solution must contain at least one interval from $\mathcal{S}^i$, in order to cover S_i. See Figure 1 for illustration.

Edge Gadgets. For each set of edges E_{ij}, between color classes C_i and C_j, where $i < j \in [k]$, let $m_{ij} = |E_{ij}|$. We place two interleaved sequences of unit-intervals. The construction of the two sequences is identical, and is done as follows. Set $M = m_{ij}$. The first sequence $\mathcal{S}_{ij}^1$ consists of unit-intervals $[e_r^1, e_r'^1]$, $r \in [M]$, such that $|e_r^1 e_{r+1}^1| = 1/M$, for $r \in [M-1]$; that is, the left endpoints of two consecutive intervals in this sequence are at distance $1/M$. Similarly, $\mathcal{S}_{ij}^2$ consists of unit-intervals $[e_r^2, e_r'^2]$, $r \in [M]$, such that $|e_r^2 e_{r+1}^2| = 1/M$, for $r \in [M-1]$. We place $\mathcal{S}_{ij}^1$ and $\mathcal{S}_{ij}^2$ on the line in an interleaving fashion, such that $[e_r^2, e_r'^2]$ in $\mathcal{S}_{ij}^2$ starts $1/(2M)$ units after $[e_r^1, e_r'^1]$ in $\mathcal{S}_{ij}^1$ ends, for $r \in [M]$. Put it differently, the sequence $\mathcal{S}_{ij}^2$ is shifted $1 + 1/(2M)$ units to the right from $\mathcal{S}_{ij}^1$. The reason behind this placement is that, if

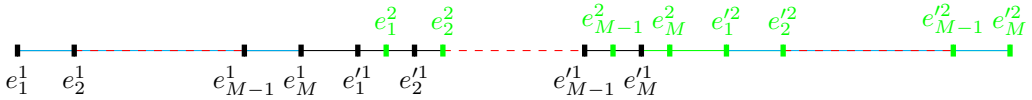


Figure 2 Illustration for the edge-gadget construction. The two segments $S_{ij}^1 = [e_1^1, e_M^1]$ and $S_{ij}^2 = [e_1'^2, e_M'^2]$ are shown in cyan color.

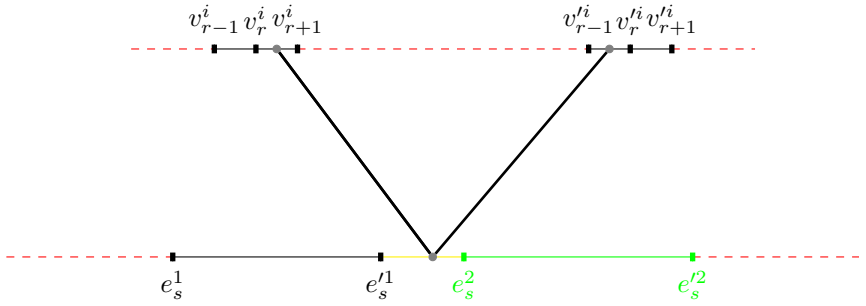


Figure 3 Illustration for the connection between a vertex-gadget and an edge-gadget that encodes vertex-edge incidency. Only the relevant portions of the gadgets are shown, and for clarity, the two gadgets are drawn on top of one another, rather than on the same line.

we assume that copies of the same interval are chosen from $\mathcal{S}_{ij}^1, \mathcal{S}_{ij}^2$, a property that will be ensured by the edge-synchronization gadgets discussed below, then the $1/(2M)$ units gap between an interval $[e_r^1, e_r'^1]$ in $\mathcal{S}_{ij}^1$ and its copy $[e_r^2, e_r'^2]$, $r \in [M]$, in $\mathcal{S}_{ij}^2$ is covered by *any* choice of an interval from $\mathcal{S}_{ij}^1$ and its copy in $\mathcal{S}_{ij}^2$, *except* the choice of $[e_r^1, e_r'^1]$ and $[e_r^2, e_r'^2]$. This property is crucial for encoding vertex-edge incidences. We add the unit-intervals of $\mathcal{S}_{ij}^1$ and $\mathcal{S}_{ij}^2$ to $\mathcal{I}$ as covering unit-intervals. We add the two segments $S_{ij}^1 = [e_1^1, e_M^1]$ and $S_{ij}^2 = [e_1'^2, e_M'^2]$ to Γ; these two segments ensure that any solution must contain at least one interval from each of $\mathcal{S}_{ij}^1$ and $\mathcal{S}_{ij}^2$. See Figure 2 for illustration.

Edge-synchronization Gadgets. For each edge-gadget consisting of two sequences $\mathcal{S}_{ij}^1$ and $\mathcal{S}_{ij}^2$ of unit-intervals, we construct a sequence of interleaved unit-intervals, $\mathcal{S}_{ij}^3$, that is constructed identically to $\mathcal{S}_{ij}^1$ and $\mathcal{S}_{ij}^2$. We add the intervals in $\mathcal{S}_{ij}^3$ to $\mathcal{I}$ as covering unit-intervals. We add the segment $S_{ij}^3 = [e_1^3, e_M^3]$ to Γ, which ensures that any solution must contain at least one interval from $\mathcal{S}_{ij}^3$. The intervals in $\mathcal{S}_{ij}^3$ will ensure that, in the desired solution, three copies of the same interval are chosen, one from each of $\mathcal{S}_{ij}^1$, $\mathcal{S}_{ij}^2$, and $\mathcal{S}_{ij}^3$.

Connecting the Gadgets. We encode vertex-edge incidences in G by adding segments between vertex-gadgets and corresponding edges-gadgets. For a vertex v_r, $r \in [N]$, in color class C_i (resp. C_j) in G, and an edge e_{ij} incident to v_r and to some vertex in color class C_j (resp. C_i), let $[e_s^1, e_s'^1]$ and $[e_s^2, e_s'^2]$ be the intervals corresponding to e_{ij} in $\mathcal{S}_{ij}^1$ and $\mathcal{S}_{ij}^2$, respectively. Do the following (see Figure 3 for an illustration): (if $r < N$) create a segment with one endpoint in the interval (v_r^i, v_{r+1}^i) in $\mathcal{S}^i$ (resp. in (v_r^j, v_{r+1}^j)), and the other in $(e_s'^1, e_s^2)$; and (if $r > 1$) create a segment with one endpoint in $(v_{r-1}'^i, v_r'^i)$ in $\mathcal{S}^i$ (resp. in $(v_{r-1}'^j, v_r'^j)$ in $\mathcal{S}^j$), and the other in $(e_s'^1, e_s^2)$.

We encode edge-synchronization for each triplet of sequences $\mathcal{S}_{ij}^1$, $\mathcal{S}_{ij}^2$, and $\mathcal{S}_{ij}^3$, corresponding to the edges in E_{ij}, where $|E_{ij}| = m_{ij}$, as follows (see Figure 4 for an illustration). For each $s \in [m_{ij} - 1]$: (i) create a segment with one endpoint in the interval (e_s^1, e_{s+1}^1) and the other in $(e_s'^3, e_{s+1}'^3)$; (ii) create a segment with one endpoint in (e_s^2, e_{s+1}^2) and the other in

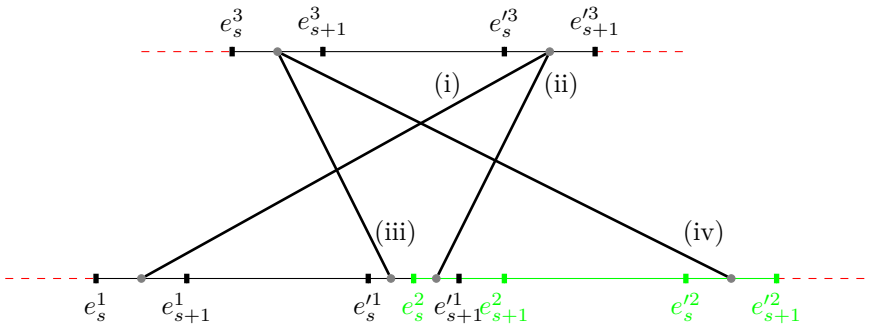


Figure 4 Illustration of the edge-synchronization gadget construction. The four types of the inter-sequence segments are indicated. For clarity, $\mathcal{S}^3_{ij}$ is drawn on top of $\mathcal{S}^1_{ij}$ and $\mathcal{S}^2_{ij}$.

(e'^3_s, e'^3_{s+1}); (iii) create a segment with one endpoint in (e^3_s, e^3_{s+1}) and the other in (e'^1_s, e'^1_{s+1}); and (iv) create a segment with one endpoint in (e^3_s, e^3_{s+1}) and the other in (e'^2_s, e'^2_{s+1}).

We can now show that (G, k) is a YES-instance of Multi-Colored Clique if and only if $(\Gamma, \mathcal{I}, k')$, where $k' = k + 3\binom{k}{2}$, is a YES-instance of Disc-SC. We conclude with:

▶ **Theorem 2.** *Disc-SC is* W[1]*-complete.*

▶ **Corollary 3.** *Cont-SC is* W[1]*-complete.*

4 NP-Completeness of the Equal Segment-Length Variants

In this section we show that all of our considered problems are NP-complete even when restricted to the case where all segments have equal length (i.e., Disc-Equal-SC, Cont-Equal-SC, Disc-Equal-Exact-SC and Cont-Equal-Exact-SC). We do so via a two-step reduction through the following intermediate problem:

Grid Segment Covering
Given: A set Γ of n vertical segments, each of length 1, with endpoints on a $q \times q$ grid with $q \leq 100 \cdot n$; $k \in \mathbb{N}$.
Parameter: k.
Question: Can the segments in Γ be covered by at most k horizontal segments of length 2?

For consistency with the terminology used in this paper (in the definition of segment covering problems under consideration), we will abuse the notation and refer to the horizontal (covering) segments of length 2 as intervals, and to the vertical segments of length 1 (to be covered) as segments. We define the Exact Grid Segment Covering analogously to Grid Segment Covering, with the sole distinction being that each segment in Γ must have precisely one endpoint covered by the solution (i.e., the set of intervals). The main technical obstacle on the way to the NP-hardness of our problems lies in showing that Grid Segment Covering and Exact Grid Segment Covering are NP-hard. The following theorem will serve as a starting point towards obtaining these results.

▶ **Theorem 4** (Theorem 5.9 of [17])**.** *Given a planar graph G of degree at most 3 with $n > 4$ vertices, there is a linear time algorithm that constructs a plane orthogonal drawing of G on an $\lfloor \frac{n}{2} \rfloor \times \lfloor \frac{n}{2} \rfloor$ grid with at most $\lfloor \frac{n}{2} \rfloor + 1$ bends[1], and with the property that there is a spanning tree of $n - 1$ straight-line edges, while all nontree edges have at most one bend.*

[1] A *bend* is the meeting point of a horizontal and a vertical line in the drawing of an edge.

4.1 Grid Segment Covering is NP-Complete

We reduce from a restriction of the NP-complete problem [14] Planar 3-Vertex Cover (i.e., Vertex Cover restricted to planar graphs of maximum degree 3). We first show that this restriction remains NP-complete:

▶ **Theorem 5.** *Planar 3-Vertex Cover is NP-hard even on instances with n vertices and a plane orthogonal drawing on an $n \times n$ grid, with no bends, even when such an embedding is provided as input.*

With Theorem 5 in hand, we can proceed to the description of the reduction strategy. Given an n-vertex instance (G, ℓ) of Planar 3-Vertex Cover, the first step is to invoke Theorem 5 to obtain a plane orthogonal drawing Ω of G on an $n \times n$ grid satisfying the property that every edge is a horizontal or a vertical line segment in this grid. Next, we refine the grid underlying Ω by a factor of 100 – more formally, we replace each cell in the grid underlying Ω with a 100×100 subgrid.

We first outline the reduction. The reduction will represent each vertex v in G with a *vertex gadget* $\alpha(v)$. This gadget consists of a set of segments placed along the border of a geometric object that is roughly centered around the position of v in Ω, and that extend in the directions of the edges incident to v. These vertex gadgets will have the following properties:

- **Vertical connections**: If v and w are adjacent vertices in G, then there is an "interface" spanning a 2×3 subgrid such that either $\alpha(v)$ is placed at the bottom right of the subgrid and $\alpha(w)$ at the top left, or vice versa. This property will be used by the *edge gadget* $\beta(uv)$.
- **Duality of choice**: There are two "optimal configurations" of intervals that allow us to cover all segments in $\alpha(v)$: one uses the minimum number of intervals required but does not help us cover the segments located in the edge gadgets connected to $\alpha(v)$, while the other requires one extra interval but also helps us cover segments in the edge gadgets connected to $\alpha(v)$. These optimal configurations cover each segment in $\alpha(v)$ only once.

The following lemma formalizes the precise properties we require from the vertex gadgets. Further intuition about the construction of the gadgets is provided in Figure 5.

▶ **Lemma 6.** *Given G, ℓ, Ω as above, in polynomial time we can construct a mapping α from the vertex set $V(G)$ that maps each $v \in V(G)$ to a gadget $\alpha(v)$. Each such gadget $\alpha(v)$ consists of a set of $|\alpha(v)|$ segments and up to 3 "link" points, each corresponding to an edge incident to v, with the following properties:*

1. *Any interval that can cover segments from $\alpha(v)$ cannot cover segments from any $\alpha(w)$ for $w \neq v$.*
2. *There exists no set of intervals of size less than $\mathsf{cost}(\alpha(v)) = \frac{|\alpha(v)|-1}{2}$ that covers all segments in $\alpha(v)$; moreover, there exists a set of intervals of size $\mathsf{cost}(\alpha(v))$ which is an exact covering of all segments in $\alpha(v)$.*
3. *There exists no set of intervals of size less than $\mathsf{cost}(\alpha(v)) + 1$ that covers all segments in $\alpha(v)$ together with at least one link point of $\alpha(v)$; moreover, there exists a set of intervals of size $\mathsf{cost}(\alpha(v)) + 1$ which is an exact covering of all segments in $\alpha(v)$ and additionally covers all link points of $\alpha(v)$.*
4. *For each edge $vw \in E(G)$ with two link points (x_v, y_v) and (x_w, y_w), it holds that either $x_v = x_w + 2$ and $y_v = y_w - 3$ or vice-versa.*

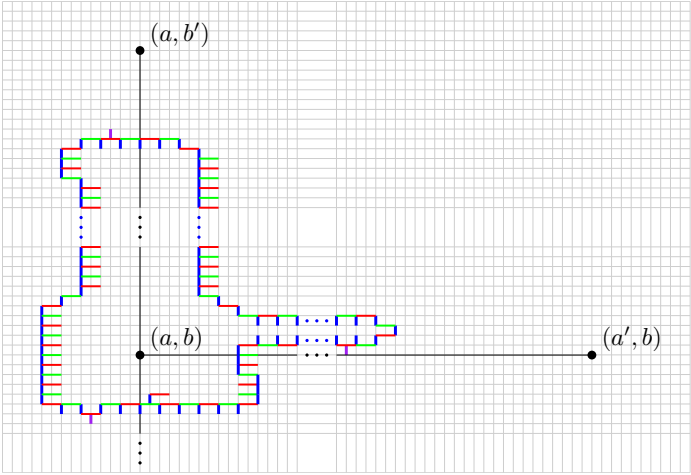


Figure 5 A vertex gadget (blue) for a vertex v located at point (a, b). Link points are marked by purple segments. The set of green intervals has size $\mathsf{cost}(\alpha(v))$ and exactly covers $\alpha(v)$. The set of red interval has size $\mathsf{cost}(\alpha(v)) + 1$ and covers $\alpha(v)$ and all the link points.

After applying Lemma 6 to construct the vertex gadgets, we will construct, for each edge vw, an edge gadget $\beta(vw)$. This will consist of the segments $[(x_v, y_v), (x_v, y_v - 1)]$, $[(x_v + 1, y_v - 1), (x_v + 1, y_v - 2)]$, and $[(x_v + 2, y_v - 2), (x_v + 2, y_v - 3)]$.

We now proceed to the NP-hardness proof.

▶ **Theorem 7.** *Grid Segment Covering and Exact Grid Segment Covering are NP-complete.*

Proof Sketch. Inclusion in NP is trivial. To show NP-hardness, we reduce from Planar 3-Vertex Cover. Given an instance (G, ℓ) of Planar 3-Vertex Cover, we apply the construction: notably, we use Theorem 4 to obtain an embedding of G on an orthogonal grid, refine this grid by a factor of 100, and replace all vertices with vertex gadgets as per Lemma 6, and all edges with edge-gadgets. Set $k = \ell + |E(G)| + \sum_{v \in V(G)} \mathsf{cost}(\alpha(v))$. Now it suffices to show that (G, ℓ) is a YES-instance if and only if (Σ, k) is a YES-instance. ◀

4.2 Reductions from the Grid Segment Covering Problem

▶ **Theorem 8.** *Disc-Equal-SC, Cont-Equal-SC, Disc-Equal-Exact-SC and Cont-Equal-Exact-SC are NP-complete.*

Proof Sketch. We sketch the reduction for Cont-Equal-SC. Let (Γ, k) be an instance of Grid Segment Covering with endpoints on $q \times q$ grid. We construct an instance (Γ', k') of Cont-Equal-SC as follows. For a segment $I \in \Gamma$ with endpoints (w, h) and $(w, h + 1)$, we add in Γ' the segment $[\frac{(q+3)h+w}{2}, \frac{(q+3)(h+1)+w}{2}]$ and we let $k' = k$. Note that if a segment $I' \in \Gamma'$ has one endpoint at $b \in \mathbb{Q}$, then there exist $h \in \mathbb{N}$ and $w \in \mathbb{Q}$ (with $0 \leq w \leq q$) such that $b = \frac{(q+3)h+w}{2}$. Hence, if a unit interval covers endpoints $b_1 = \frac{(q+3)h_1+w_1}{2}$ and $b_2 = \frac{(q+3)h_2+w_2}{2}$, then it follows that $h_1 = h_2$, because otherwise $|b_2 - b_1| \geq \frac{3}{2}$.

To conclude the proof for Cont-Equal-SC, it suffices to show that (Γ', k') is YES-instance of Cont-Equal-SC if and only if (Γ, k) is YES-instance of Grid Segment Covering. The construction is identical for Disc-Equal-Exact-SC, with the sole distinction being the use of Exact Grid Segment Covering. The other two results follow from Remark 1. ◀

5 FPT Algorithms for the Equal Segment-Length Variants

In this section, we give FPT algorithms for Cont-Equal-SC, Cont-Equal-Exact-SC, Disc-Equal-SC, and Disc-Equal-Exact-SC. Before proceeding to the technical details, we first discuss and give an overview of the FPT algorithm for Cont-Equal-SC.

Let (Γ, k) be an instance of Cont-Equal-SC. The FPT algorithm starts by computing an approximate solution, $\mathcal{S}_{apx}$, for Γ of size at most $3k$ (assuming that a solution of size k exists) whose intervals contain *all* endpoints of the segments in Γ. The algorithm then guesses (i.e., branches on all possibilities) how $\mathcal{S}_{apx}$ interacts with a solution, $\mathcal{S}_{opt}$, for Γ of size at most k. Based on this guess, the algorithm identifies endpoints of segments in $\mathcal{S}_{apx}$, called *anchors*, around which the intervals in $\mathcal{S}_{opt}$ are *anchored* (i.e., placed). The goal then becomes to assign the endpoints of the segments in Γ to the guessed anchors, where assigning an endpoint to an anchor means that the endpoint (and hence the associated segment) is covered by the interval in $\mathcal{S}_{opt}$ placed around that anchor, and then modeling the problem as an instance of 2-Sat that stipulates that, for each segment, at least one of its endpoints is covered by a unit-interval placed around an anchor. The issue, however, is that for a segment, there could be four anchors whose intervals cover its (two) endpoints, and this cannot be stipulated by size-2 clauses. This issue is resolved by exploiting the crucial property that all segments have the same length, which enables us to define a notion of domination among the anchors that could potentially cover the same segment, and guess this domination. Once the domination relations among the anchors affecting each segment are revealed, encoding the covering requirement using size-2 clauses becomes possible, as the number of anchors affecting each segment can be reduced from 4 to 2. Extra clauses are then added to the 2-Sat instance to enforce that the assignment corresponds to a proper placement of k unit-length covering intervals that cover the segments in Γ. We proceed to the details.

Let (Γ, k) be an instance of Cont-Equal-SC. We start with the following simple result:

▶ **Fact 9.** *In $\mathcal{O}(|\Gamma| \log |\Gamma|)$ time, we can compute a solution $\mathcal{S}_{apx}$ to Γ that is within ratio 3 from an optimal solution and that contains all endpoints of the segments in Γ.*

Proof. For a unit-length interval I, define the *left dual* (resp. *right dual*) of I, denoted, $\overline{I_L}$ (resp. $\overline{I_R}$), to be the interval that is the translation of I (along the horizontal line) to the left (resp. to the right) by a vector whose length is equal to the length of the segments in Γ. Observe that the set of segments in Γ whose right (resp. left) endpoints are covered by a unit-length interval I is the same set of segments whose left (resp. right) endpoints are covered by $\overline{I_L}$ (resp. $\overline{I_R}$).

The approximation algorithm, denoted APX-ALGO, finds a set of unit-length intervals of minimum cardinality that covers the endpoints of all the segments in Γ; the problem of covering a set of N points on a line by the minimum number of unit-length intervals is known to be solvable in $\mathcal{O}(N \lg N)$ time by a greedy approach (e.g., see problem 16.2-5 in [8]).

Consider now an optimal solution for Γ, and for each interval I in the optimal solution, add both its left and right duals $\overline{I_L}$ and $\overline{I_R}$. We obtain a solution that contains all segment endpoints and whose cardinality is at most thrice that of the optimal solution. Since APX-ALGO produces an optimal solution for covering the endpoints of all segments in Γ, the result follows. ◀

Based on the above, if $|\mathcal{S}_{apx}| > 3k$, the instance (Γ, k) is a no-instance of Cont-Equal-SC. Assume henceforth that $|\mathcal{S}_{apx}| \leq 3k$. Every endpoint of a segment in Γ is contained in an interval of $\mathcal{S}_{apx}$. Without loss of generality, we can assume that the intervals in $\mathcal{S}_{apx}$ are pairwise disjoint, and that each starts at an endpoint of a segment in Γ (see also Remark 1).

If not, we can process the intervals in $\mathcal{S}_{apx}$ to ensure this property, while maintaining the property that every endpoint of a segment in Γ is contained in an interval of $\mathcal{S}_{apx}$. To do so, we repeatedly pick the leftmost two overlapping intervals in $\mathcal{S}_{apx}$, say I_r, I_s where I_r starts before I_s. We shift I_s to the right until it no longer overlaps with I_r, while starting at an endpoint in Γ and retaining the set of segment endpoints contained in $I_r \cup I_s$. If at some point this shifting results in an interval that is devoid of segment endpoints, we remove this interval from $\mathcal{S}_{apx}$. Clearly, at the end of this process, the set $\mathcal{S}_{apx}$ consists of pairwise-disjoint intervals that contain all endpoints of the segments in Γ, each of whose intervals starts at an endpoint of a segment in Γ, and satisfying $|\mathcal{S}_{apx}| \leq 3k$.

Let $\mathcal{S}_{opt}$ be an optimal solution for Γ, and assume that $|\mathcal{S}_{opt}| \leq k$. W.l.o.g., we will assume that $\mathcal{S}_{opt}$ is chosen so as to maximize the number of intervals that are common to both $\mathcal{S}_{opt}$ and $\mathcal{S}_{apx}$ (i.e., maximize $|\mathcal{S}_{opt} \cap \mathcal{S}_{apx}|$).

▶ **Definition 10.** An endpoint $\mathfrak{a}$ of an interval in $\mathcal{S}_{apx}$ is called an *anchor* if there is an interval I in $\mathcal{S}_{opt}$ that contains $\mathfrak{a}$, in which case we say that I *induces* $\mathfrak{a}$.

The FPT-algorithm for Cont-Equal-SC performs the following steps:

Step (1). Guessing the Anchors: The FPT-algorithm starts by guessing how $\mathcal{S}_{opt}$ interacts with $\mathcal{S}_{apx}$. First, it guesses the number k' of intervals that are common to both $\mathcal{S}_{apx}$ and $\mathcal{S}_{opt}$ (i.e., $|\mathcal{S}_{apx} \cap \mathcal{S}_{opt}|$). Then, the algorithm guesses the k' common intervals, removes them from $\mathcal{S}_{apx}$, and updates Γ and the parameter k accordingly (by removing from Γ all segments covered by the k' intervals, and setting $k = k - k'$). By the same arguments made above about $\mathcal{S}_{apx}$, we can assume from now on that the intervals in $\mathcal{S}_{opt}$ are disjoint, and that each starts at a segment endpoint. Every interval $I \in \mathcal{S}_{opt}$ intersects two consecutive intervals in $\mathcal{S}_{apx}$. Otherwise, if I intersects only one interval $I' \in \mathcal{S}_{apx}$, since each interval in $\mathcal{S}_{opt}$ starts at a segment endpoint, I would intersect I' only at the left endpoint of I, and all segment endpoints in I must also be in I'. This contradicts our choice of $\mathcal{S}_{opt}$ as an optimal solution that maximizes $|\mathcal{S}_{apx} \cap \mathcal{S}_{opt}|$ (since I could be shifted left to obtain an optimal solution containing I'). Hence, if I contains the left (resp. right) endpoint of I', then it must contain the right (resp. left) endpoint of the predecessor (resp. successor) interval of I' in $\mathcal{S}_{apx}$. Next, for each endpoint of an interval in $\mathcal{S}_{apx}$, the algorithm guesses whether it is an anchor. Let Υ be the set of guessed anchors.

Step (2). Restructuring the Anchors: From Step (1), if an anchor $\mathfrak{a} \in \Upsilon$ is the right (resp. left) endpoint of an interval $I' \in \mathcal{S}_{apx}$, then the interval $I \in \mathcal{S}_{opt}$ that induces $\mathfrak{a}$ intersects the successor (resp. predecessor) of I' in $\mathcal{S}_{apx}$, and hence, the left (resp. right) endpoint of the successor (resp. predecessor) of I' must be an anchor induced by I as well. If after Step (1) Υ does not conform to the above, then we can reject the guess, as there will be another guess that satisfies the above property. Based on this property, we will remove from Υ the anchors that are right endpoints of intervals in $\mathcal{S}_{apx}$. After removing these anchors, each interval in $\mathcal{S}_{opt}$ induces *exactly* one anchor in Υ that is the left endpoint of an interval in $\mathcal{S}_{apx}$. Since the intervals in $\mathcal{S}_{apx}$ are pairwise disjoint, any two anchors in Υ are more than 1-unit apart. Consequently, if $|\Upsilon| > k$, then we can reject the guess in Step (1), as no solution of size at most k realizing the guess exists.

Step (3). Domination among Anchors: Let $\mathfrak{a}, \mathfrak{b}$ be two anchors, and let $\mathcal{S} \subseteq \Gamma$. We say that $\mathfrak{a}$ *dominates* $\mathfrak{b}$ w.r.t. $\mathcal{S}$, written as $\mathfrak{a} \succeq_{\mathcal{S}} \mathfrak{b}$ or $\mathfrak{b} \preceq_{\mathcal{S}} \mathfrak{a}$, if the set of segments in $\mathcal{S}$ covered by (the interval in $\mathcal{S}_{opt}$ inducing) $\mathfrak{a}$ is a superset of the set of segments in $\mathcal{S}$ covered by (the

interval in $\mathcal{S}_{opt}$ inducing) $\mathfrak{b}$. For convenience, we will define the notion of an *empty anchor*, denoted $\otimes$; the set of segments covered by the empty anchor is the empty set ϕ, and hence, every anchor dominates $\otimes$. We will use the notion of domination, in conjunction with a guessing process, to reduce the instance (Γ, k), resulting from Steps (1) and (2) above, to an instance of 2-SAT, which can then be solved in polynomial time. To do so, we consider the intervals in $\mathcal{S}_{apx}$ from left to right, and construct an instance $\mathcal{F}$ of 2-SAT. We initialize $\mathcal{F}$ to the empty set, and we will add clauses to $\mathcal{F}$ as follows.

Let $I \in \mathcal{S}_{apx}$ be the interval currently under consideration (when scanning the intervals in $\mathcal{S}_{apx}$ from left to right), and let $\mathcal{S}$ be the set of segments whose left endpoints are on I. (We are not concerned at this point about the set of segments whose right endpoint are on I since those have been considered earlier in the process.) Observe that, since all segments in Γ have the same length, and all covering intervals have the same length, the right endpoints of the segments in $\mathcal{S}$ fall either on one or on two intervals in $\mathcal{S}_{apx}$. (Otherwise, there would be two segments whose left endpoints lie on I, and hence, of distance at most 1 unit, but whose right endpoints are more than one unit apart.) This property, which again stems from the fact that all segments in Γ have the same length, is *crucial*, and is the key idea behind the FPT algorithm, as will be seen below.

We treat the more complex case in which the right endpoints of the segments in $\mathcal{S}$ fall on two intervals I_1, I_2 in $\mathcal{S}_{apx}$, where I_2 is the successor of I_1 in $\mathcal{S}_{apx}$; the case where they fall on one interval is simpler, and is a subcase of the treated case, as we explain below. Partition $\mathcal{S}$ into $\mathcal{S}_1, \mathcal{S}_2$, where $\mathcal{S}_1$ consists of those segments in $\mathcal{S}$ whose right endpoints are on I_1, and $\mathcal{S}_2$ of those whose right endpoints are on I_2. Let $\mathfrak{a}$ be the left endpoint of I if it is an anchor, and $\mathfrak{a} = \otimes$ otherwise; let $\mathfrak{b}$ be the left endpoint of the successor of I in $\mathcal{S}_{apx}$ if its an anchor, and $\mathfrak{b} = \otimes$ otherwise; let $\mathfrak{u}$ be the left endpoint of I_1 if it is an anchor, and $\mathfrak{u} = \otimes$ otherwise; let $\mathfrak{s}$ be the left endpoint of I_2 if it is an anchor, and $\mathfrak{s} = \otimes$ otherwise; and let $\mathfrak{t}$ be the left endpoint of the successor of I_2 in $\mathcal{S}_{apx}$ if it is an anchor, and $\mathfrak{t} = \otimes$ otherwise. Observe that, since each of the anchors $\mathfrak{a}$ and $\mathfrak{u}$ covers a prefix (possibly empty) of the sequence of segments in $\mathcal{S}_1$ when ordered from left to right, one of the two anchors must dominate the other w.r.t. $\mathcal{S}_1$. Similarly, each of $\mathfrak{b}$ and $\mathfrak{s}$ covers a suffix (possibly empty) of the sequence of segments in $\mathcal{S}_1$, and hence one must dominate the other w.r.t. $\mathcal{S}_1$. With respect to $\mathcal{S}_2$, one of $\mathfrak{a}$ and $\mathfrak{s}$ must dominate the other, and one of $\mathfrak{b}$ and $\mathfrak{t}$ must dominate the other. Therefore, (i) either $\mathfrak{a} \preceq_{\mathcal{S}_1} \mathfrak{u}$ or $\mathfrak{u} \preceq_{\mathcal{S}_1} \mathfrak{a}$ and (ii) either $\mathfrak{b} \preceq_{\mathcal{S}_1} \mathfrak{s}$ or $\mathfrak{s} \preceq_{\mathcal{S}_1} \mathfrak{b}$; and w.r.t. the segments in $\mathcal{S}_2$, we have (iii) either $\mathfrak{a} \preceq_{\mathcal{S}_2} \mathfrak{s}$ or $\mathfrak{s} \preceq_{\mathcal{S}_2} \mathfrak{a}$, and (iv) either $\mathfrak{b} \preceq_{\mathcal{S}_2} \mathfrak{t}$ or $\mathfrak{t} \preceq_{\mathcal{S}_2} \mathfrak{b}$.

The algorithm now guesses, for each of Cases (i) – (iv) above, which anchor dominates the other. This guessing results in four cases w.r.t. each of $\mathcal{S}_1$ and $\mathcal{S}_2$ that are described below, and hence, results in sixteen cases overall. If the right endpoints of the segments in $\mathcal{S}$ fall on one interval I_1, then the guessing results only in the four cases w.r.t. $\mathcal{S}_1$ distinguished below (and anchor $\mathfrak{t}$ would not be needed). We will create Boolean variables corresponding to endpoints of segments in Γ. A Boolean variable of the form $x_{\mathfrak{h}}$, where x is an endpoint of a segment $S \in \Gamma$ and $\mathfrak{h}$ is an anchor, is true if and only if point x (and hence S) is covered by the interval inducing $\mathfrak{h}$. We will then form an instance of 2-SAT that encodes the instance (Γ, k) of CONT-EQUAL-SC under the assumed guess. For simplicity of the presentation, we make the following assumptions. If a Boolean variable $x_{\mathfrak{h}}$, associated with anchor $\mathfrak{h}$, is such that either $\mathfrak{h} = \otimes$, or the distance between x and $\mathfrak{h}$ is more than 1 unit (i.e., more than the length of a unit-length covering interval), then we set/fix the value of $x_{\mathfrak{h}}$ to FALSE. We start by associating with every endpoint x of a segment in Γ two Boolean variables as follows. Let $\mathfrak{h}$ and $\mathfrak{h}'$ be the two anchors directly to the left and right, respectively, of x. We associate

Figure 6 Illustration for **Cases 1-4** w.r.t. $\mathcal{S}_1$ and $\mathcal{S}_2$. The two red circles designate the endpoints x, y of a segment $S \in \mathcal{S}_1$, and the green circles designate the endpoints x', y' of a segment $S' \in \mathcal{S}_2$, where x is to the left of y and x' is to the left of y'. For clarity, only the endpoints of S, S' are shown. If the guess w.r.t. $\mathcal{S}_1$ is that $\mathfrak{a} \succeq \mathfrak{u}$ and $\mathfrak{s} \succeq \mathfrak{b}$ and w.r.t. $\mathcal{S}_2$ is that $\mathfrak{s} \succeq \mathfrak{a}$ and $\mathfrak{b} \succeq \mathfrak{t}$ then clauses $\{x_\mathfrak{a} \vee y_\mathfrak{s}\}$ and $\{x'_\mathfrak{b} \vee y'_\mathfrak{s}\}$ are added to $\mathcal{F}$.

the two Boolean variables $x_\mathfrak{h}$ and $x_{\mathfrak{h}'}$ with x. (Note that $\mathfrak{h}$ or $\mathfrak{h}'$ could be $\otimes$, or of distance more than 1 unit from x, and in which case the corresponding Boolean variable would be set to FALSE.) The four cases distinguished w.r.t. $\mathcal{S}_1$ are (see Figure 6 for illustration):

Case 1: $\mathfrak{a} \succeq \mathfrak{u}$ and $\mathfrak{b} \succeq \mathfrak{s}$. For each endpoint x on I such that x is the left endpoint of a segment in $\mathcal{S}_1$, add the clause $\{x_\mathfrak{a} \vee x_\mathfrak{b}\}$ to $\mathcal{F}$.

Case 2: $\mathfrak{a} \succeq \mathfrak{u}$ and $\mathfrak{s} \succeq \mathfrak{b}$. For each endpoint x on I such that x is the left endpoint of a segment S in $\mathcal{S}_1$, let y be the right endpoint of S. Add the clause $\{x_\mathfrak{a} \vee y_\mathfrak{s}\}$ to $\mathcal{F}$.

Case 3: $\mathfrak{u} \succeq \mathfrak{a}$ and $\mathfrak{b} \succeq \mathfrak{s}$. For each endpoint x on I such that x is the left endpoint of a segment S in $\mathcal{S}_1$, let y be the right endpoint of S. Add the clause $\{x_\mathfrak{b} \vee y_\mathfrak{u}\}$ to $\mathcal{F}$.

Case 4: $\mathfrak{u} \succeq \mathfrak{a}$ and $\mathfrak{s} \succeq \mathfrak{b}$. For each endpoint x on I such that x is the left endpoint of a segment S in $\mathcal{S}_1$, let y be the right endpoint of S. Add the clause $\{y_\mathfrak{u} \vee y_\mathfrak{s}\}$ to $\mathcal{F}$.

Similarly, we can distinguish four cases w.r.t. $\mathcal{S}_2$, based on the two domination relations in (iii) and (iv) discussed earlier, and add clauses to $\mathcal{F}$ accordingly.

After processing the intervals in $\mathcal{S}_{apx}$, guessing domination, associating Boolean variables, and adding clauses to $\mathcal{F}$, we add to $\mathcal{F}$ the following "enforcement" clauses. For every anchor $\mathfrak{a}$, and every two variables $x_\mathfrak{a}, z_\mathfrak{a}$, corresponding to segment endpoints x, z, respectively, associated with $\mathfrak{a}$:

(E1) If x and z are on the right (resp. left) side of $\mathfrak{a}$, and if z (resp. x) is to the right (resp. left) of x, add $\{\overline{z}_\mathfrak{a} \vee x_\mathfrak{a}\}$ (resp. $\{\overline{x}_\mathfrak{a} \vee z_\mathfrak{a}\}$) to $\mathcal{F}$; and (E2) if x and z are on opposite sides of $\mathfrak{a}$ and the distance between them is more than 1 unit, add $\{\overline{x}_\mathfrak{a} \vee \overline{z}_\mathfrak{a}\}$ to $\mathcal{F}$. The enforcement clauses ensure that any satisfying assignment to $\mathcal{F}$ corresponds to an assignment of segment endpoints to anchors satisfying: (1) All endpoints assigned to the same anchor can be covered by a unit interval (E2); and (2) if an endpoint of a segment S that is assigned to an anchor $\mathfrak{h}$ is covered by the interval $I_\mathfrak{h}$ inducing $\mathfrak{h}$, then any segment endpoint assigned to $\mathfrak{h}$ that is between that endpoint of S and $\mathfrak{h}$ is also covered by $I_\mathfrak{h}$ (E1).

The FPT algorithm accepts if any of the guesses it makes leads to a formula $\mathcal{F}$ that is a YES-instance of 2-SAT, and rejects otherwise. We obtain the following result:

▶ **Theorem 11.** *CONT-EQUAL-SC can be solved in time* $\mathcal{O}((2^4 \cdot 3 \cdot e^{5/3})^k \cdot n \log n) = \mathcal{O}(2^{8k} \cdot n \log n)$*, where* $n = |\Gamma|$*, and hence is FPT.*

Proof. We first argue the correctness of the algorithm. The instance (Γ, k) is a YES-instance of CONT-EQUAL-SC if and only if there exists an optimal solution $\mathcal{S}_{opt}$ for Γ containing at most k intervals. The algorithm guesses in Step (1) the intervals in $\mathcal{S}_{apx}$ that are in $\mathcal{S}_{opt}$, and updates (Γ, k) accordingly. As explained before, we may assume that the intervals in the optimal solution sought (if it exists), $\mathcal{S}_{opt}$, are pairwise disjoint, start at segment endpoints, and that each interval in $\mathcal{S}_{opt}$ intersects two consecutive intervals in $\mathcal{S}_{apx}$. The algorithm then guesses which endpoints of intervals in $\mathcal{S}_{apx}$ are anchors (w.r.t. $\mathcal{S}_{opt}$).

The algorithm in Step (2) removes anchors from the set Υ of anchors, so that each anchor is the left endpoint of its interval in $\mathcal{S}_{apx}$. Note that after this restructuring, each interval in the sought solution $\mathcal{S}_{opt}$ contains exactly one anchor in Υ. Moreover, any two anchors are more than 1 unit apart, and hence, if $|\Upsilon| > k$, then the algorithm can safely reject the current guess, as it does in Step (2), since no solution $\mathcal{S}_{opt}$ of size k conforming to the current guess exists. It is clear that a solution $\mathcal{S}_{opt}$ to (Γ, k) exists if and only if there exists a guess of a set Υ of anchors satisfying the above conditions.

The algorithm then proceeds to determining how the intervals in the solution sought should be "anchored" (or placed) around their anchors in order to cover all segments in Γ. To do so, the algorithm considers the intervals in $\mathcal{S}_{apx}$ from left to right. For an interval I under consideration, the set of segments $\mathcal{S}$ whose left endpoints lie on I must be covered by $\mathcal{S}_{opt}$. The right endpoints of the segments in $\mathcal{S}$ lie on at most two intervals of $\mathcal{S}_{apx}$; we argue the more complicated case in which these endpoints lie on exactly two intervals, I_1, I_2, where I_2 is the successor of I_1 in $\mathcal{S}_{apx}$, as this case subsumes the other one. The set of segments $\mathcal{S}$ can be partitioned into $\mathcal{S}_1$ and $\mathcal{S}_2$, as explained in Step (3) of the algorithm, depending on which interval in I_1, I_2 the right endpoint of the segment in $\mathcal{S}$ lies on. Let the anchors $\mathfrak{a}, \mathfrak{b}, \mathfrak{u}, \mathfrak{s}, \mathfrak{t}$ be as defined in Step (3) of the algorithm. Observe that, since each of the anchors $\mathfrak{a}$ and $\mathfrak{u}$ covers a prefix (possibly empty) of the sequence of segments in $\mathcal{S}_1$ when ordered from left to right, one of the two anchors must dominate the other w.r.t. $\mathcal{S}_1$. Similarly, each of the anchors $\mathfrak{b}$ and $\mathfrak{s}$ covers a suffix (possibly empty) of the sequence of segments in $\mathcal{S}_1$ (when ordered from left to right), and hence one must dominate the other w.r.t. $\mathcal{S}_1$. With respect to $\mathcal{S}_2$, one of the two anchors $\mathfrak{a}$ and $\mathfrak{s}$ must dominate the other, and one of $\mathfrak{b}$ and $\mathfrak{t}$ must dominate the other. The algorithm guesses each of these domination relations, for each interval $I \in \mathcal{S}_{apx}$ and set of segments $\mathcal{S}$ whose left endpoints lie on I. If the algorithm guesses correctly, then for each segment in Γ, it assigns each of its two endpoints to an anchor such that the segment is covered by an interval inducing one of the anchors assigned to its endpoints. After guessing the domination relations among the anchors, the algorithm creates an instance $\mathcal{F}$ of 2-Sat that, for each segment $S \in \Gamma$ and each endpoint of $\mathcal{S}$, associates a Boolean variable whose value is true if and only if the endpoint of the segment is covered by the interval inducing the anchor assigned to the endpoint (based on the domination relation). The algorithm then adds enforcement clauses to $\mathcal{F}$ ensuring (the converse) that a satisfying assignment to $\mathcal{F}$ corresponds to an assignment of segment endpoints to anchors satisfying: (1) All endpoints assigned to the same anchor can be covered by a unit interval (E2); and (2) if an endpoint x of a segment that is assigned to an anchor $\mathfrak{h}$ is covered by the interval $I_{\mathfrak{h}}$ inducing $\mathfrak{h}$, then any segment endpoint assigned to $\mathfrak{h}$ that is between x and $\mathfrak{h}$ is also covered by $I_{\mathfrak{h}}$ (E1).

Given the above, it is not difficult to verify that the instance (Γ, k) is a YES-instance of Cont-Equal-SC if and only if there is a guess for the algorithm that yields a YES-instance $\mathcal{F}$ of 2-Sat, and hence, that the algorithm is correct. Next we analyze the running time of the algorithm.

First, observe that computing $\mathcal{S}_{apx}$ can be done in $\mathcal{O}(n \lg n)$ time, as this can be done by sorting the endpoints in Γ. Moreover, all processing steps for the intervals in $\mathcal{S}_{apx}$ and segments Γ can be carried out in time $\mathcal{O}(n \lg n)$. Therefore, we only need to analyze the size of the search tree needed to simulate the guesses performed by the algorithm. The algorithm performs guessing only in Step (1) and Step (3).

In Step (1), the algorithm guesses a number $k' \in \{0, \ldots, k\}$, and then it guesses a subset of k' intervals in $\mathcal{S}_{apx}$. The total number of branches needed to simulate these guesses is at most $\sum_{k'=0}^{k} \binom{3k}{k'}$. For each guess of k' intervals, the algorithm removes these intervals

from $\mathcal{S}_{apx}$, updates Γ and sets $k = k - k'$. Removing k' intervals from $\mathcal{S}_{apx}$ leaves $\mathcal{S}_{apx}$ with at most $3k - k'$ intervals. The algorithm then guesses which endpoints of the (remaining) intervals in $\mathcal{S}_{apx}$ are anchors. Since we can assume that the anchors in question are left endpoints of their intervals, guessing the anchors can be simulated by choosing a subset of $(k - k')$ endpoints, out of the at most $(3k - k')$ left endpoints of the (remaining) intervals in $\mathcal{S}_{apx}$. Therefore, the total number of branches needed to simulate all guesses in Step (1) is at most $\sum_{k'=0}^{k} \binom{3k}{k'} \cdot \binom{3k-k'}{k-k'}$.

In Step (3), the algorithm guesses the domination relations among anchors. At this point, the number of anchors (by Step (2)) is at most $k - k'$. In the guessing, each guess made is w.r.t. an interval $I \in \mathcal{S}_{apx}$ and the two anchors $\mathfrak{a}$ and $\mathfrak{b}$, as defined in Step (3). We will charge each guess to the two anchors that play the roles of $\mathfrak{a}$ and $\mathfrak{b}$ w.r.t. some interval $I \in \mathcal{S}_{apx}$. There are four guesses made, resulting in sixteen cases, and each of $\mathfrak{a}$ and $\mathfrak{b}$ is involved in the same number of guesses. Therefore, eight cases need to be distinguished with respect to each of $\mathfrak{a}$ and $\mathfrak{b}$. Since each of the at most k' anchors can play the role of $\mathfrak{a}$ once and of $\mathfrak{b}$ once, over all intervals in $\mathcal{S}_{apx}$ (note that a domination relation involving an empty anchor is determined, not guessed), it follows that the total number of cases that each anchor can be involved in is sixteen, which results in a total number of branches of at most $2^{4k'}$ over all anchors.

It follows that the size of the search tree needed to simulate all the guesses performed by the algorithm is $\sum_{k'=0}^{k} \binom{3k}{k'} \cdot \binom{3k-k'}{k-k'} \cdot 2^{4k'}$. Next, we upper bound this expression.

Applying the well-known upper bound $\binom{r}{s} \leq (e \cdot r/s)^s$ on the binomial coefficient in both binomial terms $\binom{3k}{k'}$ and $\binom{3k-k'}{k-k'}$, where e is the base of the natural logarithm, and simplifying, we obtain:

$$\sum_{k'=0}^{k} \binom{3k}{k'} \cdot \binom{3k-k'}{k-k'} \cdot 2^{4k'} \tag{1}$$

$$\leq (3e)^k \cdot 2^{4k} + (3e)^k \cdot \sum_{k'=0}^{k-1} 2^{4k'} \cdot (k/k')^{k'} \cdot ((k-k'/3)/(k-k'))^{k-k'} \tag{2}$$

$$\leq (3e)^k \cdot 2^{4k} + (3e)^k \cdot \sum_{k'=0}^{k-1} 2^{4k'} \cdot (k/k')^{k'} \cdot e^{2k'/3} \tag{3}$$

$$= (3e)^k \cdot 2^{4k} + (3e)^k \cdot \sum_{k'=0}^{k-1} ((2^4 \cdot e^{2/3} \cdot k)/k')^{k'} \tag{4}$$

$$\leq (3e)^k \cdot 2^{4k} + (3e)^k \cdot \mathcal{O}((2^4 \cdot e^{2/3})^k) = \mathcal{O}((2^4 \cdot 3 \cdot e^{5/3})^k). \tag{5}$$

In Inequality (2), we split the summation – a minor technicality – in order to avoid a denominator of 0 in the term $((k-k'/3)/(k-k'))^{k-k'}$, resulting from approximating $\binom{3k-k'}{k-k'}$, when $k' = k$. We obtain Inequality (3) from Inequality (2), by upper bounding the term $((k-k'/3)/(k-k'))^{k-k'}$ by $e^{2k'/3}$. This is done by rewriting the term $((k-k'/3)/(k-k'))^{k-k'}$ in the form $(1+1/x)^x$, and using the well-known inequality $(1+1/x)^x \leq e$, for all $x > 0$. We obtain Inequality (5) from Inequality (4) by showing that the function $((2^4 \cdot e^{2/3} \cdot k)/k')^{k'}$ is increasing in k', which then can be used to upper bound the summation $\sum_{k'=0}^{k}((2^4 \cdot e^{2/3} \cdot k)/k')^{k'}$ by $\mathcal{O}((2^4 \cdot 3 \cdot e^{2/3})^k)$. ◀

We can obtain FPT algorithms for Disc-Equal-SC, Disc-Equal-Exact-SC, and Cont-Equal-Exact-SC as well. The ideas leading to the FPT algorithm for Disc-Equal-SC are the same as those for Cont-Equal-SC, albeit the technical details become more complicated and the running time is significantly worse. The complications are mainly due to

the stipulation that the covering intervals cannot be arbitrarily chosen, and must be selected from the set $\mathcal{I}$, given as input. This makes it harder to obtain an approximate solution with the desired properties – and leads to a worse approximation ratio, and to restructure the anchors. In particular, we can no longer make the simplifying assumptions about the structure of the intervals in $\mathcal{S}_{apx}$ and $\mathcal{S}_{opt}$. The FPT results for Disc-Equal-Exact-SC and Cont-Equal-Exact-SC are byproducts of that for Disc-Equal-SC.

▶ **Theorem 12.** *Disc-Equal-SC, Disc-Equal-Exact-SC, Cont-Equal-Exact-SC can be solved in time* $\mathcal{O}(2^{30k} \cdot (|\Gamma| \lg |\Gamma| + |\mathcal{I}| \lg |\mathcal{I}|))$*, and hence are FPT.*

6 Conclusion

In this paper, we considered several variants of segment covering by unit intervals. We established the NP-hardness of the restrictions of these problems to instances in which all segments have the same length. In addition to its importance per se, this result strengthens and implies a number of NP-hardness results in the literature. We also presented parameterized complexity results for several of these problems, showing their W[1]-hardness for the general case, and presenting FPT algorithms for their restrictions to instances in which all segments have the same length. Our work gives rise to two open questions:

1. What is the parameterized complexity of Cont-Exact-SC and Disc-Exact-SC?
2. What is the complexity of the restriction of Cont-Exact-SC and Disc-Exact-SC to instances in which all segments have length at most 1 unit?

References

1 A. Acharyya, S. Nandy, S. Pandit, and S. Roy. Covering segments with unit squares. *Computational Geometry: Theory and Applications*, 79:1–13, 2019.

2 E. Arkin, A. Banik, P. Carmi, G. Citovsky, M. Katz, J. Mitchell, and M. Simakov. Choice is hard. In *ISAAC*, volume 9472 of *Lecture Notes in Computer Science*, pages 318–328. Springer, 2015.

3 E. Arkin, A. Banik, P. Carmi, G. Citovsky, M. Katz, J. Mitchell, and M. Simakov. Conflict-free covering. In *CCCG*, 2015.

4 E. Arkin, A. Banik, P. Carmi, G. Citovsky, M. Katz, J. Mitchell, and M. Simakov. Selecting and covering colored points. *Discrete Applied Mathematics*, 250:75–86, 2018.

5 P. Bonsma, T. Epping, and W. Hochstättler. Complexity results on restricted instances of a paint shop problem for words. *Discrete Applied Mathematics*, 154(9):1335–1343, 2006.

6 Y. Chen, J. Flum, and M. Grohe. Machine-based methods in parameterized complexity theory. *Theoretical Computer Science*, 339(2-3):167–199, 2005.

7 M. Claverol, E. Khramtcova, E. Papadopoulou, M. Saumell, and C. Seara. Stabbing circles for sets of segments in the plane. *Algorithmica*, 80(3):849–884, 2018.

8 T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

9 M. Cygan, F. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

10 R. Diestel. *Graph Theory, 4th Edition*. Springer, 2012.

11 R. Downey and M. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, Berlin, Heidelberg, 2013.

12 T. Erlebach and E. J. van Leeuwen. Approximating geometric coverage problems. In *SODA*, pages 1267–1276. SIAM, 2008.

13 T. Erlebach and E.J. van Leeuwen. PTAS for weighted set cover on unit squares. In *APPROX-RANDOM*, volume 6302 of *Lecture Notes in Computer Science*, pages 166–177. Springer, 2010.

14 M. R. Garey and D. S. Johnson. The rectilinear steiner tree problem in NP complete. *SIAM Journal of Applied Mathematics*, 32:826–834, 1977.

15 A. Gupta, S. Kale, V. Nagarajan, R. Saket, and B. Schieber. The approximability of the binary paintshop problem. In *APPROX-RANDOM*, volume 8096 of *Lecture Notes in Computer Science*, pages 205–217. Springer, 2013.

16 S. Hartung and R. Niedermeier. Incremental list coloring of graphs, parameterized by conservation. *Theoretical Computer Science*, 494:86–98, 2013.

17 G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16(1):4–32, 1996.

18 J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

19 K. Kobylkin. Stabbing line segments with disks: Complexity and approximation algorithms. In *AIST*, volume 10716 of *Lecture Notes in Computer Science*, pages 356–367. Springer, 2017.

20 S. Langerman and P. Morin. Covering things with things. *Discrete & Computational Geometry*, 33(4):717–729, 2005.

21 R. Madireddy and A. Mudgal. Stabbing line segments with disks and related problems. In *CCCG*, pages 201–207, 2016.

22 J. Wang, W. Li, and J. Chen. A parameterized algorithm for the hyperplane-cover problem. *Theoretical Computer Science*, 411(44-46):4005–4009, 2010.

Decidability and Periodicity of Low Complexity Tilings

Jarkko Kari
University of Turku, Finland
http://users.utu.fi/jkari/
jkari@utu.fi

Etienne Moutot
University of Turku, Finland
Université de Lyon – ENS de Lyon – UCBL – CNRS – LIP, France
http://perso.ens-lyon.fr/etienne.moutot/
etienne.moutot@ens-lyon.org

Abstract

In this paper we study low-complexity colorings (or tilings) of the two-dimensional grid $\mathbb{Z}^2$. A coloring is said to be of low complexity with respect to a rectangle if there exists $m, n \in \mathbb{N}$ such that there are no more than mn different rectangular $m \times n$ patterns in it. Open since it was stated in 1997, Nivat's conjecture states that such a coloring is necessarily periodic. Suppose we are given at most nm rectangular patterns of size $n \times m$. If Nivat's conjecture is true, one can only build periodic colorings out of these patterns – meaning that if the $m \times n$ rectangular patterns of the coloring are among these mn patterns, it must be periodic. The main contribution of this paper proves that there exists at least one periodic coloring build from these patterns. We use this result to investigate the tiling problem, also known as the domino problem, which is well known to be undecidable in its full generality. However, we show that it is decidable in the low-complexity setting. Finally, we use our result to show that Nivat's conjecture holds for uniformly recurrent configurations. The results also extend to other convex shapes in place of the rectangle.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Theory of computation → Computability

Keywords and phrases Nivat's conjecture, domino problem, decidability, low pattern complexity, 2D subshifts, symbolic dynamics

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.14

Funding *Jarkko Kari*: Supported by the Academy of Finland grant 296018.
Etienne Moutot: Supported by ANR project CoCoGro (ANR-16-CE40-0005).

1 Introduction

The tiling problem, also known as the domino problem, asks whether the two-dimensional grid $\mathbb{Z}^2$ can be colored in a way that avoids a given finite collection of forbidden local patterns. The problem is undecidable in its full generality. The undecidability relies on the fact that there are *aperiodic* systems of forbidden patterns that enforce any valid coloring to be non-periodic [1].

In this paper we consider the low complexity setup where the number of allowed local patterns is small. More precisely, suppose we are given at most nm legal rectangular patterns of size $n \times m$, and we want to know whether there exists a coloring of $\mathbb{Z}^2$ containing only legal $n \times m$ patterns. We prove that if such a coloring exists then also a periodic coloring exists (Corollary 5). This further implies, using standard arguments, that in this setup there is an algorithm to determine if the given patterns admit at least one coloring of the grid (Corollary 6). The results also extend to other convex shapes in place of the rectangle (see Section 6).

We believe the low complexity setting has relevant applications. There are numerous examples of processes in physics, chemistry and biology where macroscopic patterns and regularities arise from simple microscopic interactions. Formation of crystals and quasi-crystals is a good example where physical laws govern locally the attachments of particles to each other. Predicting the structure of the crystal from its chemical composition is a notoriously difficult problem (as already implied by the undecidability of the tiling problem) but if the number of distinct local patterns of particle attachments is sufficiently low, our results indicate that the situation may be easier to handle.

Our work is also motivated by *Nivat's conjecture* [10], an open problem concerning periodicity in low complexity colorings of the grid. The conjecture claims the following: if a coloring of $\mathbb{Z}^2$ is such that, for some $n, m \in \mathbb{N}$, the number of distinct $n \times m$ patterns is at most nm, then the coloring is necessarily periodic in some direction. If true, this conjecture directly implies a strong form of our peridicity result: in the low complexity setting, not only a coloring exists that is periodic, but in fact all admitted colorings are periodic. Our contribution to Nivat's conjecture is that we show that under the hypotheses of the conjecture, the coloring must contain arbitrarily large periodic regions (Theorem 4).

2 Preliminaries

To discuss the results in detail we need precise definitions. Let A be a finite alphabet. A coloring $c \in A^{\mathbb{Z}^2}$ of the two-dimensional grid $\mathbb{Z}^2$ with elements of A is called a (two-dimensional) *configuration*. We use the notation $c_{\mathbf{n}}$ for the color $c(\mathbf{n}) \in A$ of cell $\mathbf{n} \in \mathbb{Z}^2$. For any $\mathbf{t} \in \mathbb{Z}^2$, the *translation* $\tau^{\mathbf{t}} : A^{\mathbb{Z}^2} \longrightarrow A^{\mathbb{Z}^2}$ by $\mathbf{t}$ is defined by $\tau^{\mathbf{t}}(c)_{\mathbf{n}} = c_{\mathbf{n}-\mathbf{t}}$, for all $c \in A^{\mathbb{Z}^2}$ and all $\mathbf{n} \in \mathbb{Z}^2$. If $\tau^{\mathbf{t}}(c) = c$ for a non-zero $\mathbf{t} \in \mathbb{Z}^2$, we say that c is *periodic* and that $\mathbf{t}$ is a *vector of periodicity*. If there are two linearly independent vectors of periodicity then c is *two-periodic*, and in this case there are horizontal and vertical vectors of periodicity $(k, 0)$ and $(0, k)$ for some $k > 0$, and consequently a vector of periodicity in every rational direction.

A *finite pattern* is a coloring $p \in A^D$ of some finite domain $D \subset \mathbb{Z}^d$. For a fixed D, we call such p also a *D-pattern*. The set $[p] = \{c \in A^{\mathbb{Z}^2} \mid c|_D = p\}$ of configurations that contain pattern p in domain D is the *cylinder* determined by p. We say that pattern p *appears* in configuration c, or that c *contains* pattern p, if some translate $\tau^{\mathbf{t}}(c)$ of c is in $[p]$. For a fixed finite D, the set of D-patterns that appear in a configuration c is denoted by $\mathrm{Patt}(c, D)$, that is,

$$\mathrm{Patt}(c, D) = \{\tau^{\mathbf{t}}(c)|_D \mid \mathbf{t} \in \mathbb{Z}^2\}.$$

We say that c has *low complexity* with respect to shape D if $|\mathrm{Patt}(c, D)| \leq |D|$, and we call c a *low complexity configuration* if it has low complexity with respect to some finite D.

▶ **Conjecture** (Maurice Nivat 1997 [10]). *Let $c \in A^{\mathbb{Z}^2}$ be a two-dimensional configuration. If c has low complexity with respect to some rectangle $D = \{1, \ldots, n\} \times \{1, \ldots, m\}$ then c is periodic.*

The analogous claim in dimensions higher than two fails, as does an analogous claim in two dimensions for many other shapes than rectangles [5].

2.1 Algebraic concepts

Kari and Szabados introduced in [9] an algebraic approach to study low complexity configurations. The present paper heavily relies on this technique. In this approach we replace the colors in A by distinct integers, so that we assume $A \subseteq \mathbb{Z}$. We then express a configuration

$c \in A^{\mathbb{Z}^2}$ as a formal power series $c(x, y)$ over two variables x and y in which the coefficient of monomial $x^i y^j$ is $c_{i,j}$, for all $i, j \in \mathbb{Z}$. Note that the exponents of the variables range from $-\infty$ to $+\infty$. In the following also polynomials may have negative powers of variables so all polynomials considered are actually Laurent polynomials. Let us denote by $\mathbb{Z}[x^{\pm 1}, y^{\pm 1}]$ and $\mathbb{Z}[[x^{\pm 1}, y^{\pm 1}]]$ the sets of such polynomials and power series, respectively. We call a power series $c \in \mathbb{Z}[[x^{\pm 1}, y^{\pm 1}]]$ *finitary* if its coefficients take only finitely many different values. Since we color the grid using finitely many colors, configurations are identified with finitary power series.

Multiplying a configuration $c \in \mathbb{Z}[[x^{\pm 1}, y^{\pm 1}]]$ by a monomial corresponds to translating it, and the periodicity of the configuration by vector $\mathbf{t} = (n, m)$ is then equivalent to $(x^n y^m - 1)c = 0$, the zero power series. More generally, we say that polynomial $f \in \mathbb{Z}[x^{\pm 1}, y^{\pm 1}]$ *annihilates* power series c if the formal product fc is the zero power series. Note that variables x and y in our power series and polynomials are treated only as "position indicators": in this work we never plug in any values to the variables.

The set of polynomials that annihilates a power series is a Laurent polynomial ideal, and is denoted by

$$\mathrm{Ann}(c) = \{f \in \mathbb{Z}[x^{\pm 1}, y^{\pm 1}] \mid fc = 0\}.$$

It was observed in [9] that if a configuration has low complexity with respect to some shape D then it is annihilated by some non-zero polynomial $f \neq 0$.

▶ **Lemma 1** ([9]). *Let $c \in \mathbb{Z}[[x^{\pm 1}, y^{\pm 1}]]$ be a low complexity configuration. Then $Ann(c)$ contains a non-zero polynomial.*

One of the main results of [9] states that if a configuration c is annihilated by a non-zero polynomial then it has annihilators of particularly nice form:

▶ **Theorem 2** ([9]). *Let $c \in \mathbb{Z}[[x^{\pm 1}, y^{\pm 1}]]$ be a configuration (a finitary power series) annihilated by some non-zero polynomial. Then there exist pairwise linearly independent $(i_1, j_1), \ldots, (i_m, j_m) \in \mathbb{Z}^2$ such that*

$$(x^{i_1} y^{j_1} - 1) \cdots (x^{i_m} y^{j_m} - 1) \in Ann(c).$$

Note that both Lemma 1 and Theorem 2 were proved in [9] for configurations $c \in A^{\mathbb{Z}^d}$ in arbitrary dimension d. In this work we only deal with two-dimensional configurations, so above we stated these results for $d = 2$.

If $X \subseteq A^{\mathbb{Z}^2}$ is a set of configurations, we denote by $\mathrm{Ann}(X)$ the set of Laurent polynomials that annihilate all elements of X. We call $\mathrm{Ann}(X)$ the annihilator ideal of X.

2.2 Dynamical systems concepts

Cylinders $[p]$ are a base of a compact topology on $A^{\mathbb{Z}^2}$, namely the product of discrete topologies on A. See, for example, the first few pages of [6]. The topology is equivalently defined by a metric on $A^{\mathbb{Z}^2}$ where two configurations are close to each other if they agree with each other on a large region around cell $\mathbf{0}$.

A subset X of $A^{\mathbb{Z}^2}$ is a *subshift* if it is closed in the topology and closed under translations. Equivalently, every configuration c that is not in X contains a finite pattern p that prevents it from being in X: no configuration that contains p is in X. We can then as well define subshifts using forbidden patterns: for a set P of finite patterns, define

$$X_P = \{c \in A^{\mathbb{Z}^2} \mid \forall \mathbf{t} \in \mathbb{Z}^2 \; \forall p \in P \; : \; \tau^{\mathbf{t}}(c) \notin [p] \},$$

the set of configurations that avoid all patterns in P. Set X_P is a subshift, and every subshift is X_P for some P. If $X = X_P$ for some finite P then X is a *subshift of finite type* (SFT).

The *tiling problem* (aka the domino problem) is the decision problem that asks whether a given SFT is empty, that is, whether there exists a configuration avoiding a given finite collection P of forbidden finite patterns. Usually this question is asked in terms of so-called Wang tiles, but our formulation is equivalent. The tiling problem is undecidable [1]. An SFT is called *aperiodic* if it is non-empty but does not contain any periodic configurations. Aperiodic SFTs exist [1], and in fact they must exist because of the undecidability of the tiling problem [13]. We recall the reason for this fact in the proof of Corollary 6.

Convergence of a sequence $c^{(1)}, c^{(2)}, \ldots$ of configurations to a configuration c in our topology has the following simple meaning: For every cell $\mathbf{n} \in \mathbb{Z}^2$ we must have $c^{(i)}_{\mathbf{n}} = c_{\mathbf{n}}$ for all sufficiently large i. As usual, we denote then $c = \lim_{i\to\infty} c^{(i)}$. Note that if all $c^{(i)}$ are in a subshift X, so is the limit. Compactness of space $A^{\mathbb{Z}^2}$ means that every sequence has a converging subsequence. In the proof of Theorem 3 in Section 4 we frequently use this fact and extract converging subsequences from sequences of configurations.

The *orbit* of configuration c is the set $\mathcal{O}(c) = \{\tau^{\mathbf{t}}(c) \mid \mathbf{t} \in \mathbb{Z}^2\}$ that contains all translates of c. The *orbit closure* $\overline{\mathcal{O}(c)}$ of c is the topological closure of the orbit $\mathcal{O}(c)$. It is a subshift, and in fact it is the intersection of all subshifts that contain c. The orbit closure $\overline{\mathcal{O}(c)}$ can hence be called the subshift generated by c. In terms of finite patterns, $c' \in \overline{\mathcal{O}(c)}$ if and only if every finite pattern that appears in c' appears also in c. $\overline{\mathcal{O}(c)}$ can be seen as the subshift containing all the translates of c (its orbit) and all the limits of those translates. Thus it can be different of $\mathcal{O}(c)$: if c is the configuration that with a black cell at the origin and white everywhere else, all the configurations of its orbit will contain a black cell, but at different positions; however its orbit closure contains the configuration with only white cells, as it is a limit of translations of c.

A configuration c is called *uniformly recurrent* if for every $c' \in \overline{\mathcal{O}(c)}$ we have $\overline{\mathcal{O}(c')} = \overline{\mathcal{O}(c)}$. This is equivalent to $\overline{\mathcal{O}(c)}$ being a *minimal subshift* in the sense that it has no proper non-empty subshifts inside it. A classical result by Birkhoff [3] implies that every non-empty subshift contains a minimal subshift, so there is a uniformly recurrent configuration in every non-empty subshift.

We use the notation $\langle \mathbf{x}, \mathbf{y} \rangle$ for the inner product of vectors $\mathbf{x}, \mathbf{y} \in \mathbb{Z}^2$. For a nonzero vector $\mathbf{u} \in \mathbb{Z}^2 \setminus \{\mathbf{0}\}$ we denote

$$H_{\mathbf{u}} = \{\mathbf{x} \in \mathbb{Z}^2 \mid \langle \mathbf{x}, \mathbf{u} \rangle < 0\}$$

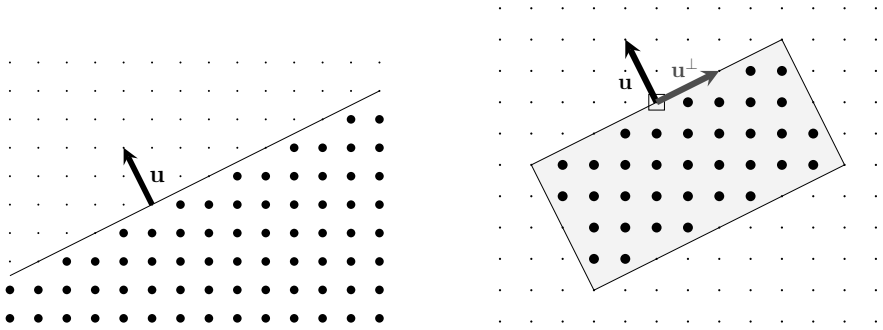
for the discrete *half plane* in direction $\mathbf{u}$. See Figure 1(a) for an illustration. A subshift X is *deterministic* in direction $\mathbf{u}$ if for all $c, c' \in X$

$$c|_{H_{\mathbf{u}}} = c'|_{H_{\mathbf{u}}} \implies c = c',$$

that is, if the contents of a configuration in the half plane $H_{\mathbf{u}}$ uniquely determines the contents in the rest of the cells. Note that it is enough to verify that the value $c_{\mathbf{0}}$ on the boundary of the half plane is uniquely determined. Indeed, if $c|_{H_{\mathbf{u}}}$ uniquely determines the line at its boundary, it is also true for all the translations of c, so the next line is also uniquely determined. By repeating this process the whole configuration is determined by $c|_{H_{\mathbf{u}}}$. Moreover, by compactness, determinism in direction $\mathbf{u}$ implies that there is a finite number k such that already the contents of a configuration in the discrete box

$$B^k_{\mathbf{u}} = \{\mathbf{x} \in \mathbb{Z}^2 \mid -k < \langle \mathbf{x}, \mathbf{u} \rangle < 0 \text{ and } -k < \langle \mathbf{x}, \mathbf{u}^\perp \rangle < k\}$$

are enough to uniquely determine the contents in cell $\mathbf{0}$, where we denote by $\mathbf{u}^\perp$ a vector that is orthogonal to $\mathbf{u}$ and has the same length as $\mathbf{u}$, e.g., $(n, m)^\perp = (m, -n)$. See Figure 1(b) for an illustration.



(a) The discrete half plane $H_{\mathbf{u}}$

(b) The discrete box $B^k_{\mathbf{u}}$ with $k = 10$.

Figure 1 Discrete regions determined by vector $\mathbf{u} = (-1, 2)$.

If X is deterministic in directions $\mathbf{u}$ and $-\mathbf{u}$ we say that $\mathbf{u}$ is a direction of *two-sided* determinism. If X is deterministic in direction $\mathbf{u}$ but not in direction $-\mathbf{u}$ we say that $\mathbf{u}$ is a direction of *one-sided* determinism. Directions of two-sided determinism correspond to directions of expansivity in the symbolic dynamics literature. If X is not deterministic in direction $\mathbf{u}$ we call $\mathbf{u}$ a *direction of non-determinism*. Finally, note that the concept of determinism in direction $\mathbf{u}$ only depends on the orientation of vector $\mathbf{u}$ and not on its magnitude.

3 Our results

Our first main new technical result is the following:

▶ **Theorem 3.** *Let c be a two-dimensional configuration that has a non-trivial annihilator. Then $\overline{\mathcal{O}(c)}$ contains a configuration c' such that $\overline{\mathcal{O}(c')}$ has no direction of one-sided determinism.*

From this result, using a technique by Cyr and Kra [7], we then obtain the second main results, stating that under the hypotheses of Nivat's conjecture, a configuration contains arbitrarily large periodic regions.

▶ **Theorem 4.** *Let c be a two-dimensional configuration that has low complexity with respect to a rectangle. Then $\overline{\mathcal{O}(c)}$ contains a periodic configuration.*

These two theorems are proved in Sections 4 and 5, respectively. But let us first demonstrate how these results imply relevant corollaries. First we consider SFTs defined in terms of allowed rectangular patterns. Let $D = \{1, \dots, n\} \times \{1, \dots, m\}$ for some $m, n \in \mathbb{N}$, and let $P \subseteq A^D$ be a set of D-patterns over alphabet A. Define $X = X_{A^D \setminus P} = \{x \in A^{\mathbb{Z}^2} \mid \mathrm{Patt}(c, D) \subseteq P\}$, the set of configurations whose D-patterns are among P.

▶ **Corollary 5.** *With the notations above, if $|P| \leq nm$ and $X \neq \emptyset$ then X contains a periodic configuration.*

Proof. Let $c \in X$ be arbitrary. By Theorem 4 then, $\overline{\mathcal{O}(c)} \subseteq X$ contains a periodic configuration. ◀

▶ **Corollary 6.** *With the notations above, there is an algorithm to determine whether $X \neq \emptyset$ for a given P of cardinality $|P| \leq nm$.*

Proof. This is a classical argumentation by H. Wang [13]: there is a semi-algorithm to test if a given SFT is empty, and there is a semi-algorithm to test if a given SFT contains a periodic configuration. Since X is an SFT, we can execute both of these semi-algorithms on X. By Corollary 5, if $X \neq \emptyset$ then X contains a periodic configuration. Hence, exactly one of these two semi-algorithms will return a positive answer. ◀

The next corollary solves Nivat's conjecture for uniformly recurrent configurations.

▶ **Corollary 7.** *A uniformly recurrent configuration c that has low complexity with respect to a rectangle is periodic.*

Proof. Because c has low complexity with respect to a rectangle then by Theorem 4 there is a periodic configuration $c' \in \overline{\mathcal{O}(c)}$. Because $\overline{\mathcal{O}(c')}$ contains only translates and limits of translates of c', all configurations in $\overline{\mathcal{O}(c')}$ are periodic. Finally, because c is uniformly recurrent we have $\overline{\mathcal{O}(c)} = \overline{\mathcal{O}(c')}$, which implies that all elements of $\overline{\mathcal{O}(c)}$, including c itself, are periodic. ◀

In Section 6 we briefly argue that all of our results remain true if the $m \times n$ rectangle is replaced by any convex discrete shape.

4 Removing one-sided determinism

In this section we prove Theorem 3 by showing how we can "remove" one-sided directions of determinism from subshifts with annihilators.

Let c be a configuration over alphabet $A \subseteq \mathbb{Z}$ that has a non-trivial annihilator. By Theorem 2 it has then an annihilator $\phi_1 \cdots \phi_m$ where each ϕ_i is of the form

$$\phi_i = x^{n_i} y^{m_i} - 1 \text{ for some } \mathbf{v}_i = (n_i, m_i) \in \mathbb{Z}^2. \tag{1}$$

Moreover, vectors $\mathbf{v}_i$ can be chosen pairwise linearly independent, that is, in different directions. We may assume $m \geq 1$.

Denote $X = \overline{\mathcal{O}(c)}$, the subshift generated by c. A polynomial that annihilates c annihilates all elements of X, because they only have local patterns that already appear in c. It is easy to see that X can only be non-deterministic in a direction that is perpendicular to one of the directions $\mathbf{v}_i$ of the polynomials ϕ_i:

▶ **Proposition 8.** *Let c be a configuration annihilated by $\phi_1 \cdots \phi_m$ where each ϕ_i is of the form (1). Let $\mathbf{u} \in \mathbb{Z}^2$ be a direction that is not perpendicular to $\mathbf{v}_i$ for any $i \in \{1, \ldots, m\}$. Then $X = \overline{\mathcal{O}(c)}$ is deterministic in direction $\mathbf{u}$.*

Proof. Suppose X is not deterministic in direction $\mathbf{u}$. By definition, there exist $d, e \in X$ such that $d \neq e$ but $d|_{H_\mathbf{u}} = e|_{H_\mathbf{u}}$. Denote $\Delta = d - e$. Because $\Delta \neq 0$ but $\phi_1 \cdots \phi_m \cdot \Delta = 0$, for some i we have $\phi_1 \cdots \phi_{i-1} \cdot \Delta \neq 0$ and $\phi_1 \cdots \phi_i \cdot \Delta = 0$. Denote $\Delta' = \phi_1 \cdots \phi_{i-1} \cdot \Delta$. Because $\phi_i \cdot \Delta' = 0$, configuration Δ' is periodic in direction $\mathbf{v}_i$. But because Δ is zero in the half plane $H_\mathbf{u}$, also Δ' is zero in some translate $H' = H_\mathbf{u} - \mathbf{t}$ of the half plane. Since the periodicity vector $\mathbf{v}_i$ of Δ' is not perpendicular to $\mathbf{u}$, the periodicity transmits the values 0 from the region H' to the entire $\mathbb{Z}^2$. Hence $\Delta' = 0$, a contradiction. ◀

Let $\mathbf{u} \in \mathbb{Z}^2$ be a one-sided direction of determinism of X. In other words, $\mathbf{u}$ is a direction of determinism but $-\mathbf{u}$ is not. By the proposition above, $\mathbf{u}$ is perpendicular to some $\mathbf{v}_i$. Without loss of generality, we may assume $i = 1$. We denote $\phi = \phi_1$ and $\mathbf{v} = \mathbf{v}_1$.

Let k be such that the contents of the discrete box $B = B^k_{\mathbf{u}}$ determine the content of cell $\mathbf{0}$, that is, for $d, e \in X$

$$d|_B = e|_B \implies d_{\mathbf{0}} = e_{\mathbf{0}}. \tag{2}$$

As pointed out in Section 2.2, any sufficiently large k can be used. We can choose k so that $k > |\langle \mathbf{u}^\perp, \mathbf{v} \rangle|$. To shorten notations, let us also denote $H = H_{-\mathbf{u}}$.

▶ **Lemma 9.** *For any $d, e \in X$ such that $\phi d = \phi e$ holds:*

$$d|_B = e|_B \implies d|_H = e|_H.$$

Proof. Let $d, e \in X$ be such that $\phi d = \phi e$ and $d|_B = e|_B$. Denote $\Delta = d - e$. Then $\phi\Delta = 0$ and $\Delta|_B = 0$. Property $\phi\Delta = 0$ means that Δ has periodicity vector $\mathbf{v}$, so this periodicity transmits values 0 from the region B to the stripe

$$S = \bigcup_{i \in \mathbb{Z}} (B + i\mathbf{v}) = \{\mathbf{x} \in \mathbb{Z}^2 \mid -k < \langle \mathbf{x}, \mathbf{u} \rangle < 0\},$$

See Figure 2 for an illustration of the regions H, B and S. As $\Delta|_S = 0$, we have that $d|_S = e|_S$. Applying (2) on suitable translates of d and e allows us to conclude that $d|_H = e|_H$. ◀

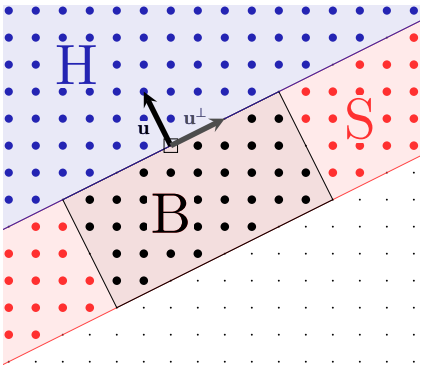


Figure 2 Discrete regions $H = H_{-\mathbf{u}}$, $B = B^k_{\mathbf{u}}$ and S in the proof of Lemma 9. In the illustration $\mathbf{u} = (-1, 2)$ and $k = 10$.

A reason to prove the lemma above is the following corollary, stating that X can only contain a bounded number of configurations that have the same product with ϕ:

▶ **Corollary 10.** *Let $c_1, \dots, c_n \in X$ be pairwise distinct. If $\phi c_1 = \cdots = \phi c_n$ then $n \leq |A|^{|B|}$.*

Proof. Let $H' = H - \mathbf{t}$, for $\mathbf{t} \in \mathbb{Z}^2$, be a translate of the half plane $H = H_{-\mathbf{u}}$ such that $c_1, \dots, c_n$ are pairwise different on H'. Consider the translated configurations $d_i = \tau^{\mathbf{t}}(c_i)$. We have that $d_i \in X$ are pairwise different on H and $\phi d_1 = \cdots = \phi d_n$. By Lemma 9, configurations d_i must be pairwise different on domain B. There are only $|A|^{|B|}$ different patterns in domain B. ◀

Let $c_1, \dots, c_n \in X$ be pairwise distinct such that $\phi c_1 = \cdots = \phi c_n$, with n as large as possible. By Corollary 10 such configurations exist. Let us repeatedly translate the configurations c_i by $\tau^{\mathbf{u}}$ and take a limit: by compactness there exists $n_1 < n_2 < n_3 \dots$ such that

$$d_i = \lim_{j \to \infty} \tau^{n_j \mathbf{u}}(c_i)$$

exists for all $i \in \{1, \dots, n\}$. Configurations $d_i \in X$ inherit the following properties from c_i:

▶ **Lemma 11.** *Let $d_1, \dots, d_n$ be defined as above. Then*
(a) *$\phi d_1 = \cdots = \phi d_n$, and*
(b) *Configurations d_i are pairwise different on translated discrete boxes $B' = B - \mathbf{t}$ for all $\mathbf{t} \in \mathbb{Z}^2$.*

Proof. Let $i_1, i_2 \in \{1, \dots, n\}$ be arbitrary, $i_1 \neq i_2$.

(a) Because $\phi c_{i_1} = \phi c_{i_2}$ we have, for any $n \in \mathbb{N}$,

$$\phi\tau^{n\mathbf{u}}(c_{i_1}) = \tau^{n\mathbf{u}}(\phi c_{i_1}) = \tau^{n\mathbf{u}}(\phi c_{i_2}) = \phi\tau^{n\mathbf{u}}(c_{i_2}).$$

Function $c \mapsto \phi c$ is continuous in the topology so

$$\phi d_{i_1} = \phi \lim_{j\to\infty} \tau^{n_j\mathbf{u}}(c_{i_1}) = \lim_{j\to\infty} \phi\tau^{n_j\mathbf{u}}(c_{i_1}) = \lim_{j\to\infty} \phi\tau^{n_j\mathbf{u}}(c_{i_2}) = \phi \lim_{j\to\infty} \tau^{n_j\mathbf{u}}(c_{i_2}) = \phi d_{i_2}.$$

(b) Let $B' = B - \mathbf{t}$ for some $\mathbf{t} \in \mathbb{Z}^2$. Suppose $d_{i_1}|_{B'} = d_{i_2}|_{B'}$. By the definition of convergence, for all sufficiently large j we have $\tau^{n_j\mathbf{u}}(c_{i_1})|_{B'} = \tau^{n_j\mathbf{u}}(c_{i_2})|_{B'}$. This is equivalent to $\tau^{n_j\mathbf{u}+\mathbf{t}}(c_{i_1})|_B = \tau^{n_j\mathbf{u}+\mathbf{t}}(c_{i_2})|_B$. By Lemma 9 then also $\tau^{n_j\mathbf{u}+\mathbf{t}}(c_{i_1})|_H = \tau^{n_j\mathbf{u}+\mathbf{t}}(c_{i_2})|_H$ where $H = H_{-\mathbf{u}}$. This means that for all sufficiently large j the configurations c_{i_1} and c_{i_2} are identical on the domain $H - n_j\mathbf{u} - \mathbf{t}$. But these domains cover the whole $\mathbb{Z}^2$ as $j \longrightarrow \infty$ so that $c_{i_1} = c_{i_2}$, a contradiction. ◀

Now we pick one of the configurations d_i and consider its orbit closure. Choose $d = d_1$ and set $Y = \overline{\mathcal{O}(d)}$. Then $Y \subseteq X$. Any direction of determinism in X is also a direction of determinism in Y. Indeed, this is trivially true for any subset of X. But, in addition, we have the following:

▶ **Lemma 12.** *Subshift Y is deterministic in direction $-\mathbf{u}$.*

Proof. Suppose the contrary: there exist configurations $x, y \in Y$ such that $x \neq y$ but $x|_H = y|_H$ where, as usual, $H = H_{-\mathbf{u}}$. In the following we construct $n+1$ configurations in X that have the same product with ϕ, which contradicts the choice of n as the maximum number of such configurations.

By the definition of Y all elements of Y are limits of sequences of translates of $d = d_1$, that is, there are translations $\tau_1, \tau_2, \dots$ such that $x = \lim_{i\to\infty} \tau_i(d)$, and translations $\sigma_1, \sigma_2, \dots$ such that $y = \lim_{i\to\infty} \sigma_i(d)$. Apply the translations $\tau_1, \tau_2, \dots$ on configurations $d_1, \dots, d_n$, and take jointly converging subsequences: by compactness there are $k_1 < k_2 < \dots$ such that

$$e_i = \lim_{j\to\infty} \tau_{k_j}(d_i)$$

exists for all $i \in \{1, \dots, n\}$. Here, clearly, $e_1 = x$.

Let us prove that $e_1, \dots, e_n$ and y are $n+1$ configurations that (i) have the same product with ϕ, and (ii) are pairwise distinct. This contradicts the choice of n as the maximum number of such configurations, and thus completes the proof.

(i) First, $\phi x = \phi y$: Because $x|_H = y|_H$ we have $\phi x|_{H-\mathbf{t}} = \phi y|_{H-\mathbf{t}}$ for some $\mathbf{t} \in \mathbb{Z}^2$. Consider $c' = \tau^{\mathbf{t}}(\phi x - \phi y)$, so that $c'|_H = 0$. As $\phi_2 \cdots \phi_m$ annihilates ϕx and ϕy, it also annihilates c'. An application of Proposition 8 on configuration c' in place of c shows that $\overline{\mathcal{O}(c')}$ is deterministic in direction $-\mathbf{u}$. (Note that $-\mathbf{u}$ is not perpendicular to $\mathbf{v}_j$ for any $j \neq 1$, because $\mathbf{v}_1$ and $\mathbf{v}_j$ are not parallel and $-\mathbf{u}$ is perpendicular to $\mathbf{v}_1$.) Due to the determinism, $c'|_H = 0$ implies that $c' = 0$, that is, $\phi x = \phi y$.

Second, $\phi e_{i_1} = \phi e_{i_2}$ for all $i_1, i_2 \in \{1, \dots, n\}$: By Lemma 11 we know that $\phi d_{i_1} = \phi d_{i_2}$. By continuity of the function $c \mapsto \phi c$ we then have

$$\begin{aligned} \phi e_{i_1} &= \phi \lim_{j\to\infty} \tau_{k_j}(d_{i_1}) = \lim_{j\to\infty} \phi\tau_{k_j}(d_{i_1}) = \lim_{j\to\infty} \tau_{k_j}(\phi d_{i_1}) \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \| \\ \phi e_{i_2} &= \phi \lim_{j\to\infty} \tau_{k_j}(d_{i_2}) = \lim_{j\to\infty} \phi\tau_{k_j}(d_{i_2}) = \lim_{j\to\infty} \tau_{k_j}(\phi d_{i_2}) \end{aligned}$$

Because $e_1 = x$, we have shown that $e_1, \dots, e_n$ and y all have the same product with ϕ.

(ii) Pairwise distinctness: First, y and $e_1 = x$ are distinct by the initial choice of x and y. Next, let $i_1, i_2 \in \{1, \dots, n\}$ be such that $i_1 \neq i_2$. Let $\mathbf{t} \in \mathbb{Z}^2$ be arbitrary and consider the translated discrete box $B' = B - \mathbf{t}$. By Lemma 11(b) we have $\tau_{k_j}(d_{i_1})|_{B'} \neq \tau_{k_j}(d_{i_2})|_{B'}$ for all $j \in \mathbb{N}$, so taking the limit as $j \longrightarrow \infty$ gives $e_{i_1}|_{B'} \neq e_{i_2}|_{B'}$. This proves that $e_{i_1} \neq e_{i_2}$. Moreover, by taking $\mathbf{t}$ such that $B' \subseteq H$ we see that $y|_{B'} = x|_{B'} = e_1|_{B'} \neq e_i|_{B'}$ for $i \geq 2$, so that y is also distinct from all e_i with $i \geq 2$. ◀

The following proposition captures the result established above.

▶ **Proposition 13.** *Let c be a configuration with a non-trivial annihilator. If $\mathbf{u}$ is a one-sided direction of determinism in $\overline{\mathcal{O}(c)}$ then there is a configuration $d \in \overline{\mathcal{O}(c)}$ such that $\mathbf{u}$ is a two-sided direction of determinism in $\overline{\mathcal{O}(d)}$.* ◀

Now we are ready to prove Theorem 3.

Proof of Theorem 3. Let c be a two-dimensional configuration that has a non-trivial annihilator. Every non-empty subshift contains a minimal subshift [3], and hence there is a uniformly recurrent configuration $c' \in \overline{\mathcal{O}(c)}$. If $\overline{\mathcal{O}(c')}$ has a one-sided direction of determinism $\mathbf{u}$, we can apply Proposition 13 on c' and find $d \in \overline{\mathcal{O}(c')}$ such that $\mathbf{u}$ is a two-sided direction of determinism in $\overline{\mathcal{O}(d)}$. But because c' is uniformly recurrent, $\overline{\mathcal{O}(d)} = \overline{\mathcal{O}(c')}$, a contradiction. ◀

5 Periodicity in low complexity subshifts

In this section we prove Theorem 4. Every non-empty subshift contains a uniformly recurrent configuration, so we can safely assume that c is uniformly recurrent.

Our proof of Theorem 4 splits in two cases based on Theorem 3: either $\overline{\mathcal{O}(c)}$ is deterministic in all directions or for some $\mathbf{u}$ it is non-deterministic in both directions $\mathbf{u}$ and $-\mathbf{u}$. The first case is handled by the following well-known corollary from a theorem of Boyle and Lind [4]:

▶ **Proposition 14.** *A configuration c is two-periodic if and only if $\overline{\mathcal{O}(c)}$ is deterministic in all directions.* ◀

For the second case we apply the technique by Cyr and Kra [7]. This technique was also used in [11] to address Nivat's conjecture. The result that we read from [7, 11], although it is not explicitly stated in this form, is the following:

▶ **Proposition 15.** *Let c be a two-dimensional uniformly recurrent configuration that has low complexity with respect to a rectangle. If for some $\mathbf{u}$ both $\mathbf{u}$ and $-\mathbf{u}$ are directions of non-determinism in $\overline{\mathcal{O}(c)}$ then c is periodic in a direction perpendicular to $\mathbf{u}$.*

Let us prove this proposition using lemmas from [11]. We first recall some definitions, adjusted to our terminology. Let $D \subseteq \mathbb{Z}^2$ be non-empty and let $\mathbf{u} \in \mathbb{Z}^2 \setminus \{\mathbf{0}\}$. The *edge* $E_{\mathbf{u}}(D)$ of D in direction $\mathbf{u}$ consists of the cells in D that are furthest in the direction $\mathbf{u}$:

$$E_{\mathbf{u}}(D) = \{\mathbf{v} \in D \mid \forall \mathbf{x} \in D \ \langle \mathbf{x}, \mathbf{u} \rangle \leq \langle \mathbf{v}, \mathbf{u} \rangle\}.$$

We call D *convex* if $D = C \cap \mathbb{Z}^2$ for a convex subset $C \subseteq \mathbb{R}^2$ of the real plane. For $D, E \subseteq \mathbb{Z}^2$ we say that D *fits* in E if $D + \mathbf{t} \subseteq E$ for some $\mathbf{t} \in \mathbb{Z}^2$.

The (closed) *stripe* of width k perpendicular to $\mathbf{u}$ is the set

$$S^k_{\mathbf{u}} = \{\mathbf{x} \in \mathbb{Z}^2 \mid -k < \langle \mathbf{x}, \mathbf{u} \rangle \leq 0\}.$$

Consider the stripe $S = S^k_{\mathbf{u}}$. The reader can refer to Figure 2 for an illustration of a closed stripe, the only difference being the inclusion of the upper boundary of S. Clearly its edge $E_{\mathbf{u}}(S)$ in direction $\mathbf{u}$ is the discrete line $\mathbb{Z}^2 \cap L$ where $L \subseteq \mathbb{R}^2$ is the real line through $\mathbf{0}$ that is perpendicular to $\mathbf{u}$. The *interior* S° of S is $S \setminus E_{\mathbf{u}}(S)$, that is, $S^\circ = \{\mathbf{x} \in \mathbb{Z}^2 \mid -k < \langle \mathbf{x}, \mathbf{u} \rangle < 0\}$.

A central concept from [7, 11] is the following. Let c be a configuration and let $\mathbf{u} \in \mathbb{Z}^2 \setminus \{\mathbf{0}\}$ be a direction. Recall that $\mathrm{Patt}(c, D)$ denotes the set of D-patterns that c contains. A finite discrete convex set $D \subseteq \mathbb{Z}^2$ is called $\mathbf{u}$-*balanced in* c if the following three conditions are satisfied, where we denote $E = E_{\mathbf{u}}(D)$ for the edge of D in direction $\mathbf{u}$:

(i) $|\mathrm{Patt}(c, D)| \leq |D|$,
(ii) $|\mathrm{Patt}(c, D)| < |\mathrm{Patt}(c, D \setminus E)| + |E|$, and
(iii) $|D \cap L| \geq |E| - 1$ for every line L perpendicular to $\mathbf{u}$ such that $D \cap L \neq \emptyset$.

The first condition states that c has low complexity with respect to shape D. The second condition implies that there are fewer than $|E|$ different $(D \setminus E)$-patterns in c that can be extended in more than one way into a D-pattern of c. The last condition states that the edge E is nearly the shortest among the parallel cuts across D.

▶ **Lemma 16** (Lemma 2 in [11]). *Let c be a two-dimensional configuration that has low complexity with respect to a rectangle, and let $\mathbf{u} \in \mathbb{Z}^2 \setminus \{\mathbf{0}\}$. Then c has a $\mathbf{u}$-balanced or a $(-\mathbf{u})$-balanced set $D \subseteq \mathbb{Z}^2$.*

A crucial observation in [7] connects balanced sets and non-determinism to periodicity. This leads to the following statement.

▶ **Lemma 17** (Lemma 4 in [11]). *Let d be a two-dimensional configuration and let $\mathbf{u} \in \mathbb{Z}^2 \setminus \{\mathbf{0}\}$ be such that d admits a $\mathbf{u}$-balanced set $D \subseteq \mathbb{Z}^2$. Assume there is a configuration $e \in \overline{\mathcal{O}(d)}$ and a stripe $S = S^k_{\mathbf{u}}$ perpendicular to $\mathbf{u}$ such that D fits in S and $d|_{S^\circ} = e|_{S^\circ}$ but $d|_S \neq e|_S$. Then d is periodic in direction perpendicular to $\mathbf{u}$.*

With these we can prove Proposition 15.

Proof of Proposition 15. Let c be a two-dimensional uniformly recurrent configuration that has low complexity with respect to a rectangle. Let $\mathbf{u}$ be such that both $\mathbf{u}$ and $-\mathbf{u}$ are directions of non-determinism in $\overline{\mathcal{O}(c)}$. By Lemma 16 configuration c admits a $\mathbf{u}$-balanced or a $(-\mathbf{u})$-balanced set $D \subseteq \mathbb{Z}^2$. Without loss of generality, assume that D is $\mathbf{u}$-balanced in c. As $\overline{\mathcal{O}(c)}$ is non-deterministic in direction $\mathbf{u}$, there are configurations $d, e \in \overline{\mathcal{O}(c)}$ such that $d|_{H_{\mathbf{u}}} = e|_{H_{\mathbf{u}}}$ but $d_{(0,0)} \neq e_{(0,0)}$. Because c is uniformly recurrent, exactly the same finite patterns appear in d as in c. This means that D is $\mathbf{u}$-balanced also in d. From the uniform recurrence of c we also get that $e \in \overline{\mathcal{O}(d)}$. Pick any k large enough so that D fits in the stripe $S = S^k_{\mathbf{u}}$. Because $\mathbf{0} \in S$ and $S^\circ \subseteq H_{\mathbf{u}}$, the conditions in Lemma 17 are met. By the lemma, configuration d is $\mathbf{p}$-periodic for some $\mathbf{p}$ that is perpendicular to $\mathbf{u}$. Because d has the same finite patterns as c, it follows that c cannot contain a pattern that breaks period $\mathbf{p}$. So c is also $\mathbf{p}$-periodic. ◀

Now Theorem 4 follows from Propositions 14 and 15, using Theorem 3 and the fact that every subshift contains a uniformly recurrent configuration.

Proof of Theorem 4. Let c be a two-dimensional configuration that has low complexity with respect to a rectangle. Replacing c by a uniformly recurrent element of $\overline{\mathcal{O}(c)}$, we may assume that c is uniformly recurrent. Since c is a low-complexity configuration, by Lemma 1 it has a non-trivial annihilator. By Theorem 3 there exists $c' \in \overline{\mathcal{O}(c)}$ such that $\overline{\mathcal{O}(c')}$ has no direction of one-sided determinism. If all directions are deterministic in $\overline{\mathcal{O}(c')}$, it follows from Proposition 14 that c' is two-periodic. Otherwise there is a direction $\mathbf{u}$ such that both $\mathbf{u}$ and $-\mathbf{u}$ are directions of non-determinism in $\overline{\mathcal{O}(c')}$. Now it follows from Proposition 15 that c' is periodic. ◀

6 Conclusions

We have demonstrated how the low local complexity assumption enforces global regularities in the admitted configurations, yielding algorithmic decidability results. The results were proved in full details for low complexity configurations with respect to an arbitrary rectangle. The reader can easily verify that the fact that the considered shape is a rectangle is not used in any proofs presented here, and the only quoted result that uses this fact is Lemma 16. A minor modification in the proof of Lemma 16 presented in [11] yields that the lemma remains true for any two-dimensional configuration that has low complexity with respect to any convex shape. We conclude that also all our results remain true if we use any convex discrete shape in place of a rectangle.

If the considered shape is not convex the situation becomes more difficult. Theorem 4 is not true for an arbitrary shape in place of the rectangle but all counter examples we know are based on periodic sublattices [5, 8]. For example, even lattice cells may form a configuration that is horizontally but not vertically periodic while the odd cells may have a vertical but no horizontal period. Such a non-periodic configuration may be uniformly recurrent and have low complexity with respect to a scatted shape D that only sees cells of equal parity. It remains an interesting direction of future study to determine if a sublattice structure is the only way to contradict Theorem 4 for arbitrary shapes. We conjecture that Corollaries 5 and 6 hold for arbitrary shapes, that is, that there does not exist a two-dimensional low complexity aperiodic SFT. A special case of this is the recently solved periodic cluster tiling problem [2, 12].

References

1. R. Berger. *The Undecidability of the Domino Problem.* Memoirs of the American Mathematical Society. American Mathematical Society, 1966.
2. S. Bhattacharya. Periodicity and decidability of tilings of $\mathbb{Z}^2$. *ArXiv e-prints*, February 2016. arXiv:1602.05738.
3. George D. Birkhoff. Quelques théorèmes sur le mouvement des systèmes dynamiques. *Bulletin de la Société Mathématique de France*, 40:305–323, 1912. doi:10.24033/bsmf.909.
4. Mike Boyle and Douglas Lind. Expansive subdynamics. *Transactions of the American Mathematical Society*, 349(1):55–102, 1997. URL: http://www.jstor.org/stable/2155304.
5. Julien Cassaigne. Subword complexity and periodicity in two or more dimensions. In Grzegorz Rozenberg and Wolfgang Thomas, editors, *Developments in Language Theory. Foundations, Applications, and Perspectives. Aachen, Germany, 6-9 July 1999*, pages 14–21. World Scientific, 1999. doi:10.1142/9789812792464_0002.
6. T. Ceccherini-Silberstein and M. Coornaert. *Cellular Automata and Groups.* Springer Monographs in Mathematics. Springer Berlin Heidelberg, 2010.

7 Van Cyr and Bryna Kra. Nonexpansive $\mathbb{Z}^2$-subdynamics and Nivat's Conjecture. *Transactions of the American Mathematical Society*, 367(9):6487–6537, February 2015. doi:10.1090/s0002-9947-2015-06391-0.

8 Jarkko Kari and Etienne Moutot. Nivat's conjecture and pattern complexity in algebraic subshifts. *Theoretical Computer Science*, 2019. doi:10.1016/j.tcs.2018.12.029.

9 Jarkko Kari and Michal Szabados. An Algebraic Geometric Approach to Nivat's Conjecture. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, pages 273–285. Springer, 2015.

10 M. Nivat. Keynote address at the 25th anniversary of EATCS, during ICALP 1997, Bologna, 1997.

11 Michal Szabados. Nivat's conjecture holds for sums of two periodic configurations. In A Min Tjoa, Ladjel Bellatreche, Stefan Biffl, Jan van Leeuwen, and Jiří Wiedermann, editors, *SOFSEM 2018: Theory and Practice of Computer Science*, pages 539–551, Cham, 2018. Springer International Publishing.

12 Mario Szegedy. Algorithms to tile the infinite grid with finite clusters. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 137–147. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743437.

13 H. Wang. Proving theorems by pattern recognition – II. *The Bell System Technical Journal*, 40(1):1–41, 1961. doi:10.1002/j.1538-7305.1961.tb03975.x.

The Tandem Duplication Distance Is NP-Hard

Manuel Lafond
Department of Computer Science, Universite de Sherbrooke, Sherbrooke, Quebec J1K 2R1, Canada
manuel.lafond@usherbrooke.ca

Binhai Zhu
Gianforte School of Computing, Montana State University, Bozeman, MT 59717, USA
bhz@montana.edu

Peng Zou
Gianforte School of Computing, Montana State University, Bozeman, MT 59717, USA
peng.zou@student.montana.edu

Abstract

In computational biology, tandem duplication is an important biological phenomenon which can occur either at the genome or at the DNA level. A tandem duplication takes a copy of a genome segment and inserts it right after the segment – this can be represented as the string operation $AXB \Rightarrow AXXB$. Tandem exon duplications have been found in many species such as human, fly or worm, and have been largely studied in computational biology.

The *Tandem Duplication* (TD) distance problem we investigate in this paper is defined as follows: given two strings S and T over the same alphabet, compute the smallest sequence of tandem duplications required to convert S to T. The natural question of whether the TD distance can be computed in polynomial time was posed in 2004 by Leupold et al. and had remained open, despite the fact that tandem duplications have received much attention ever since. In this paper, we prove that this problem is NP-hard, settling the 16-year old open problem. We further show that this hardness holds even if all characters of S are distinct. This is known as the *exemplar* TD distance, which is of special relevance in bioinformatics. One of the tools we develop for the reduction is a new problem called the *Cost-Effective Subgraph*, for which we obtain W[1]-hardness results that might be of independent interest. We finally show that computing the exemplar TD distance between S and T is fixed-parameter tractable. Our results open the door to many other questions, and we conclude with several open problems.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Tandem duplication, Text processing, Formal languages, Computational genomics, FPT algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.15

Related Version A full version of this paper is available at `https://arxiv.org/abs/1906.05266`.

Funding ML was supported by NSERC of Canada, and BZ was partially supported by NNSF of China under project 61628207.

1 Introduction

Tandem duplication is a biological process that creates consecutive copies of a segment of a genome during DNA replication. Representing genomes as strings, this event transforms a string AXB into another string $AXXB$. This process is known to occur either at small scale at the nucleotide level, or at large scale at the genome level [5, 6, 7, 23, 12]. For instance, it is known that the Huntington disease is associated with the duplication of 3 nucleotides `CAG` [13], whereas at genome level, tandem duplications are known to involve multiple genes during cancer progression [26]. Furthermore, gene duplication is believed to be the main driving force behind evolution, and the majority of duplications affecting organisms are believed to be of the tandem type (see e.g. [29]). As a result, around 3% of the human genome are formed of tandem repeats.

For these reasons, tandem duplications have received significant attention in the last decades, both in practice and theory. The combinatorial aspects of tandem duplications have been studied extensively by computational biologists [2, 14, 16, 22, 30], one question of interest being to reconstruct the evolution of a cluster of tandem repeats by duplications that could have given rise to the observed sequences. In parallel, various formal language communities [9, 31, 25] have investigated the expressive power of tandem duplications on strings.

From the latter perspective, a natural question arises: given a string S, what is the language that can be obtained starting from S and applying (any number of) tandem duplications, i.e. rules of the form $AXB \to AXXB$, where X can be any substring of S? This question was first asked in 1984 in the context of so-called *copying systems* [11]. Combined with results from [3], it was shown that this language is regular if S is on a binary alphabet, but not regular for larger alphabets. These results were rediscovered 15 years later in [9, 31]. In [25], it was shown that the membership, inclusion and regularity testing problems on the language defined by S can all be decided in linear time (still on binary alphabets). In [25, 24, 19], similar problems are also considered on non-binary alphabets, when the length $|X|$ of duplicated strings is bounded by a constant. More recently, Cho et al. [8] introduced a tandem duplication system where the *depth* of a character, i.e. the number of "generations" it took to generate it, is considered. In [18, 20], the authors study the *expressive power* of tandem duplications, a notion based on the subsequences that can be obtained from various types of copying mechanisms.

More directly related to our work, Alon et al. [1] recently investigated the minimum number of duplications required to transform a string S into another string T. We call this the *Tandem Duplication (TD) distance*. More specifically, the authors show that on binary strings, the maximum TD distance between a square-free string S and a string T of length n is $\Theta(n)$. They also mention the unsolved algorithmic problem of computing the TD distance between S and T. In fact, this question was posed earlier in [25] (pp. 306, Open Problem 3) by Leupold et al. and has remained open ever since. We settle this open problem in this paper for an unbounded alphabet. As will be seen, our technique is different from that used in [1], which only works for binary strings.

On the other hand, the TD distance is one of the many ways of comparing two genomes represented as strings in computational biology – other notable examples include breakpoint [17] and transpositions distances, the latter having recently been shown NP-hard in a celebrated paper of Bulteau et al. [4]. The TD distance has itself received special attention recently, owing to its role in cancer evolution [27].

Our results. In this paper, we solve the problem posed by Leupold et al. in 2004 and show that computing the TD distance from a string S to a string T is NP-hard. We show that this result holds even if S is *exemplar*, i.e. if each character of S is distinct. Exemplar strings are commonly studied in computational biology [28], since they represent genomes that existed prior to duplication events. We note that simply deciding if S can be transformed into T by a sequence of TDs still has unknown complexity. In our case, we show that the hardness of minimizing TDs holds on instances in which such a sequence is guaranteed to exist.

As demonstrated by the transpositions distance in [4], obtaining NP-hardness results for string distances can sometimes be an involving task. Our hardness reduction is also quite technical, and one of the tools we develop for it is a new problem we call the *Cost-Effective Subgraph*. In this problem, we are given a graph G with a cost c, and we must choose a subset X of $V(G)$. Each edge with both endpoints in X has a cost of $|X|$, every other edge

costs c, and the goal is to find a subset X of minimum cost. We show that this problem is W[1]-hard for parameter $p + c$, where p is a parameter that asks if one can achieve a cost of at most $c|E(G)| - p$ (here $c|E(G)|$ is an upper bound on the cost[1]). The problem enforces optimizing the tradeoff between covering many edges versus having a large subset of high cost, which might be applicable to other problems. In our case it captures the main difficulty in computing TD distances. We then obtain some positive results by showing that if S is exemplar, then one can decide if S can be transformed into T using at most k duplications in time $2^{O(k^2)} + poly(n)$, where n is the length of T. The result is obtained through an exponential size kernel. All of our results concern strings with unbounded alphabet sizes. Finally, we conclude with several open problems that might be of interest to the theoretical computer science community.

This paper is organized as follows. In Section 2, we give basic definitions. In Section 3, we show that computing the TD distance is NP-hard through the Cost-Effective Subgraph problem. In Section 4, we show that computing the exemplar TD distance is FPT. In Section 5, we conclude the paper with several open problems.

2 Preliminary notions

We borrow the string terminology and notation from [15]. In particular, $[n]$ denotes the set of integers $\{1, 2, \ldots, n\}$. Unless stated otherwise, all the strings in the paper are on an alphabet denoted Σ. If S_1 and S_2 are two strings, we usually denote their concatenation by S_1S_2. For a string S, we write $\Sigma(S)$ for the subset of characters of Σ that have at least one occurrence in S. A string S is called *exemplar* if $|S| = |\Sigma(S)|$, i.e. each character present in S occurs only once. A *substring* of S is a contiguous sequence of characters within S. A *prefix* (resp. *suffix*) of S is a substring that occurs at the beginning (resp. end) of S, i.e. if $S = S_1S_2$ for some strings S_1 and S_2, then S_1 is a prefix of S and S_2 a suffix of S. A *subsequence* of S is a string that can be obtained by successively deleting characters from S.

A *tandem duplication* (TD) is an operation on a string S that copies a substring X of S and inserts the copy after the occurrence of X in S. In other words, a TD transforms $S = AXB$ into $AXXB$. Given another string T, we write $S \Rightarrow T$ if there exist strings A, B, X such that $S = AXB$ and $T = AXXB$. More generally, we write $S \Rightarrow_k T$ if there exist $S_1, \ldots, S_{k-1}$ such that $S \Rightarrow S_1 \Rightarrow \ldots \Rightarrow S_{k-1} \Rightarrow T$. We also write $S \Rightarrow_* T$ if there exists some k such that $S \Rightarrow_k T$.

▶ **Definition 1.** *The TD distance $dist_{TD}(S, T)$ between two strings S and T is the minimum value of k satisfying $S \Rightarrow_k T$. If $S \Rightarrow_* T$ does not hold, then $dist_{TD}(S, T) = \infty$.*

We use the term *distance* here to refer to the number of TD operations from a string S to another string T, but one may note that TD is not a metric in the formal sense. In particular, $dist_{TD}$ is not symmetric since duplications can only increase the length of a string.

A *square string* is a string of the form XX, i.e. a concatenation of two identical substrings. Given a string S, a *contraction* is the reverse of a tandem duplication. That is, it takes a square string XX contained in S and deletes one of the two copies of X. We write $T \rightarrowtail S$ if there exist strings A, B, X such that $T = AXXB$ and $S = AXB$. We also define $T \rightarrowtail_k S$ and $T \rightarrowtail_* S$ for contractions analogously as for TDs. Observe that by the symmetry of

[1] In other words, if we were to state the maximization version of the Cost-Effective Subgraph problem, p would be the value to maximize. The minimization version, however, is more convenient to use for our needs.

duplications and contractions, $T \rightarrowtail_k S$ if and only if $S \Rightarrow_k T$ and $T \rightarrowtail_* S$ if and only if $S \Rightarrow_* T$. When there is no possible confusion, we will sometimes write $T \rightarrowtail S$ instead $T \rightarrowtail_* S$.

We have the following problem.

The Tandem Duplication (TD) problem:
Input: Two strings S and T over the same alphabet Σ and an integer k.
Question: Is $dist_{TD}(S,T) \leq k$?

In the Exemplar-TD variant of this problem, S is required to be exemplar. In either variant, we may call S the *source string* and T the *target string*. We will often use the fact that S and T form a YES instance if and only if T can be transformed into S by a sequence of at most k contractions. See Fig.1 for a simple example.

		Sequence	*Operations*
Sequence	$T = \langle a, c, \underline{g, g}, a, c, g\rangle$		*contraction on* $\langle g, g\rangle$
		$\langle \underline{a, c, g, a, c, g}\rangle$	*contraction on* $\langle a, c, g, a, c, g\rangle$
Sequence	$S = \langle a, c, g\rangle$		

Figure 1 An example for transforming sequence T to S by two contractions. The corresponding sequence of TDs from S to T would duplicate a, c, g, and then duplicate the first g.

We recall that although we study the minimization problem here, it is unknown whether the question $S \Rightarrow_* T$ can be decided in polynomial time. Nonetheless, our NP-hardness reduction applies to "promise" instances in which $S \Rightarrow_* T$ always holds.

3 NP-hardness of Exemplar-TD

To facilitate the presentation of our hardness proof, we first make an intermediate reduction using the Cost-Effective Subgraph problem, which we will then reduce to the promise version of the Exemplar-TD problem.

The Cost-Effective Subgraph problem

Suppose we are given a graph $G = (V, E)$ and an integer cost $c \in \mathbb{N}_{>0}$. For a subset $X \subseteq V$, let $E(X) = \{uv \in E : u, v \in X\}$ denote the edges inside of X. The *cost* of X is defined as

$$cost(X) = c \cdot (|E(G)| - |E(X)|) + |X| \cdot |E(X)|.$$

The Cost-Effective Subgraph problem asks for a subset X of minimum cost. In the decision version of the problem, we are given an integer r and we want to know if there is a subset X whose cost is at most r. Observe that $X = \emptyset$ or $X = V$ are possible solutions.

The idea is that each edge "outside" of X costs c and each edge "inside" costs $|X|$. Therefore, we pay for each edge not included in X, but if X gets too large, we pay more for edges in X. We must therefore find a balance between the size of X and its number of edges. The connection with the TD problem can be roughly described as follows: in our reduction, we will have many substrings which need to be deleted through contractions. We will have to choose an initial set of contractions X and then, each substring will have two ways to be contracted: one way requires c contractions, and the other requires $|X|$.

An obvious solution for a Cost-Effective Subgraph is to take $X = \emptyset$, which is of cost $c|E(G)|$. Another formulation of the problem could be whether there is a subset X of cost at most $c|E(G)| - p$, where p can be seen as a "profit" to maximize. Treating c and p as parameters, we show the NP-hardness and W[1]-hardness in parameters $c + p$ of the Cost-Effective Subgraph problem (we do not study the parameter r). Our reduction to the TD problem does not preserve W[1]-hardness and we only use the NP-hardness in this paper, but the W[1]-hardness might be of independent interest.

Before proceeding, we briefly argue the relevance of parameter c in the W[1]-hardness. If c is a fixed constant, then we may assume that any solution X satisfies $|X| \leq c$. This is because if $|X| > c$, every edge included in X will cost more than c and putting $X = \emptyset$ yields a lower cost. Thus for fixed c, it suffices to brute-force every subset X of size at most c and we get a $n^{O(c)}$ time algorithm. Our W[1]-hardness shows that it is difficult to remove this exponential dependence between n and c.

▶ **Theorem 2.** *The Cost-Effective Subgraph problem is NP-hard and W[1]-hard for parameter $c + p$.*

Proof. We reduce from CLIQUE. In this classic problem, we are given a graph G and an integer k, and must decide whether G contains a clique of size at least k, where a clique is a set of vertices in which every pair shares an edge. This problem is NP-hard [21] and also W[1]-hard in parameter k [10]. We will assume that k is even (which does not alter either hardness results).

Let (G, k) be a CLIQUE instance, letting $n := |V(G)|$ and $m := |E(G)|$. The graph in our Cost-Effective Subgraph instance is also G. We set the cost $c = 3k/2$, which is an integer since k is even, and set

$$r := c\left(m - \binom{k}{2}\right) + k\binom{k}{2} = cm + \binom{k}{2}(k - c) = cm - \frac{k}{2}\binom{k}{2}.$$

We ask whether G admits a subgraph X satisfying $cost(X) \leq r$. We show that (G, k) is a YES instance to CLIQUE if and only if G contains a set $X \subseteq V(G)$ of cost at most r. This will prove both NP-hardness and W[1]-hardness in $c + p$ (noting that here $p = k/2\binom{k}{2}$).

The forward direction is easy to see. If G is a YES instance, it has a clique X of size exactly k. Since $|E(X)| = \binom{k}{2}$, the cost of X is precisely r.

Let us consider the converse direction. Assume that (G, k) is a NO instance of CLIQUE. Let $X \subseteq V(G)$ be any subset of vertices. We will show that $cost(X) > r$. There are 3 cases to consider depending on $|X|$.

Case 1: $|X| = k$. Since G is a NO instance, X is not a clique and thus $|E(X)| = \binom{k}{2} - h$, where $h > 0$. We have that $cost(X) = c(m - \binom{k}{2} + h) + k(\binom{k}{2} - h) = cm + \binom{k}{2}(k - c) + h(c - k) = r + h(c - k)$. Since $c > k$ and $h > 0$, the cost of X is strictly greater than r.

Case 2: $|X| = k + l$ for some $l > 0$. Denote $|E(X)| = \binom{k+l}{2} - h$, where $0 \leq h \leq \binom{k+l}{2}$ The cost of X is

$$\begin{aligned} cost(X) &= c\left(m - \binom{k+l}{2} + h\right) + (k + l)\left(\binom{k+l}{2} - h\right) \\ &= cm + \binom{k+l}{2}(k + l - c) + h(c - k - l) \\ &= cm + \binom{k+l}{2}\left(l - \frac{k}{2}\right) + h\left(\frac{k}{2} - l\right). \end{aligned}$$

Considering the difference

$$\begin{aligned} cost(X) - r &= \binom{k+l}{2}\left(l - \frac{k}{2}\right) + h\left(\frac{k}{2} - l\right) - \left(-\frac{k}{2}\right)\binom{k}{2} \\ &= \frac{3kl^2}{4} - \frac{kl}{4} + \frac{l^3}{2} - \frac{l^2}{2} + h\left(\frac{k}{2} - l\right), \end{aligned}$$

if $k/2 - l \geq 0$, then the difference is clearly above 0 regardless of h, and then $cost(X) > r$ as desired. Thus we may assume that $k/2 - l < 0$. In this case, we may further assume that $h = \binom{k+l}{2}$, as this minimizes the difference. But in this case,

$$cost(X) = cm + \binom{k+l}{2}\left(l - \frac{k}{2}\right) + \binom{k+l}{2}\left(\frac{k}{2} - l\right) = cm > r,$$

which concludes this case.

Case 3: $|X| = k - l$, with $l > 0$. If $k = l$, then $X = \emptyset$ and $cost(X) = cm > r$. So we assume $k > l$. Put $|E(X)| = \binom{k-l}{2} - h$, where $h \geq 0$. We have

$$\begin{aligned} cost(X) &= c\left(m - \binom{k-l}{2} + h\right) + (k-l)\left(\binom{k-l}{2} - h\right) \\ &= cm + \binom{k-l}{2}(k - l - c) + h(c - k + l) \\ &= cm + \binom{k-l}{2}\left(-\frac{k}{2} - l\right) + h\left(\frac{k}{2} + l\right). \end{aligned}$$

The difference with this cost and r is

$$\begin{aligned} cost(X) - r &= \binom{k-l}{2}\left(-\frac{k}{2} - l\right) + h\left(\frac{k}{2} + l\right) - \left(-\frac{k}{2}\right)\binom{k}{2} \\ &= \frac{3kl^2}{4} + \frac{kl}{4} - \frac{l^3}{2} - \frac{l^2}{2} + h\left(\frac{k}{2} + l\right) \\ &> \frac{1}{4}(3l^3 + l^2) - \frac{1}{2}(l^3 + l^2) \geq 0, \end{aligned}$$

the latter since $k > l \geq 1$. Again, it follows that $cost(X) > r$. ◀

Reduction to Exemplar-TD (promise version)

Since the reduction is somewhat technical, we provide an overview of the techniques that we will use. Let (G, c, r) be a Cost-Effective Subgraph instance where c is the cost and r the optimization value, and with vertices $V(G) = \{v_1, \ldots, v_n\}$. We will construct strings S and T and argue on the number of contractions to go from T to S. We would like our source string to be $S = x_1x_2 \ldots x_n$, where each x_i is a distinct character that corresponds to vertex v_i. Let S' be obtained by doubling every x_i, i.e. $S' = x_1x_1x_2x_2 \ldots x_nx_n$. Our goal is to put $T = S'E_1E_2 \ldots E_m$, where each E_i is a substring gadget corresponding to edge $e_i \in E(G)$ that we must remove to go from T to S. Assuming that there is a sequence of contractions that transforms T into S, we make it so that we first want to contract some, but not necessarily all, of the doubled x_i's of S', resulting in another string S''. Let t be the number of x_i's contracted from S' to S''. For instance, we could have $S'' = x_1x_1x_2x_3x_3x_4x_5x_5$, where only x_2 and x_4 were contracted, and thus $t = 2$. The idea is that these contracted x_i's correspond to the vertices of a cost-effective subgraph. After T is transformed to $S''E_1 \ldots E_m$, we then

force each E_i to use S'' to contract it. For $m = 3$, a contraction sequence that we would like to enforce would take the form

$$\underline{S'}E_1E_2E_3 \rightarrowtail \underline{S''E_1}E_2E_3 \rightarrowtail \underline{S''E_2}E_3 \rightarrowtail \underline{S''E_3} \rightarrowtail \underline{S''} \rightarrowtail S,$$

where we underline the substring affected by contractions at each step. We make it so that when contracting $S''E_iE_{i+1}\dots E_m$ into $S''E_{i+1}\dots E_m$, we have two options. Suppose that v_j, v_k are the endpoints of edge e_i. If, in S'', we had chosen to contract x_j and x_k, we can contract E_i using a sequence of t moves. Otherwise, we must contract E_i using another more costly sequence of c moves. The total cost to eliminate the E_i gadgets will be $c(m-e)+te$, where e is the number of edges that can be contracted using the first choice, i.e. for which both endpoints were chosen in S''.

Unfortunately, constructing S' and the E_i's to implement the above idea is not straightforward. The main difficulty lies in forcing an optimal solution to behave as we describe, i.e. enforcing going from S' to S'' first, enforcing the E_i's to use S'', and enforcing the two options to contract E_i with the desired costs. In particular, we must replace the x_i's by carefully constructed substrings X_i. We must also repeat the sequence of E_i's a certain number p times. We now proceed with the technical details.

▶ **Theorem 3.** *The Exemplar-TD problem is NP-complete, even if for the given string S and T, $S \Rightarrow_* T$ is guaranteed to hold.*

Proof. To see that the problem is in NP, note that $dist_{TD}(S,T) \leq |T|$ since each contraction from T to S removes at least character. Thus a sequence of contractions can serve as a certificate, has polynomial size and is easy to verify.

For hardness, we reduce from the Cost-Effective Subgraph problem, which has been shown NP-hard in Theorem 2. Let (G,c,r) be an instance of Cost-Effective Subgraph, letting $n := |V(G)|$ and $m := |E(G)|$. Here c is the "outsider edge" cost and we ask whether there is a subset $X \subseteq V(G)$ such that $c(m-|E(X)|)+|X||E(X)| \leq r$. We denote $V(G) = \{v_1,\dots,v_n\}$ and $E(G) = \{e_1,\dots,e_m\}$. The ordering of vertices and edges is arbitrary but remains fixed for the remainder of the proof. For convenience, we allow the edge indices to loop through 1 to m, and so we put $e_i = e_{i+lm}$ for any integer $l \geq 0$. Thus we may sometimes refer to an edge e_h with an index $h > m$, meaning that e_h is actually the edge $e_{((h-1) \mod m)+1}$.

The construction. Let us first make an observation. If we take an exemplar string $X = x_1\dots x_l$ (i.e. a string in which no character occurs twice), we can double its characters and obtain a string $X' = x_1x_1\dots x_lx_l$. The length of X' is only twice that of X and $dist_{TD}(X,X') = l$, i.e. going from X' to X requires l contractions. We will sometimes describe pairs of strings X and X' at distance l without explicitly describing X and X', but the reader can assume that X starts as an exemplar string of length l and we obtain X' by doubling each character, as above.

Now we show how to construct S and T. First let $d = m+1$ and $p = m(n+m)^{10}$. The exact values of d and p are not crucial and will only refer to them when needed: for the most part, it is enough to think of d and p as simply "large enough". Note however that p is a multiple of m. For later reference, the value of k we will use in the reduction is $k = p/m \cdot d(r+nm) + 4cdn$.

Instead of doubling x_i's as in the intuition paragraph above, we will duplicate some characters d times. Moreover, we can't create a T string that behaves exactly as described above, but we will show that we can append p copies of carefully crafted substring to obtain the desired result. We need d and p to be high enough so that "enough" copies behave as we desire.

For each $i \in [n]$, define an exemplar string X_i of length d. Moreover, create enough characters so that no two X_i strings contain a character in common. Let X_i^d be a string satisfying $dist_{TD}(X_i, X_i^d) = d$.

Then for each $j \in \{0, 1, \ldots, 2p\}$, define an exemplar string B_j. Ensure that no B_j contains a character from an X_i string, and no two B_j's contain a common character. The B_j strings can consist of a single character, with the exception of B_0 and B_1 which are special. We assume that for B_0 and B_1, we have strings B_0^* and B_1^* such that

$$dist_{TD}(B_0, B_0^*) = dc + 2d - 2,$$
$$dist_{TD}(B_1, B_1^*) = dn + 2d - 1.$$

Again, this can be done using the doubling trick on exemplar strings. The B_j's are the building blocks of larger strings. For each $q \in [2p]$, define

$$\mathcal{B}_q = B_q B_{q-1} \ldots B_2 B_1 B_0, \qquad \mathcal{B}_q^0 = B_q B_{q-1} \ldots B_2 B_1 B_0^*,$$
$$\mathcal{B}_q^1 = B_q B_{q-1} \ldots B_2 B_1^* B_0, \qquad \mathcal{B}_q^{01} = B_q B_{q-1} \ldots B_2 B_1^* B_0^*.$$

These strings are used as "blockers" and prevent certain contractions from happening. Note that $\mathcal{B}_q^0$ and $\mathcal{B}_q^1$ can be turned into $\mathcal{B}_q$ using $dc+2d-2$ contractions and $dn+2d-1$ contractions, respectively. Moreover, $\mathcal{B}_q^{01}$ can be turned into $\mathcal{B}_q^0$ using $dn + 2d - 1$ contractions and into $\mathcal{B}_q^1$ using $dc + 2d - 2$ contractions.

Also define the strings

$$\mathcal{X} = X_1 X_2 \ldots X_n, \qquad \mathcal{X}^d = X_1^d X_2^d \ldots X_n^d,$$

and for edge $e_q = v_i v_j$ with $q \in [p]$ whose endpoints are v_i and v_j, define

$$\mathcal{X}_{e_q} = X_1^d \ldots X_{i-1}^d X_i X_{i+1}^d \ldots X_{j-1}^d X_j X_{j+1}^d \ldots X_n^d.$$

Thus in $\mathcal{X}_{e_q}$, all X_k substrings are turned into X_k^d, except X_i and X_j.

Finally, define a new additional character Δ, which will be used to separate some of the components of our string. We can now define S and T. We have

$$S = \mathcal{B}_{2p}\mathcal{X}\Delta = B_{2p}B_{2p-1} \ldots B_2 B_1 B_0 X_1 X_2 \ldots X_n \Delta.$$

It follows from the definitions of $\mathcal{B}_{2p}, \mathcal{X}$ and Δ that S is exemplar. Now for $i \in [p]$, define

$$E_i := \mathcal{B}_i^{01}\mathcal{X}_{e_i}\Delta\mathcal{B}_{2p}^1\mathcal{X}\Delta,$$

which we will call the *edge gadget*. Define T as

$$\begin{aligned} T &= \mathcal{B}_{2p}^0\mathcal{X}^d\Delta\mathcal{B}_{2p}^1\mathcal{X}\Delta E_1 E_2 \ldots E_p \\ &= \mathcal{B}_{2p}^0\mathcal{X}^d\Delta\mathcal{B}_{2p}^1\mathcal{X}\Delta \left[\mathcal{B}_1^{01}\mathcal{X}_{e_1}\Delta\mathcal{B}_{2p}^1\mathcal{X}\Delta\right]\left[\mathcal{B}_2^{01}\mathcal{X}_{e_2}\Delta\mathcal{B}_{2p}^1\mathcal{X}\Delta\right]\ldots\left[\mathcal{B}_p^{01}\mathcal{X}_{e_p}\Delta\mathcal{B}_{2p}^1\mathcal{X}\Delta\right]. \end{aligned}$$

We add brackets for clarity only – they indicate the distinct E_i substrings, but the brackets are not actual characters of T. The idea is that T starts with $S' = \mathcal{B}_{2p}^0\mathcal{X}^d\Delta$, a modified S in which $\mathcal{B}_{2p}$ becomes $\mathcal{B}_{2p}^0$ and the X_i substrings are turned into X_i^d. This $\mathcal{X}^d$ substring serves as a choice of vertices in our cost-effective subgraph. Each edge e_i has a "gadget substring" $E_i = \mathcal{B}_i^{01}\mathcal{X}_{e_i}\Delta\mathcal{B}_{2p}^1\mathcal{X}\Delta$. Since p is a multiple of m, the sequence of edge gadgets $E_1 E_2 \ldots E_m$ is repeated p/m times. Our goal to go from T to S is to get rid of all these edge gadgets by contractions. Note that because a E_i gadget starts with $\mathcal{B}_i^{01}$ and the gadget E_{i+1} starts with $\mathcal{B}_{i+1}^{01}$, the substring E_{i+1} has a character that the substring E_i does not have.

The hardness proof. We now show that G admits a subset of vertices W of cost at most r if and only if T can be contracted to S using at most $p/m \cdot d(r+nm) + 4cdn$ contraction operations. We include the forward direction, which is the most instructive, in the main text. The other direction can be found in the full version. Although we shall not dig into details here, it can be deduced from the ($\Rightarrow$) direction that $T \rightarrowtail_* S$ holds.

($\Rightarrow$) Suppose that G admits a subset of vertices W of cost at most r. Thus $c(m - |E(W)|) + |W| \cdot |E(W)| \leq r$. To go from T to S, first consider an edge e_i that does not have both endpoints in W. We show how to get rid of the gadget substring E_i for e_i using $dn + dc$ contractions. Note that T contains the substring $\mathcal{B}^1_{2p}\mathcal{X}\Delta E_i = \mathcal{B}^1_{2p}\mathcal{X}\Delta[\mathcal{B}^{01}_i\mathcal{X}_{e_i}\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta]$, where brackets surround the E_i occurrence that we want to remove (note that here for $i > 1$, the prefix $\mathcal{B}^1_{2p}\mathcal{X}\Delta$ is the suffix of the previous E_{i-1} gadget, and for $i = 1$, it is the suffix of the starting block of T). We can first contract $\mathcal{B}^{01}_i$ to $\mathcal{B}^1_i$ using $dc + 2d - 2$ contractions, then contract $\mathcal{X}_{e_i}$ to $\mathcal{X}$ using $d(n-2)$ contractions. The result is the $\mathcal{B}^1_{2p}\mathcal{X}\Delta[\mathcal{B}^1_i\mathcal{X}\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta]$ substring, which becomes $\mathcal{B}^1_{2p}\mathcal{X}\Delta$ using two contractions (see below). This sums to $dc + 2d - 2 + d(n-2) + 2 = dc + dn$ operations. More visually, the sequence of contractions works as follows (as before, brackets indicate the E_i substring and what remains of it, and the underlines are there to emphasize the substrings that participate in the contractions(s)):

$$
\begin{aligned}
&\mathcal{B}^1_{2p}\mathcal{X}\Delta\left[\underline{\mathcal{B}^{01}_i}\mathcal{X}_{e_i}\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta\right] && (dc+2d-2 \text{ contractions})\\
\rightarrowtail&\mathcal{B}^1_{2p}\mathcal{X}\Delta\left[\mathcal{B}^1_i\underline{\mathcal{X}_{e_i}}\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta\right] && (d(n-2) = dn-2d \text{ contractions})\\
\rightarrowtail&\mathcal{B}^1_{2p}\mathcal{X}\Delta\left[\mathcal{B}^1_i\mathcal{X}\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta\right]\\
=&B_{2p}B_{2p-1}\dots B_{i+1}\underline{\mathcal{B}^1_i\mathcal{X}\Delta}\left[\underline{\mathcal{B}^1_i\mathcal{X}\Delta}\mathcal{B}^1_{2p}\mathcal{X}\Delta\right] && (1 \text{ contraction})\\
\rightarrowtail&B_{2p}B_{2p-1}\dots B_{i+1}\mathcal{B}^1_i\mathcal{X}\Delta\left[\mathcal{B}^1_{2p}\mathcal{X}\Delta\right]\\
=&\underline{\mathcal{B}^1_{2p}\mathcal{X}\Delta}\left[\underline{\mathcal{B}^1_{2p}\mathcal{X}\Delta}\right] && (1 \text{ contraction})\\
\rightarrowtail&\mathcal{B}^1_{2p}\mathcal{X}\Delta.
\end{aligned}
$$

This sequence of $dn+dc$ contractions effectively removes the E_i substring gadget. Observe that after applying this sequence, it is still true that every remaining E_j gadget substring is preceded by $\mathcal{B}^1_{2p}\mathcal{X}\Delta$. We may therefore repeatedly apply this contraction sequence to every e_i not contained in W (including those e_i gadgets for which $i > m$). This procedure is thus applied to $p/m \cdot (m - |E(W)|)$ gadgets. We assume that we have done so, and that every e_i for which the E_i gadget substring remains is in W. Call the resulting string T'.

Now, let $\mathcal{X}_W$ be the substring obtained from $\mathcal{X}^d$ by contracting, for each $v_i \in W$, the string X^d_i to X_i. We assume that we have contracted the $\mathcal{X}^d$ substring of T' to $\mathcal{X}_W$, which uses $d|W|$ contractions (note that there is only one occurrence of $\mathcal{X}^d$ in T', namely right before the first Δ). Call T'' the resulting string. At this point, for every E_i substring gadget that remains, where E_i corresponds to edge $e_i = v_jv_k$, $\mathcal{X}_W$ contains the substrings X_j and X_k (instead of X^d_j and X^d_k).

Let i be the smallest integer for which the e_i substring gadget E_i is still in T'. This is the leftmost edge gadget still in T'', meaning that T'' has the prefix

$$\mathcal{B}^0_{2p}\mathcal{X}_W\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta\left[\mathcal{B}^{01}_i\mathcal{X}_{e_i}\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta\right],$$

where brackets indicate the E_i substring. To remove E_i, first contract $\mathcal{B}^{01}_i$ to $\mathcal{B}^0_i$, and contract $\mathcal{X}_{e_i}$ to X_W (this is possible since $e_i \subseteq W$). The result is $\mathcal{B}^0_{2p}\mathcal{X}_W\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta\left[\mathcal{B}^0_i\mathcal{X}_W\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta\right]$. One more contraction gets rid of the second half. This requires $dn + 2d - 1 + d(|W| - 2) + 1 =$

$dn + d|W|$ contractions. This procedure is applied to $p/m \cdot |E(W)|$ gadgets. To recap, the contraction sequence for E_i does as follows:

$$\begin{aligned}
&\mathcal{B}^0_{2p}\mathcal{X}_W\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta\left[\underline{\mathcal{B}^{01}_i}\mathcal{X}_{e_i}\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta\right] && (dn+2d-1 \text{ contractions})\\
\rightarrowtail&\mathcal{B}^0_{2p}\mathcal{X}_W\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta\left[\mathcal{B}^0_i\underline{\mathcal{X}_{e_i}}\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta\right] && (d(|W|-2) \text{ contractions})\\
\rightarrowtail&\underline{\mathcal{B}^0_{2p}\mathcal{X}_W\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta}\left[\underline{\mathcal{B}^0_i\mathcal{X}_W\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta}\right] && (1 \text{ contraction})\\
\rightarrowtail&\mathcal{B}^0_{2p}\mathcal{X}_W\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta.
\end{aligned}$$

After we repeat this for every E_i, all that remains is the string $\mathcal{B}^0_{2p}\mathcal{X}_W\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta$. We contract $\mathcal{X}_W$ to $\mathcal{X}$ using $d(n-|W|)$ contractions (in total, going from $\mathcal{X}^d$ to $\mathcal{X}$ required dn contractions). Then contract $\mathcal{B}^0_{2p}$ and $\mathcal{B}^1_{2p}$ to $\mathcal{B}_{2p}$ using $dc+2d-2+dn+2d-1 = d(c+n+4)-3$ contractions. One more contraction of the second half of the string yields S. The summary of the number of contractions made is

$$\begin{aligned}
&\frac{p}{m}\cdot(m-|E(W)|)\cdot(dc+dn)+\frac{p}{m}\cdot|E(W)|\cdot(dn+d|W|)+dn+d(c+n+4)-3\\
\leq&\frac{p}{m}\cdot(m-|E(W)|)\cdot(dc+dn)+\frac{p}{m}\cdot|E(W)|\cdot(dn+d|W|)+4cdn\\
=&\frac{p}{m}\cdot d\cdot(c+n)(m-|E(W)|)+\frac{p}{m}\cdot d\cdot(n+|W|)|E(W)|+4cdn\\
=&\frac{p}{m}\cdot d\cdot[c(m-|E(W)|)+|W||E(W)|+nm]+4cdn\\
\leq&\frac{p}{m}\cdot d(r+nm)+4cdn,
\end{aligned}$$

as desired.

($\Leftarrow$): this direction of the proof is somewhat involved and can be found in the full version. The idea is to show that a minimum contraction sequence must have the form similar to that in the ($\Rightarrow$) direction. The challenging part is to show that each E_i substring must get removed separately in this sequence, and that "most" of them incur a cost of either $dn + dt - 2$ or $dn + dc - 2$ for some t (this "most" is the reason that we need a large p). ◀

4 An FPT algorithm for the exemplar problem

In this section, we will show that Exemplar-k-TD can be solved in time $2^{O(k^2)} + poly(n)$ by obtaining a kernel of size $O(k2^k)$, where n is the length of T.

We first note that there is a very simple, brute-force algorithm to solve the k-TD problem, which is the variant of the TD problem with parameter k, the number of TDs to turn S into T (including Exemplar-k-TD as a particular case). This only establishes membership in the XP class, but it will be useful to evaluate the complexity of our kernelization later on.

▶ **Proposition 4.** *The k-TD problem can be solved in time $O(n^{2k})$, where n is the size of the target string.*

Proof. Let (S, T) be a given instance of k-TD. Consider the branching algorithm that, starting from T, tries to contract every substring of the form XX in T and recurses on each resulting string, decrementing k by 1 each time (the branching stops when S is obtained or when k reaches 0 without attaining S). We obtain a search tree of depth at most k and degree at most n^2, and thus it has $O(n^{2k})$ nodes. Visiting the internal nodes of this search tree only requires enumerating $O(n^2)$ substrings, which form the set of children of the node. Hence, there is no added computation cost to consider when visiting a node. ◀

From now on, we assume that we have an Exemplar-k-TD instance (S, T), and so that S is exemplar.

Let x and y be two consecutive characters in S (i.e. xy is a substring of S). We say that xy is (S, T)-*stable* if in T, every occurrence of x in T is followed by y and every occurrence of y is preceded by x. That is, the direct successor of every x character is y, and the direct predecessor of every y character is y. An (S, T)-*stable substring* $X = x_1 \dots x_l$, where $l \geq 2$, is a substring of S such that $x_i x_{i+1}$ is (S, T)-stable for every $i \in [l-1]$. We also define a string with a single character x_i to be a (S, T)-stable substring (provided x_i appears in S and T). If any substring of S that strictly contains X is not an (S, T)-stable substring, then X is called a *maximal* (S, T)-*stable substring*. Note that these definitions are independent of S and T, and so the same definitions apply for (X, Y)-stability, for any strings X and Y.

We will show that every maximal (S, T)-stable substring can be replaced by a single character, and that if T can be obtained from S using at most k tandem duplications, then this leaves strings of bounded size.

We first show that, roughly speaking, stability is maintained by all tandem duplications when going from S to T.

▶ **Lemma 5.** *Suppose that* $dist_{TD}(S, T) = k$ *and let* X *be an* (S, T)*-stable substring. Let* $S = S_0, S_1, \dots, S_k = T$ *be any minimum sequence of strings transforming* S *to* T *by tandem duplications. Then* X *is* (S, S_i)*-stable for every* $i \in [k]$*.*

Proof. Assume the lemma is false, and let S_i be the first of $S_1, \dots, S_k$ that does not satisfy the statement. Then there are two characters x, y belonging to X such that xy is (S, T)-stable, but xy is not (S, S_i)-stable.

We claim that, under our assumption, xy is not (S, S_j)-stable for any $j \in \{i, \dots, k\}$. As this includes $S_k = T$, this will contradict that xy is (S, T)-stable. We do this by induction – as a base case, xy is not (S, S_i)-stable so this is true for $j = i$. Assume that xy is not (S, S_{j-1})-stable, where $i < j \leq k$. Let D be the duplication transforming S_{j-1} to S_j (here $D = (a, b)$ contains the start and end positions of the substring of S_{j-1} to duplicate).

Suppose first that xy is not (S, S_{j-1})-stable because S_{j-1} has an occurrence of x that is not followed by y. Thus S_{j-1} has an occurrence of x, say at position p_x, followed by $z \neq y$. If we assume that xy is (S, S_j)-stable, then a y character must have appeared after this x from S_{j-1} to S_j. Changing the character next to this x is only possible if the last character duplicated by D is the x at position p_x and the first character of D is a y. In other words, denoting $S_{j-1} = A_1 y A_2 x z A_3$ for appropriate A_1, A_2, A_3 substrings, the D duplication must do the following

$$A_1 \underline{y A_2 x} z A_3 \Rightarrow A_1 \underline{y A_2 x y A_2 x} z A_3,$$

as otherwise, the character next to the above occurrence of x will remain z. But then, there is still an occurrence of x followed by z, and it follows that xy cannot be (S, S_j)-stable.

So suppose instead that xy is not (S, S_{j-1})-stable because S_{j-1} has an occurrence of y preceded by $z \neq x$. Assume the character preceding this y has changed in S_j and has become x. But one can verify that this is impossible. For completeness, we present each possible case: either D includes both z and y, includes one of them or none. These cases are represented below, and each one of them leads to an occurrence of y still preceded by z (the left-hand side represents S_{j-1} and the right-hand side represents S_j, and the A_i's are the

relevant substrings in each case):

Include both: $A_1\underline{A_2zyA_3}A_4 \Rightarrow A_1\underline{A_2zyA_3A_2zyA_3}A_4$,
Include z only: $A_1\underline{A_2zyA_3} \Rightarrow A_1\underline{A_2zA_2zyA_3}$,
Include y only: $A_1zy\underline{A_2}A_3 \Rightarrow A_1zy\underline{A_2yA_2}A_3$,
Include none: $A_1\underline{A_2}zyA_3 \Rightarrow A_1\underline{A_2A_2}zyA_3$ or $A_1zy\underline{A_2}A_3 \Rightarrow A_1zy\underline{A_2A_2}A_3$.

We have therefore shown that xy cannot be (S, S_j)-stable, and therefore not (S, T)-stable, which concludes the proof. ◀

Let S' be a substring obtained from S by tandem duplications, and let $X := S'[a..b]$ be the substring of S' at positions from a to b. Suppose that we apply a duplication $D = (c, d)$, which copies the substring $S'[c..d]$. Then we say that D *cuts* X if one of the following holds:

- $a < c \leq b$ and $b < d$, in which case we say that D cuts X *to the right*;
- $c < a$ and $a \leq d < b$, in which case we say that D cuts X *to the left*;
- $(a, b) \neq (c, d)$ and $a \leq c < d \leq b$, in which case D cuts X *inside*.

In other words, if we write $X = X_1X_2$ and $S' = UVX_1X_2WY$, cutting to the right takes the form $UVX_1\underline{X_2WY} \Rightarrow UBX_1\underline{X_2WX_2WY}$. Cutting to the left takes the form $U\underline{VX_1}X_2WY \Rightarrow U\underline{VX_1VX_1}X_2WY$. Rewriting $X = X_1X_2X_3$ and $S' = UX_1X_2X_3V$, cutting inside takes the form $UX_1\underline{X_2}X_3V \Rightarrow UX_1\underline{X_2X_2}X_3V$. Note that if D does not cut any occurrence of a maximal (S, S')-stable substring X and S'' is obtained by applying D on S', then X is (S, S'')-stable.

The next lemma shows that we can assume that maximal stable substrings never get cut, and thus always get duplicated together. The idea is that any duplication that cuts an X_j can be replaced by an equivalent duplication that doesn't.

▶ **Lemma 6.** *Suppose that $dist_{TD}(S, T) = k$, and let $X_1, \ldots, X_l$ be the set of maximal (S, T)-stable substrings. Then there exists a sequence of tandem duplications $D_1, \ldots, D_k$ transforming S into T such that no occurrence of an X_j gets cut by a D_i.*

In other words, for all $i \in [k]$ and all $j \in [l]$, the tandem duplication D_i does not cut any occurrence of X_j in the string obtained by applying $D_1, \ldots, D_{i-1}$ to S.

The above implies that we may replace each maximal (S, T)-stable substring X of S and T by a single character, since we may assume that characters of X are always duplicated together (assuming, of course, that S is exemplar). It only remains to show that the resulting strings are small enough. The proof of the following lemma has a very simple intuition. First, S has exactly 1 maximal (S, S)-stable substring. Each time we apply a duplication, we "break" at most 2 stable substrings, which creates 2 new ones. So if we apply k duplications, there are at most $2k + 1$ such substrings in the end.

▶ **Lemma 7.** *If $dist_{TD}(S, T) \leq k$, then there are at most $2k + 1$ maximal (S, T)-stable substrings.*

Proof. Let $S = S_0, S_1, \ldots, S_k = T$ be any minimum sequence of strings transforming S to T by tandem duplications. We show by induction that, for each $i \in \{0, 1, \ldots, k\}$, the number of maximal (S, S_i)-stable substrings is at most $2i + 1$. For $i = 0$, there is only one maximal (S, S)-stable substring, namely S itself. Now assume that there are at most $2(i-1)+1 = 2i-1$ maximal (S, S_{i-1})-stable substrings. Let $\mathbb{X} = \{X_1, \ldots, X_l\}$ be the set of these substrings, $l \leq 2i - 1$. We then know that S_{i-1} can be written as a concatenation of X_j's from $\mathbb{X}$

(with possible repetitions). The duplication D transforming S_{i-1} to S_i copies some of these X_j's entirely, except at most two X_j's at the ends which it may copy partially (i.e. D cuts at most two substrings from $\mathbb{X}$). In other words, the substring duplicated by D can be written as $X_j^2 X_{a_1} X_{a_2} \dots X_{a_r} X_h^1$, where $X_j = X_j^1 X_j^2$ and $X_h = X_h^1 X_h^2$ for some $j, h \in [l]$ (and $X_{a_1}, \dots, X_{a_r} \in \mathbb{X}$). Going further, S_{i-1} and S_i can be written, using appropriate substrings A, B, C that are concatenation of elements of $\mathbb{X}$, as

$$S_{i-1} = AX_j^1 \underline{X_j^2 B X_h^1} X_h^2 C \Rightarrow AX_j^1 \underline{X_j^2 B X_h^1 X_j^2 B X_h^1} X_h^2 C = S_i.$$

Now, any $X_r \in \mathbb{X} \setminus \{X_j, X_h\}$ is (S, S_i)-stable. Moreover, X_j^1, X_j^2, X_h^1 and X_h^2 are also (S, S_i)-stable. This shows that the number of maximal (S, S_i)-stable substrings is at most $2i - 1 - 2 + 4 = 2i + 1$, as desired. ◀

We can now transform an instance (S, T) of Exemplar-k-TD to a kernel, an equivalent instance (S', T') of size depending only on k.

▶ **Theorem 8.** *An instance (S, T) of Exemplar-k-TD admits a kernel (S', T') in which $|S'| \leq 2k + 1$ and $|T'| \leq (2k + 1)2^k$.*

Proof. Let S', T' be obtained from an instance (S, T) by replacing each maximal (S, T)-stable substring by a distinct character. We first prove that (S', T') is indeed a kernel by establishing its equivalence with (S, T). Clearly if (S', T') can be solved using at most k duplications, then the same applies to (S, T). By Lemma 6, the converse also holds: if (S, T) can be solved with at most k duplications, we may assume that these duplications never cut a maximal (S, T)-stable substring, and so these duplications can be applied on (S', T').

Then by Lemma 7, we know that S' has at most $2k + 1$ characters. If $dist_{TD}(S', T') \leq k$, then each duplication can at most double the size of the previous string. Therefore, T' must have size at most $(2k + 1)2^k$. ◀

The kernelization can be performed in polynomial time, as one only needs to identify maximal (S, T)-stable substrings and contract them (we do not bother with the exact complexity for now). Running the brute-force algorithm from Proposition 4 yields the following.

▶ **Corollary 9.** *The exemplar k-tandem duplication problem can be solved in time $O(((2k + 1)2^k)^{2k} + poly(n)) = 2^{O(k^2)} + poly(n)$, where n is the size of the input.*

5 Open problems

Although this work answers some open questions, many of them still deserve investigation. We conclude with some of these questions along with future research perspectives.

- Is the k-TD problem FPT in parameter k? As we observe in our Exemplar-k-TD kernelization, if T and S are large compared to k, they must share many long common substrings which could be exploited for an FPT algorithm. It is also an interesting question whether Exemplar-k-TD admits a polynomial size kernel.
- If $|\Sigma|$ is fixed, is k-TD in P? Even the $|\Sigma| = 2$ case is open. One possibility it to check whether we can reduce the alphabet of any instance to some constant by encoding each character appropriately.

- Can one decide in polynomial time whether $S \Rightarrow_* T$? The only known result on this topic is that it can be done if $|\Sigma| = 2$, as one can construct a finite automaton accepting all strings generated by S (though this automaton does not give the minimum number of duplications required).
- Does the k-TD problem admit a constant factor approximation algorithm? The answer might depend on the hardness of deciding whether $S \Rightarrow_* T$, but one might still consider the promise version of the problem.
- If the length of each duplicated string is bounded by d, is k-TD in P (with d treated as a constant)? We believe that it is FPT in $k + d$, but is it FPT in d?

References

1 Noga Alon, Jehoshua Bruck, Farzad F. Hassanzadeh, and Siddharth Jain. Duplication distance to the root for binary sequences. *IEEE Transactions on Information Theory*, 63(12):7793–7803, 2017. doi:10.1109/TIT.2017.2744608.

2 Gary Benson and Lan Dong. Reconstructing the duplication history of a tandem repeat. In *Proceedings of the 17th International Conference on Intelligent Systems for Molecular Biology*, ISMB'99, pages 44–53. AAAI, 1999.

3 Daniel P. Bovet and Stefano Varricchio. On the regularity of languages on a binary alphabet generated by copying systems. *Information Processing Letters*, 44(3):119–123, 1992. doi:10.1016/0020-0190(92)90050-6.

4 Laurent Bulteau, Guillaume Fertin, and Irena Rusu. Sorting by transpositions is difficult. *SIAM Journal on Discrete Mathematics*, 26(3):1148–1180, 2012. doi:10.1137/110851390.

5 Brian Charlesworth, Paul Sniegowski, and Wolfgang Stephan. The evolutionary dynamics of repetitive dna in eukaryotes. *Nature*, 371:215–220, 1994. doi:10.1038/371215a0.

6 Kamalika Chaudhuri, Kevin Chen, Radu Mihaescu, and Satish Rao. On the tandem duplication-random loss model of genome rearrangement. In *Proceedings of 17th ACM-SIAM Symposium on Discrete Algorithms*, SODA'06, pages 564–570. SIAM, 2006.

7 Zhi-Zhong Chen, Lusheng Wang, and Zhanyong Wang. Approximation algorithms for reconstructing the duplication history of tandem repeats. *Algorithmica*, 54(4):501–529, 2009. doi:10.1007/s00453-008-9209-8.

8 Da-Jung Cho, Yo-Sub Han, and Hwee Kim. Bound-decreasing duplication system. *Theoretical Computer Science*, 793:152–168, 2019. doi:10.1016/j.tcs.2019.06.018.

9 Juegen Dassow, Victor Mitrana, and Gheorghe Paun. On the regularity of the duplication closure. *Bulletin of the EATCS*, 69:133–136, 1999.

10 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. doi:10.1016/0304-3975(94)00097-3.

11 Andrzej Ehrenfeucht and Grzegorz Rozenberg. On regularity of languages generated by copying systems. *Discrete Applied Mathematics*, 8(3):313–317, 1984. doi:10.1016/0166-218X(84)90129-X.

12 Andrew J. Sharp et al. and Even E. Eichler. Segmental duplications and copy-number variation in the human genome. *The American J. of Human Genetics*, 77(1):78–88, 2005. doi:10.1086/431652.

13 Marcy Macdonald et al. and Peter S. Harper. A novel gene containing a trinucleotide repeat that is expanded and unstable on huntington's disease. *Cell*, 72(6):971–983, 1993. doi:10.1016/0092-8674(93)90585-E.

14 Olivier Gascuel, Michael D. Hendy, Alain Jean-Marie, and Robert McLachlan. The combinatorics of tandem duplication trees. *Systematic Biology*, 52(1):110–118, 2003. doi:10.1080/10635150390132821.

15 Dan Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge University Press, 1997.

16 Dan Gusfield and Jens Stoye. Linear time algorithms for finding and representing all the tandem repeats in a string. *Journal of Computer and System Sciences*, 69(4):525–546, 2004. doi:10.1016/j.jcss.2004.03.004.

17 Sridhar Hannenhalli and Pavel A. Pevzner. Transforming men into mice (polynomial algorithm for genomic distance problem). In *Proceedings of IEEE 36th Symposium on Foundations of Computer Science*, FOCS'95, pages 581–592. IEEE, 1995. doi:10.1109/SFCS.1995.492588.

18 Farzad F. Hassanzadeh, Moshe Schwartz, and Jehoshua Bruck. The capacity of string-duplication systems. *IEEE Transactions on Information Theory*, 62(2):811–824, 2016. doi:10.1109/TIT.2015.2505735.

19 Masami Ito, Peter Leupold, and Kayoko Shikishima-Tsuji. Closure of languages under bounded duplications. In *Proceedings of the 10th International Conference on Developments in Language Theory*, volume 4036 of *Lecture Notes in Computer Science*, pages 238–247. Spring-Verlag, 2006. doi:10.1007/11779148_22.

20 Siddharth Jain, Farzad F. Hassanzadeh, and Jehoshua Bruck. Capacity and expressiveness of genomic tandem duplication. *IEEE Transactions on Information Theory*, 63(10):6129–6138, 2017. doi:10.1109/TIT.2017.2728079.

21 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer, Boston, MA, 1972. doi:10.1007/978-1-4684-2001-2_9.

22 Gad M. Landau, Jeanette P. Schmidt, and Dina Sokol. An algorithm for approximate tandem repeats. *Journal of Computational Biology*, 8(1):1–18, 2001. doi:10.1089/106652701300099038.

23 Ivica Letunic, Richard R. Copley, and Peer Bork. Common exon duplication in animals and its role in alternative splicing. *Human Molecular Genetics*, 11(13):1561–1567, 2002. doi:10.1093/hmg/11.13.1561.

24 Peter Leupold, Carlos Martin-Vide, and Victor Mitrana. Uniformly bounded duplication languages. *Discrete Applied Mathematics*, 146(3):301–310, 2005. doi:10.1016/j.dam.2004.10.003.

25 Peter Leupold, Victor Mitrana, and Jose M. Sempere. Formal languages arising from gene repeated duplication. In Gheorghe Paun Natasa Jonoska and Grzegorz Rozenberg, editors, *Aspects of Molecular Computing*, volume 2950 of *Lecture Notes in Computer Science*, pages 297–308. Springer-Verlag, Berlin, 2004. doi:10.1007/978-3-540-24635-0_22.

26 Layla Oesper, Anna M. Ritz, Sarah J. Aerni, Ryan Drebin, and Benjamin J. Raphael. Reconstructing cancer genomes from paired-end sequencing data. *BMC Bioinformatics*, 13(Suppl 6):S10, 2012. doi:10.1186/1471-2105-13-S6-S10.

27 Letu Qingge, Xiaozhou He, Zhihui Liu, and Binhai Zhu. On the minimum copy number generation problem in cancer genomics. In *Proceedings of the 10th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, BCB'18, pages 260–269, New York, NY, 2018. ACM Press. doi:10.1145/3233547.3233586.

28 David Sankoff. Gene and genome duplication. *Current opinion in genetics and development*, 11(6):681–684, 2001. doi:10.1016/S0959-437X(00)00253-7.

29 Jack W. Szostak and Ray Wu. Unequal crossing over in the ribosomal dna of saccharomyces cerevisiae. *Nature*, 284:426–430, 1980. doi:10.1038/284426a0.

30 Olivier Tremblay-Savard, Denis Bertrand, and Nadia El-Mabrouk. Evolution of orthologous tandemly arrayed gene clusters. *BMC Bioinformatics*, 12(Suppl 9):S2, 2011. doi:10.1186/1471-2105-12-S9-S2.

31 Ming-Wei Wang. On the irregularity of the duplication closure. *Bulletin of the EATCS*, 70:162–163, 2000.

Existential Length Universality

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Wrocław, Poland
gawry@cs.uni.wroc.pl

Martin Lange
School of Electr. Eng. and Comp. Sc., University of Kassel, Kassel, Germany
martin.lange@uni-kassel.de

Narad Rampersad
Department of Math/Stats, University of Winnipeg, Winnipeg, Canada
narad.rampersad@gmail.com

Jeffrey Shallit
School of Computer Science, University of Waterloo, Waterloo, Canada
shallit@cs.uwaterloo.ca

Marek Szykuła
Institute of Computer Science, University of Wrocław, Wrocław, Poland
msz@cs.uni.wroc.pl

Abstract

We study the following natural variation on the classical universality problem: given a language $L(M)$ represented by M (e.g., a DFA/RE/NFA/PDA), does there exist an integer $\ell \geq 0$ such that $\Sigma^\ell \subseteq L(M)$? In the case of an NFA, we show that this problem is NEXPTIME-complete, and the smallest such ℓ can be doubly exponential in the number of states. This particular case was formulated as an open problem in 2009, and our solution uses a novel and involved construction. In the case of a PDA, we show that it is recursively unsolvable, while the smallest such ℓ is not bounded by any computable function of the number of states. In the case of a DFA, we show that the problem is NP-complete, and $e^{\sqrt{n \log n}(1+o(1))}$ is an asymptotically tight upper bound for the smallest such ℓ, where n is the number of states. Finally, we prove that in all these cases, the problem becomes computationally easier when the length ℓ is also given in binary in the input: it is polynomially solvable for a DFA, PSPACE-complete for an NFA, and co-NEXPTIME-complete for a PDA.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Problems, reductions and completeness

Keywords and phrases decision problem, deterministic automaton, nondeterministic automaton, pushdown automaton, regular expression, regular language, universality

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.16

Related Version A full version of the paper is available at `https://arxiv.org/abs/1702.03961`.

Funding *Narad Rampersad*: Supported in part by a grant from NSERC.
Jeffrey Shallit: Supported in part by a grant from NSERC.
Marek Szykuła: Supported in part by the National Science Centre, Poland under project number 2017/25/B/ST6/01920.

1 Introduction

The classical universality problem is the question, for a given language L over an alphabet Σ, whether $L = \Sigma^*$. Depending on how L is specified, the complexity of this problem varies. For example, when L is given as the language accepted by a DFA M, the problem is solvable

in linear time (reachability of a non-final state) and further is NL-complete [10]. When L is given by an NFA or a regular expression, it is PSPACE-complete [1]. When L is specified by a PDA (push-down automaton) or a context-free grammar, the problem is undecidable [9].

Studies on universality problems have a long tradition in computer science and still attract much interest. For instance, the universality has been studied for visibly push-down automata [2], where the question was shown to be decidable in this model (in contrast to undecidability in the ordinary model); timed automata [3]; the language of all prefixes (resp., suffixes, factors, subwords) of the given language [14]; and recently, for partially (and restricted partially) ordered NFAs [11].

In this paper, we study a basic variation of the universality problem, where instead of testing the full language, we ask whether there is a single length that is universal for the language. We focus on the following two problems:

▶ **Problem 1** (Existential length universality)**.** *Given a language L represented by a machine M of some type (DFA/RE/NFA/PDA) over an alphabet Σ of a fixed size, does there exist an integer $\ell \geq 0$ such that $\Sigma^\ell \subseteq L$?*

▶ **Problem 2** (Specified-length universality)**.** *Given a language L by a machine M of some type (DFA/RE/NFA/PDA) over an alphabet Σ of a fixed size and an integer ℓ (given in binary), is $\Sigma^\ell \subseteq L$?*

Furthermore, if such an ℓ exists, we are interested in how large the smallest ℓ can be.

▶ **Definition 3.** *The* minimum universality length *of a language L over an alphabet Σ is the smallest integer $\ell \geq 0$ such that $\Sigma^\ell \subseteq L$.*

1.1 Motivation

From the mathematical point of view, Problems 1 and 2 are natural variations of universality that surprisingly, to the best of our knowledge, have not been thoroughly investigated in the literature. Both problems can be seen as an interesting generalization of the famous Chinese remainder theorem to languages, in the sense that given periodicities with multiple periods, we ask where all these periodicities coincide. Hence, languages stand as succinct representations of integers. Moreover, both problems are motivated by potential applications in verification listed below. Finally, the techniques developed to study them are interesting on their own and are likely to find applications elsewhere. In particular, to solve the case of an NFA, we develop a novel formalism that helps to build NFAs with particular properties. Indeed, similar constructions to some of the first ingredients in our proof (variables and the Incrementation Gadget), were recently independently discovered to solve the problem of a maximal chain length of the Green relation components of the transformation semigroup of a given DFA [5].

Games with imperfect information

We consider games with imperfect information on a labeled graph [4], which are used to model, e.g., reactive systems. In such a game there are two players, P modeling the program and E modeling the environment. The game starts at the initial vertex. In one round, P chooses a label (action) and E chooses an edge (effect) from the current vertex with this label. If P is deterministic and cannot see the choices of E, then a strategy for P is just a word over the alphabet of labels. The game can end under various criteria, and the sequence of the resulting labels determines which player wins.

Under one of the simplest ending criteria, the game lasts for a given number of steps known to P. This models the situation when we are interested in the status of the system after some known amount of time. For example, this occurs if a system must work effectively for a specified duration, but in the end, we must be able to shut it down, which requires that there are no incomplete processes still running inside. Another example could be a distributed system, where processes have limited possibilities to communicate with the others and are affected by the environment; in general, processes do not know if the system has reached its global goal; hence, for a given strategy, we may need a guarantee of reaching the goal after a known number of rounds, instead of monitoring termination externally.

The main question for such a game is: does P have a winning strategy? This is equivalent to asking whether all strategies of P are losing. In other words, if L is the language of all losing strategies of P, then we ask whether all words of the given length are in L. For instance, if the winning criterion for P is just being in one of the specified vertices, then L is directly defined by the NFA obtained from the graph, where we mark all the non-specified vertices to be final. We can also ask whether the system is unsafe, i.e., we cannot find a strategy for some number of steps, which corresponds to existential length universality.

One can mention the relationship with the "firing squad synchronization problem" [12, 13]. In this classical problem, we are given a cellular automaton of n cells, with one active cell, and the goal is to reach a state in which all cells are simultaneously active. Thinking of an evolving device where the state at time i is represented by the strings of length i in L, our problem concerns how many time steps are needed until "all cells are active", that is, until all strings of length i are accepted.

Formal specifications

Our problems are strongly related to a few other questions that can be applied in formal verification, where program correctness is often expressed through inclusion problems [17]. The universality problem is closely related to the inclusion problem: clearly, universality is a special case of inclusion if the underlying language model can express Σ^*; and inclusion can be reduced to universality when this model is closed under unions and includes some (simple) regular languages (possibly folklore, cf. [6]). These reductions carry through to the *given* and the *existential length inclusion* problems. We can consider the constrained inclusion problems corresponding to the universality problems mentioned above; for instance, *existential length inclusion* asks for two languages K and L whether $K \cap \Sigma^\ell \subseteq L$ for some ℓ. Specified-length inclusion contains the essence of a specialized program verification problem: do all program runs *of a particular duration* adhere to a given specification? Likewise, existential length universality can be used to check whether there is some number ℓ such that running the given program for exactly ℓ steps ensures that the specification is met.

Another question of potential interest in program verification is the *bounded-length universality* problem, i.e., whether $\Sigma^{\leq \ell} \subseteq L$ for a given ℓ, resp. its inclusion variant. This could then be used to check whether an implementation meets its specification up to some point, in order to know, for example, whether the safety of a program can be guaranteed for as long as it is terminated externally at some point. The complexity of bounded-length universality is the same as that of specified-length universality, which is easy to show by modifying our proofs.

Table 1 Computational complexity of universality problems.

	DFA	RE	NFA	PDA
Universality	NL-c	PSPACE-c	PSPACE-c	Undecidable
Existential length universality (Problem 1)	NP-c	PSPACE-hard, in NEXPTIME	NEXPTIME-c	Undecidable
Specified-length universality (Problem 2)	PTIME	PSPACE-c	PSPACE-c	co-NEXPTIME-c
Minimal universality length	subexponential	*open*	doubly exponential	uncountable

1.2 Contribution

We have studied the problems in four cases, when L is represented by a DFA, NFA, regular expression, or PDA.

In the case where M is a DFA, existential length universality is NP-complete, and there exist n-state DFAs for which the minimal universality length is of the form $e^{\sqrt{n \log n}(1+o(1))}$, which is the best possible even when the input alphabet is binary. Specified-length universality is solvable in polynomial time.

The case of existential length universality where M is an NFA was formulated as an open question in May 2009, as mentioned in [15], and our solution requires the most involved construction of all problems studied in this paper. We show that this problem is NEXPTIME-complete.

In the case when M is a regular expression, existential length universality is PSPACE-hard and in NEXPTIME, and there are examples where the minimal universality length is exponential. The question about the exact complexity class remains open in this case. Specified-length universality for REs and also for NFAs is PSPACE-complete, which follows from modifying the PSPACE-hardness proof of the usual universality [1, Section 10.6].

Finally, in the case where M is a PDA, existential length universality is recursively unsolvable, while the minimal universality length grows faster than any computable function, which follows from the undecidability of the universality of a PDA. On the other hand, specified-length universality is co-NEXPTIME-complete, which we show by another original construction[1], though less involved than that for NFAs, reducing from an exponential variant of the tiling problem [16].

While for proving hardness we use larger alphabets, a standard binarization applies to our problems, so all the complexity results remain valid when the alphabet is binary.

Our results are summarized in Table 1. In the conference version, we present only the most involved case of the NEXPTIME-completeness of the existential length universality of an NFA. Due to the length of the proof, the main ideas are exposed, with some technical details omitted. These, and all the other proofs, are available in the full version of the paper.

2 The Case Where M is an NFA

The classical universality problem for regular expressions and so for NFAs is known to be PSPACE-complete [1, Section 10.6]. Also, if the NFA does not accept Σ^*, then the length of the shortest non-accepted words is at most exponential. Specified-length universality for NFAs is also PSPACE-complete, which can be shown by a modification of the usual proof for universality. However, we show that existential length universality is harder: it

[1] We thank an anonymous referee for pointing out that coNEXPTIME-hardness of given-length universality for PDA could also be obtained through a modification of the proof of [18, Theorem 8.1].

is NEXPTIME-complete, and there are examples where the minimal universality length is approximately doubly exponential in the number of states of the NFA.

We begin with upper bounds, which follow by determinization and the results for DFAs.

▶ **Proposition 4.** *Let M be an NFA with n states. If there exists an ℓ such that M accepts all strings of length ℓ, then the smallest such ℓ is $\leq e^{2^{n/2}\sqrt{n \log 2}(1+o(1))}$.*

▶ **Proposition 5.** *Existential length universality (Problem 1) for NFAs is in NEXPTIME.*

The difficult part is to show that existential length universality for NFAs is NEXPTIME-hard. Note that the usual method which is applied to show PSPACE-hardness of the classical universality problem does not seem enough in this case and, after a suitable modification, results only in a proof of PSPACE-hardness. The reason for this difficulty is that, to show NEXPTIME-hardness, we need to be able to construct NFAs whose minimum universality length is larger than exponential, which is a non-trivial task in itself. The NFA constructed by the reduction must have a polynomial size, whereas to solve an NEXPTIME-complete problem we may need exponential memory. If the length of a word is superexponential, then some subsets of the states of the NFA must be repeated multiple times.

To overcome this major technical hurdle we construct an NFA with minimum universality length being roughly doubly exponential by using an indirect approach. We design the automaton such that there are many exponentially long cycles on subsets of the states and it accepts all words only for the lengths that are solutions given by the Chinese Remainder Theorem for these cycle lengths. By exhibiting a family of NFAs with large minimal universality lengths we show that our construction is essentially tight. The techniques are rather involved, and hence we first develop an intermediate formalism that will be used for both tasks. Having developed our formalism, we are able to solve the first task. To solve the second task, we proceed in two steps. First, we reduce an auxiliary logic decision problem concerning the divisibility of integers to existential length universality through our formalism. Second, we reduce a canonical NEXPTIME-complete problem to this divisibility problem.

2.1 A Programming Language

We define a simple programming language that will be used to construct NFAs with particular properties in a convenient way. Our language is nondeterministic, i.e., the programs can admit many possible computations of the same length. In contrast to the usual programming languages, we are interested only in this set of admitted computations by the program.

A program will be translated in polynomial time directly to an NFA, or, more precisely, to an extended structure called a *gadget*, which is defined below. A computation of the program will correspond to a word for the constructed NFA. If the computation is not admitted, the word will be always accepted. Otherwise, usually, the word will not be accepted, with some exceptions when we additionally make some states final in the NFA.

2.1.1 Gadget Definition

Let $m \geq 1$ be a fixed integer. A *variable* V is a set of states $\{v_1, \ldots, v_m, \bar{v}_1, \ldots, \bar{v}_m\}$. These states are called *variable states*, and m is the *width* of the variable. Besides variable states, in our NFAs there will be also *control flow states*, and the unique special final state q_{acc}, which will be fixed by all transitions.

A *gadget* G is a 7-tuple $(P^G, \mathcal{V}^G, \Sigma^G, \delta^G, s^G, t^G, F^G)$. When specifying the elements of such a tuple, we usually omit the superscript if it is clear from the context. P is a set of control flow states, $\mathcal{V}$ is a set of (disjoint) variables on which the gadget *operates*, $s, t \in P$

are distinguished *start* and *target* control flow states, respectively, and $F \subseteq P$ is a set of final states. The set of states of G is $Q = \{q_{\text{acc}}\} \cup P \cup \bigcup_{V \in \mathcal{V}} V$. Then $\delta \colon Q \times \Sigma \to 2^Q$ is the transition function, which is extended to a function $2^Q \times \Sigma^* \to 2^Q$ as usual. We always have $\delta(q_{\text{acc}}, a) = \{q_{\text{acc}}\}$ for every $a \in \Sigma$.

The NFA of G is $(Q, \Sigma, \delta, s, F)$. A *configuration* is a subset $C \subseteq Q$. Given a configuration C, we say a state is *active* if it belongs to C. We say a configuration C is *proper* if it does not contain q_{acc}. Given a proper configuration C and a word w, we say that w is a *proper computation from* C if the obtained configuration after reading w is also proper, i.e., $\delta(C, w)$ is proper. Therefore, from a non-proper configuration we cannot obtain a proper one after reading any word since q_{acc} is always fixed, and so every non-proper computation from $\{s\}$ is an accepted word by the NFA.

We say that a variable V is *valid* in a configuration $C \subseteq Q$ if for all $1 \leq i \leq m$, $v_i \in C$ if and only if $\bar{v}_i \notin C$. In other words, the states $\bar{v}_1, \ldots, \bar{v}_m$ are complementary to the states $v_1, \ldots, v_m$. A valid variable stores an integer from $\{0, \ldots, 2^m - 1\}$ encoded in binary; the states v_1 and v_m represent the least and the most significant bit, respectively. Formally, if V is valid in a configuration C, then its *value* $V(C)$ is defined as $V(C) = \sum_{\substack{1 \leq i \leq m \\ v_i \in V \cap C}} 2^{i-1}$.

We say that a configuration C is *initial* for a gadget G if it is proper, contains the start state s but no other control flow states, and the gadget's variables are valid in C (if not otherwise stated, which is the case for some gadgets). A *final* configuration is a proper configuration that contains the target state t and no other control flow states. A *complete computation* is a proper computation from an initial configuration to a final configuration. Every gadget will possess some properties about its variables and the length of complete computations according to its semantics. These properties are of the form that, depending on an initial configuration C, there exists or not a complete computation of some length from C to a final configuration C', where C' also satisfies some properties. Usually, proper computations from an initial configuration will have bounded length (but not always, as we will also create cycles). Also usually, proper configurations will have exactly one active control flow state (with the exception of the Parallel Gadget, introduced later). If a variable is not required to be valid in C, then these properties will not depend on its active states in C.

We start from defining *basic* gadgets, which are elementary building blocks, and then we will define *compound* gadgets, which are defined using the other gadgets inside.

2.1.2 Basic Gadgets

Selection Gadget. This gadget is denoted by $\textsc{Select}(V)$, where V is a variable. It allows a nondeterministic selection of an arbitrary value for V. An initial configuration for this gadget does not require that V is valid. For every integer $c \in \{0, \ldots, 2^m - 1\}$ and for every initial configuration C, there exists a complete computation from C to a final configuration C' such that $V(C') = c$.

The gadget is illustrated in Fig. 1. It consists of control flow states $P = \{s = p_0, p_1, \ldots, p_{m-1}, p_m = t\}$, one variable V, and letters $\Sigma = \{\alpha_0, \alpha_1\}$. The letters α_0 and α_1 allow moving the active control flow state over the states $p_0, p_1, \ldots, p_m$ and, at each transition, choosing either v_1 or $\bar{v}_1$ to be active. Also, each v_i and $\bar{v}_i$ are shifted to v_{i+1} and $\bar{v}_{i+1}$, respectively, and both v_m and $\bar{v}_m$ are mapped to no state ($\emptyset$), which ensures that the initial content of V is neglected. Note that a word $w = \alpha_{b_1} \ldots \alpha_{b_m}$, for $b_i \in \{0, 1\}$, sets the value of the variable to $\sum_{1 \leq i \leq m} 2^{i-1} b_i$.

The semantic properties are summarized in the following

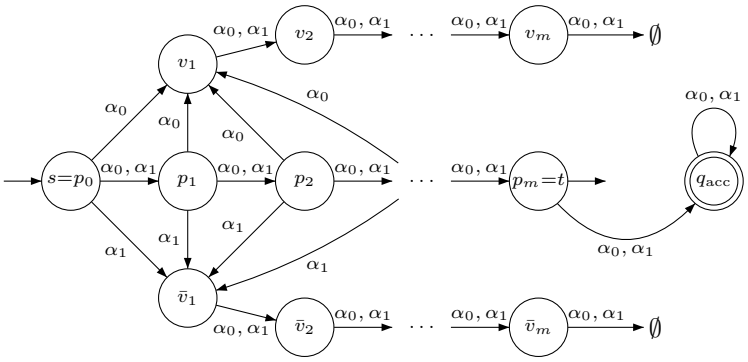


Figure 1 Selection Gadget.

▶ **Lemma 6.** *Let C be an initial configuration for the Selection Gadget* Select(V)*. For every value $c \in \{0, \ldots, 2^m - 1\}$, there exists a complete computation in Σ^m from C to a proper configuration C' such that $V(C') = c$. Every complete computation has length m, and every longer computation is not proper.*

Equality Gadget. This gadget is denoted by $U = V$, where U and V are two distinct variables. It checks if the values of valid variables U and V are equal in the initial configuration. If so, the gadget admits a complete computation, which is of length m; otherwise, every word of length at least m is a non-proper computation.

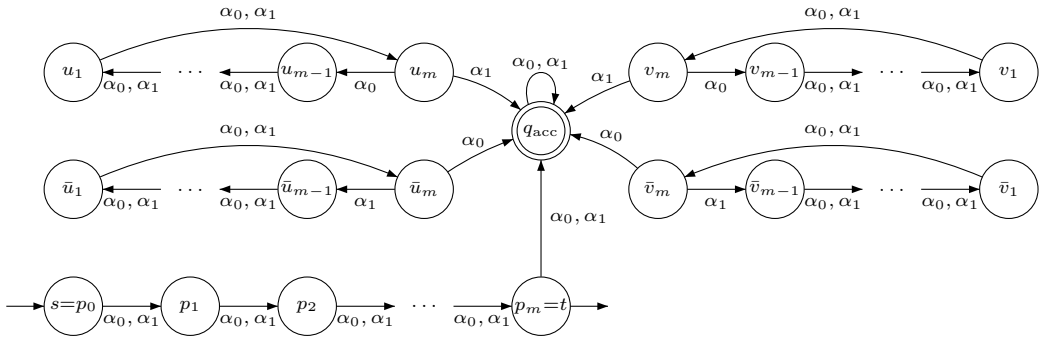


Figure 2 Equality Gadget.

The gadget is illustrated in Fig. 2. It consists of control flow states $P = \{s = p_0, p_1, \ldots, p_m = t\}$, two variables U and V, and letters $\Sigma = \{\alpha_0, \alpha_1\}$. The letters α_0 and α_1 allow moving the active control flow state over the states $s = p_0, p_1, \ldots, p_m = t$ and, at each transition, checking if the corresponding positions of U and V agree.

Inequality Gadget. This gadget is denoted by $U \neq V$, where U and V are two distinct variables. It checks if the values of the valid variables U and V are different. The construction is very similar to the Equality Gadget and its complete computations have length $m + 1$.

Incrementation Gadget. This gadget is denoted by $V{+}{+}$, where V is a variable. It increases the value of the valid variable V by 1. If the value of V is the largest possible $(2^m - 1)$, then the gadget does not allow to obtain a proper configuration by any word of length $m + 1$. Variable V must be valid in an initial configuration. The construction is similar to Equality and Inequality Gadgets and enforces a written addition of one to the value of V interpreted in binary. All complete computations have length $m + 1$.

Assignment Gadget. This gadget is denoted either by $U \leftarrow c$ or by $U \leftarrow V$, where $c \in \{0, \ldots, 2^m - 1\}$ and U and V are two distinct variables. It assigns to U either the fixed constant c or the value of the other variable V. Variable V must be valid in an initial configuration, but U does not have to be. It consists of control flow states $P = \{s, t\}$, a variable U or two variables U, V, and the unary alphabet $\Sigma = \{\alpha\}$. The transitions of α map s to t, and additionally map either s to the states of U encoding value c or the states of V to the corresponding states of U.

In fact, the case $U \leftarrow V$ could be alternatively implemented by a Selection Gadget followed by an Equality Gadget, although it will add more states and letters.

Waiting Gadget. This gadget is denoted by $\textsc{Wait}_D$, where D is a fixed positive integer. This is a very simple gadget which just does nothing for D number of letters. There is exactly one complete computation, which has length D.

2.1.3 Joining Gadgets Together

Compound gadgets are defined by other gadgets, which are joined together in the way specified by a program. The method of definition is the only difference, as compound gadgets are objects of the same type as basic gadgets. They will be also used incrementally to define further compound gadgets.

The general scheme for creating a compound gadget by joining gadgets $G_1, \ldots, G_k$ operating on variables from the sets $\mathcal{V}_1, \ldots, \mathcal{V}_k$ (all of width m), respectively, is as follows:

1. There are fresh (unique) control flow states of the gadgets, and there are the variables from $\mathcal{V}_1 \cup \cdots \cup \mathcal{V}_k$. Thus when gadgets operate on the same variable, its states are shared.
2. The alphabet contains fresh (unique) copies of the letters of the gadgets.
3. Final states in the gadgets are also final in the compound gadget.
4. The transitions are defined as in the gadgets, whereas the transitions of a letter from a gadget G_i map every control flow state that does not belong to G_i to $\{q_{\text{acc}}\}$ and fix the states of the variables on which G_i does not operate.
5. Particular definitions of compound gadgets may additionally identify some of the start and target states of the gadgets and may add more (fresh) control flow states and letters.

In our constructions, the control flow states with their transitions will form a directed graph, where the out-degree at every state of every letter is one (except the Parallel Gadget, defined later, which is an exception from the above scheme) – it either maps a control flow state to another one or to q_{acc}. States of variables will never be mapped to control flow states. This will ensure that during every proper computation from an initial configuration, exactly one control flow state is active. The active control flow state will determine which letters can be used by a proper computation, i.e., the letters from the gadget owning this state (but in which it is not the target state).

Moreover, we will ensure that whenever a proper computation activates the start state of an internal gadget G_i, the current configuration restricted to the states of G_i is an initial configuration for G_i – this boils down to assuring that the variables required to be valid have been already initialized (e.g., by a Selection Gadget or an Assignment Gadget). Hence, complete computations for the compound gadget will contain complete computations for the internal gadgets, and the semantic properties of the compound gadget are defined in a natural way from the properties of the internal gadgets.

Now we define basic ways to join gadgets together. Let $G_1, \ldots, G_k$ be some gadgets with start states $s^{G_1}, \ldots, s^{G_k}$ and target states $t^{G_1}, \ldots, t^{G_k}$, respectively.

Sequence Gadget. For each $i = 1, \ldots, k-1$, we identify the target state t^{G_i} with the start state $s^{G_{i+1}}$. Then s^{G_1} and t^{G_k} are respectively the start and target states of the Sequence Gadget. We represent this construction by writing $I_1 \ \ldots \ I_k$. Complete computations for this gadget are concatenations of complete computations for the internal gadgets.

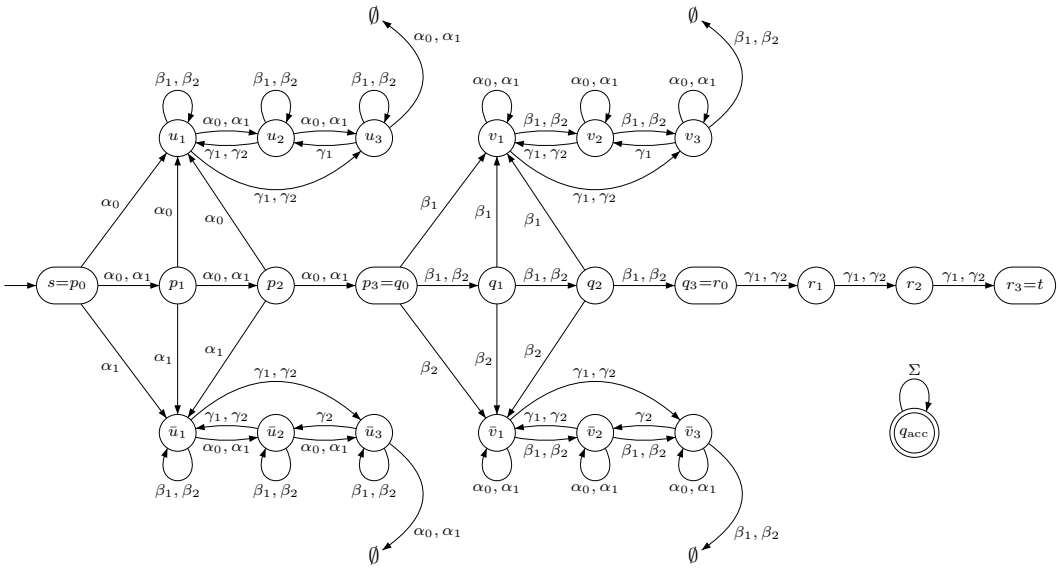


Figure 3 The complete NFA of the Sequence Gadget SELECT(U) SELECT(V) $U = V$. All omitted transitions go to q_{acc}. The states p_i, q_i, r_i and letters $\alpha_i, \beta_i, \gamma_i$ belong to the three gadgets, respectively, that is: $p_i = p_i^{\mathrm{SELECT}(U)}$, $\alpha_i = \alpha_i^{\mathrm{SELECT}(U)}$, $q_i = p_i^{\mathrm{SELECT}(V)}$, $\beta_i = \alpha_i^{\mathrm{SELECT}(V)}$, $r_i = p_i^{U=V}$, $\gamma_i = \alpha_i^{U=V}$.

For example, for $k = 3$ and $m = 3$, the Sequence Gadget SELECT(U) SELECT(V) $U = V$ is shown in Fig. 3. It has the property that every complete computation has length $3m = 9$ and is a concatenation of complete computations for the three gadgets; a final configuration C' is such that $U(C') = V(C')$. There exists a complete computation for every possible value of both variables, and longer computations are not proper.

Choose Gadget. This gadget allows selecting one of the given gadgets nondeterministically. We add a fresh start state s and k unique letters $\alpha_1, \ldots, \alpha_k$. The action of a letter α_i maps s to $\{s^{G_i}\}$, maps control flow states from the gadgets to $\{q_{\mathrm{acc}}\}$, and fixes variables states. All target states t^{G_i} are identified into the target state t of the Choose Gadget. We represent this construction by: **choose:** I_1 **or:** $\ldots$ **or:** I_k **end choose**.

Note that for this gadget there may exist complete computations of different lengths, even for the same initial configuration. Nevertheless, there exists an upper bound on the length such that every computation longer than this bound is not proper (which is 1 plus the maximum from the bounds for the internal gadgets).

Using the above constructions, we can easily develop **If-Else Gadget** and **While Gadget** (which use Equality or Inequality Gadgets for checking conditions). A special version of the latter is **While-True Gadget**, which creates an unconditional loop by identifying the start and target states of the internal gadget.

Then we can define **Addition Gadget** and **Multiplication Gadget** for performing the corresponding arithmetic operations. We also need **Primality Gadget** testing if the value of a variable is prime (there is also a negated version), and a **Prime Number Gadget**

computing the i'th prime number. For each of the defined gadgets so far, except for While Gadget in general, there always exists an upper bound on the length of every complete computation, and every longer computation is not proper. This bound is at most exponential in the size of the gadget, because proper computations cannot repeat the same configuration.

2.2 An NFA With a Large Minimal Universality Length

Our first application is to show a lower bound on the maximum minimal universality length. Alg. 1 gives the program encoding our NFA. The numbers in the brackets [ ] at the right denote the length of complete computations for the gadget (or for a part of it) in the current line. In line 7, the annotation indicates that the start control flow state of this Waiting Gadget is final, so the NFA of the program has two final states in total.

Algorithm 1 Large minimal universality length.

Variables: X, Y

```
 1: SELECT(Y)                                   ▷ [m]
 2: X ← 0                                       ▷ [1]
 3: while true do
 4:     choose:                                 ▷ [1]
 5:         X = Y                               ▷ [m]
 6:         X ← 0                               ▷ [1]
 7:         [start final state] WAIT_{m+1}      ▷ [m+1]
 8:     or:
 9:         X ≠ Y                               ▷ [m+1]
10:         X++                                 ▷ [m+1]
11:     end choose
12: end while
```

The idea of the program is as follows: In the beginning, we select an arbitrary value for Y, and then in an infinite loop, we increment X modulo $Y + 1$. The Choose Gadget in lines 4–11 is, in fact, an If-Else Gadget with condition $X = Y$, unrolled for easier calculation of computation lengths. Every iteration (complete computation of the Choose Gadget) of the loop takes the same number of letters $(2m + 3)$, hence given a computation length we know that we must perform $d - 1$ complete iterations and end in the d'th iteration, for some d. A proper computation of this length can avoid the final state in line 7 only in the iterations where the value of X does not equal the value of Y. We can ensure this for every length smaller than $\mathrm{lcm}(1, 2, \ldots, 2^m) \cdot (2m + 3)$, as we can always select Y such that $Y + 1$ does not divide $d + 1$, but it is not possible for length $\mathrm{lcm}(1, 2, \ldots, 2^m) \cdot (2m + 3)$. After a detailed technical analysis and a calculation we get:

▶ **Theorem 7.** *For a* 15*-letter alphabet, the minimal universality length can be as large as* $e^{2^{n/11}(1+o(1))}$.

2.3 Controlling the Computation Length

As we noted, because of Choose Gadgets, complete computations may have different lengths. For example, the Addition Gadget for an initial configuration C performs $V(C)$ iterations of its internal while loop. Moreover, two or more branches of a Choose Gadget may admit complete computations of different lengths even for the same current configuration. This is an obstacle that makes it difficult or impossible to further rely on the exact length $|w|$ of a proper computation, based on which we would like to decide if w must be accepted. Therefore, if we want to still use our constructions, we need a possibility to ensure that all complete computations have a fixed known length, and furthermore, that there are no proper computations longer than that length.

Delaying Gadget. The first new ingredient is the Delaying Gadget $\textsc{Delay}_D$, where D is a fixed integer ≥ 0. It is a stronger version of Waiting Gadget. Using a While Gadget with an Inequality Gadget and an Incrementation Gadget, it enforces proper computations that incrementally count from 0 to D. Complete computations have length exactly $T(D)$, which is a function polynomial in D and exponential in the size of the gadget.

Parallel Gadget. The idea to control the computation length is to implement computation in parallel. A given gadget for which there may exist complete computations of different lengths is computed in parallel with a Delaying Gadget. When the computation is completed for the given gadget, we still must wait in its target state until the computation for the Delaying Gadget is also finished. In this way, as long as complete computations for the Delaying Gadget are always longer than those for the given gadget (we can ensure this by choosing D), complete computations for the joint construction will have fixed length $T(D) + 1$. The joint computation is realized by replacing the alphabet with new letters for every combined pair of actions in both gadgets.

2.4 Length Divisibility

In Subsection 2.2 we have constructed an NFA for which every word encoding a proper computation must be accepted or can be not accepted depending on its length, namely, it is always accepted if the length is divisible by $\operatorname{lcm}(1, 2, \ldots, 2^m) \cdot \mathcal{O}(m)$. We generalize this idea so that we will be able to express more complex properties about the length of which all words must be accepted.

We are going to test whether $|w|$ satisfies some properties, in particular, whether a function of $|w|$ is divisible by some integers. This extends the idea from Alg. 1, which just verifies whether $|w|/r$, for some constant r, is not divisible by some integer from 2 to 2^m. We define the *divisibility program* shown in Alg. 2. It is constructed for given numbers k and m, and a *verifying procedure*, which is any gadget satisfying some technical properties: Complete computations must be exponentially bounded by some L, longer computations must not be proper, and all outgoing transitions from the target state must go to q_{acc}; these conditions will allow synchronizing the length of all complete computations to one length $T(D) + 1 > L$ with a Parallel Gadget. Furthermore, the values of variables X_i and X'_i may not be modified and the existence of complete computations cannot depend on the setting of any internal variables in the initial configuration; these conditions ensure that the gadget can be activated repetitively in the same proper computation of the whole program. Finally, there may not be final states.

There is an infinite while loop, which consists of two parts. In the first part, a nondeterministic choice is made (line 4): either to run the verifying procedure or to wait. For the verifying procedure (line 5) we use the Parallel Gadget; this ensures that this part finishes after exactly $T(D) + 1 > L$ letters. In the waiting case (line 7), we use the Delaying Gadget with the same value of D as that in the Parallel Gadget. Then there is a single final state (line 8). In the second part (lines 10–15), every variable X'_i counts the number of iterations of the while loop modulo X_i. The *for* loop denotes that the body is instantiated for every i (it is a Sequence Gadget). Every complete computation of the second part (lines 10–15) has exactly $(2m + 3)k$ letters.

The idea is that, for certain lengths, every proper computation must end with a configuration with the non-final control flow states in line 5 (these are precisely the two target states, of the verifying procedure and of the Delaying Gadget) or with the final state in line 8. However, for the first option, it must succeed in the last iteration with the verifying procedure

Algorithm 2 Divisibility program.

Variables: $X_1, \ldots, X_k, X'_1, \ldots, X'_k$

1: $\textsc{Select}(X_1), \ldots, \textsc{Select}(X_k)$. ▷ $[km]$
2: $X'_1 \leftarrow 0, \ldots, X'_k \leftarrow 0$ ▷ $[k]$
3: **while true do**
4: **choose:** ▷ $[1]$
5: execute in parallel { $\textsc{Delay}_D$; Verifying procedure } ▷ $[T(D)+1]$
6: **or:**
7: $\textsc{Delay}_D$ ▷ $[T(D)]$
8: [final state] $\textsc{Wait}_1$ ▷ $[1]$
9: **end choose**
10: **for** $i = 1, \ldots, k$ **do**
11: X'_i++ ▷ $[m+1]$
12: **if** $X'_i = X_i$ **then** ▷ $[m+1$ if $X'_i = X_i$, and $m+2$ otherwise$]$
13: $X'_i \leftarrow 0$ ▷ $[1]$
14: **end if**
15: **end for**
16: **end while**

when $X'_i = \ell' \bmod X_i$. In other words, for some selection of the values for $X_1, \ldots, X_k$, there must exist a complete computation of the verifying procedure from an initial configuration with these values for X_i and $X'_i = \ell' \bmod X_i$. Due to these auxiliary variables X'_i, the verifying procedure can check the divisibility of ℓ by X_i.

▶ **Lemma 8.** *Consider Alg. 2 for some k, m, and a verifying procedure. There exist integers $r_1 \geq 1$ and $r_2 \geq 1$ such that the NFA of Alg. 2 accepts all words of a length ℓ if and only if there exist an integer $\ell' \geq 0$ such that:*

- *$\ell = r_1 \cdot \ell' + r_2$, and*
- *for every initial configuration C of the verifying procedure where variables $X_1, \ldots, X_k$ are valid and $X'_i(C) = \ell' \bmod X_i(C)$ for all $1 \leq i \leq k$, there does not exist a complete computation for the verifying procedure.*

2.4.1 Existential Divisibility Formulas

We develop a method for verifying the properties of the computation length in a flexible way. We use a subset of first-order logic, where formulas are in a special form. For a given integer m, we say that a formula φ is in *existential divisibility* form if its only free variable is ℓ' (not necessarily occurring in φ) and it has the following form:

$$\exists_{X_1,\ldots,X_k \in \{0,\ldots,2^m-1\}} \psi(X_1, \ldots, X_k, \ell').$$

Formula ψ is any propositional logic formula that uses operators $\wedge$, $\vee$, and whose simple propositions are of the following possible forms:

1. $(X_i = c)$, where $c \in \{0, \ldots, 2^m - 1\}$,
2. $(X_h = X_i + X_j)$,
3. $(X_h = X_i \cdot X_j)$,
4. X_i is prime,
5. X_i is the X_j'th prime number,
6. $(X_i \mid \ell')$ or $(X_i \nmid \ell')$,

where X_i, X_j, X_h are some variables from $\{X_1, \ldots, X_k\}$.

Given a φ, we can ask for what integer values of ℓ' the formula is satisfied, and in particular whether it is not a tautology over positive integers.

▶ **Problem 9** (Non-satisfiability of existential divisibility formulas). *Given an existential divisibility formula φ, is there a positive integer ℓ' such that $\varphi(\ell')$ is not satisfied?*

Verifying Gadget. For the formula ψ occurring in an existential divisibility formula φ, we construct the gadget VERIFY(ψ) for verifying ψ. The gadget uses the external variables $X_1, \ldots, X_k$, which are assumed to correspond with those in ψ, and the external auxiliary variables $X'_1, \ldots, X'_k$. There are also some fresh internal variables. The value of ℓ' is not given, but instead, we assume that the value of every X'_i is equal to ℓ' mod X_i (and 0 when $X_i = 0$), hence we will be able to check the divisibility of ℓ'.

The construction uses our components designed so far. It is built using Sequence Gadgets for conjunctions, Choose Gadgets for disjunctions, and other appropriate gadgets for (1)–(6).

The construction is such that, for an initial configuration C for VERIFY(ψ) with valid variables $X_1, \ldots, X_k$ and where $X'_i(C) = \ell'$ mod $X_i(C)$, there exists a complete computation if and only if $\psi(X_1, \ldots, X_k, \ell')$ is satisfied. Note that the gadget meets the conditions of the verifying procedure in Alg. 2.

2.4.2 Reduction from Problem 9

We already have all ingredients to reduce Problem 9 to Problem 1 (existential length universality). Given an existential divisibility formula $\varphi(\ell')$, we construct the program from Alg. 2 with the Verifying Gadget VERIFY(ψ) as the verifying procedure. Hence, the formula is translated to an NFA in polynomial time. By Lemma 8, there exist integers $r_1, r_2 \geq 1$ such that the NFA accepts all words of some length ℓ if and only if for some integer $\ell' \geq 0$, $\ell = r_1 \cdot \ell' + r_2$ and $\varphi(\ell')$ is not satisfied. Hence, if φ is not satisfied for some ℓ', then the NFA accepts all words of length ℓ, and if the NFA accepts all words of a length ℓ, then ℓ must be expressible as $r_1 \cdot \ell' + r_2$ and $\varphi(\ell')$ must be not satisfied.

▶ Remark 10. With a few more technical steps, which require, e.g., adding the negation and controlling variable bounds, it is possible to represent the negated Problem 9 as the satisfiability problem of the Presburger arithmetic with the prefix class $\exists\forall^*$ and whose formulas are of a specific form. The Presburger arithmetic with the prefix class $\exists\forall^*$ is NEXPTIME-hard [7], and a little more general one with the prefix class $\exists^*\forall^*$ is Σ_1^{EXP}-complete [8]. However, our problem is a strict subclass of the first case, because the first and the only unbounded variable ℓ' can be checked only for divisibility, all the other variables are exponentially bounded, and the propositions are of particular forms. Hence, we cannot directly infer the hardness from that known result. In the last reduction step, we show that NEXPTIME-hardness still holds for our restricted problem.

2.5 Reduction to the non-satisfiability of existential divisibility formulas

In the final step, we reduce from the canonical NEXPTIME-complete problem: whether a nondeterministic Turing machine N with s states accepts the empty input after at most 2^s steps. The idea is encoding by ℓ' a $2^s \times 2^s$ table representing a proper computation of the machine. Each symbol placed at each cell has assigned a unique prime number, and we define that the symbol is present if and only if ℓ' modulo its prime number is non-zero. Then we construct an existential divisibility formula $\varphi(\ell')$ that is satisfied for an ℓ' if and only if the encoded computation by ℓ' is not correct. Thus, the formula is a disjunction of several cases that express a possible error in the computation.

This reduces (by a polynomial reduction) an NEXPTIME-complete to Problem 9, which was reduced to Problem 1 (existential length universality). Then we can further reduce to the binary case by a standard binarization.

▶ **Theorem 11.** *Existential length universality (Problem 1) for NFAs is NEXPTIME-hard, even if the alphabet is binary.*

References

1. A. V. Aho and J. E. Hopcroft. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Longman Publishing Co., Inc., 1st edition, 1974.
2. R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proc. of the 36th Annual ACM Symposium on Theory of Computing*, STOC 2004, pages 202–211. ACM, 2004.
3. N. Bertrand, P. Bouyer, T. Brihaye, and A. Stainer. Emptiness and universality problems in timed automata with positive frequency. In *Proceedings of the 38th International Conference on Automata, Languages and Programming - Volume Part II*, ICALP 2011, pages 246–257. Springer, 2011.
4. L. Doyen and J.-F. Raskin. Games with imperfect information: theory and algorithms. In Krzysztof R. Apt and Erich Grädel, editors, *Lectures in Game Theory for Computer Scientists*, pages 185–212. Cambridge University Press, 2011.
5. L. Fleischer and M. Kufleitner. Green's relations in finite transformation semigroups. In Pascal Weil, editor, *CSR*, pages 112–125. Springer, 2017.
6. O. Friedmann and M. Lange. Ramsey-Based Analysis of Parity Automata. In *TACAS*, volume 7214 of *LNCS*, pages 64–78. Springer, 2012.
7. E. Grädel. Dominoes and the Complexity of Subclasses of Logical Theories. *Annals of Pure and Applied Logic*, 43(1):1–30, 1989.
8. C. Haase. Subclasses of Presburger Arithmetic and the Weak EXP Hierarchy. In *CSL–LICS*, CSL–LICS, pages 47:1–47:10. ACM, 2014.
9. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
10. N. D. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11(1):68–85, 1975.
11. M. Krötzsch, T. Masopust, and M. Thomazo. On the complexity of universality for partially ordered NFAs. In *MFCS*, pages 61:1–61:14, 2016.
12. J. Mazoyer. A six-state minimal time solution to the firing squad synchronization problem. *Theoretical Computer Science*, 50(2):183–238, 1987.
13. F. R. Moore and G. G Langdon. A generalized firing squad problem. *Information and Control*, 12(3):212–220, 1968.
14. N. Rampersad, J. Shallit, and Z. Xu. The computational complexity of universality problems for prefixes, suffixes, factors, and subwords of regular languages. *Fundamenta Informaticae*, 116(1–4):223–236, 2012.
15. J. Shallit. Open problems in automata theory: an idiosyncratic view, LMS Keynote Address in Discrete Mathematics, BCTCS 2014, April 10 2014, Loughborough, England. `https://cs.uwaterloo.ca/~shallit/Talks/bc4.pdf`.
16. P. van Emde Boas. The convenience of tilings. In *Complexity, Logic, and Recursion Theory*, pages 331–363. Marcel Dekker Inc., 1997.
17. M. Y. Vardi and P. Wolper. Reasoning About Infinite Computations. *Information and Computation*, 115(1):1–37, 1994.
18. G. Zetzsche. The complexity of downward closure comparisons. `https://arxiv.org/abs/1605.03149`, 2016.

On the Termination of Flooding

Walter Hussak
Computer Science, Loughborough University, UK
W.Hussak@lboro.ac.uk

Amitabh Trehan[1]
Computer Science, Loughborough University, UK
www.amitabhtrehan.net
A.Trehan@lboro.ac.uk

Abstract

Flooding is among the simplest and most fundamental of all graph/network algorithms. Consider a (distributed network in the form of a) finite undirected graph G with a distinguished node v that begins flooding by sending copies of the same message to all its neighbours and the neighbours, in the next round, forward the message to all and only the neighbours they did not receive the message from in that round and so on. We assume that nodes do not keep a record of the flooding event, thus, raising the possibility that messages may circulate infinitely even on a finite graph. We call this history-less process *amnesiac flooding* (to distinguish from a classic distributed implementation of flooding that maintains a history of received messages to ensure a node never sends the same message again). Flooding will terminate when no node in G sends a message in a round, and, thus, subsequent rounds. As far as we know, the question of termination for amnesiac flooding has not been settled – rather, non-termination is implicitly assumed.

In this paper, we show that surprisingly synchronous amnesiac flooding always terminates on any arbitrary finite graph and derive exact termination times which differ sharply in bipartite and non-bipartite graphs. In particular, synchronous flooding terminates in e rounds, where e is the eccentricity of the source node, if and only if G is bipartite, and, otherwise, in j rounds where $e < j \leq e + d + 1$ and d is the diameter of G. Since e is bounded above by d, this implies termination times of at most d and of at most $2d + 1$ for bipartite and non-bipartite graphs respectively. This suggests that if communication/broadcast to all nodes is the motivation, the history-less amnesiac flooding is asymptotically time optimal and obviates the need for construction and maintenance of spanning structures like spanning trees. Moreover, the clear separation in the termination times of bipartite and non-bipartite graphs may suggest possible mechanisms for distributed discovery of the topology/distances in an arbitrary graph.

For comparison, we also show that, for asynchronous networks, however, an adversary can force the process to be non-terminating.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Graph algorithms analysis; Theory of computation → Distributed algorithms

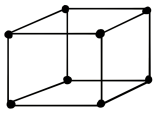
Keywords and phrases Flooding algorithm, Network algorithms, Distributed algorithms, Graph theory, Termination, Bipartiteness, Communication, Broadcast

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.17

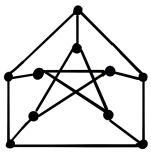
Related Version A brief announcement of this paper appeared at ACM PODC 2019 (https://doi.org/10.1145/3293611.3331586) and a related (Arxiv) version is available at https://arxiv.org/abs/1907.07078.

Funding *Amitabh Trehan*: This work was supported by the Engineering and Physical Sciences Research Council grant number EP/P021247/1.

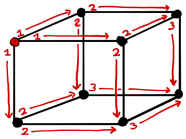
[1] corresponding author



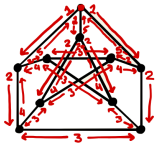
(a) The Hypercube graph (on 8 nodes).



(b) The Petersen Graph.



(c) Flooding on Hypercube.



(d) Flooding on the Petersen Graph.

Figure 1 Two well known graph topologies (Hypercube and the Petersen Graph) and execution of Amnesiac flooding (from the red coloured node) on them. The arrows point to direction of the transmission of the message with the label giving the round number. Double headed arrows indicate the message crossing over in both directions on the edge. The flooding on the hypercube terminates in only 3 = diameter rounds, whereas on the Petersen graph, it takes 5 = 2* diameter + 1 rounds.

Acknowledgements We would like to thank the anonymous reviewers for their comments, and Saket Saurabh, Jonas Lefèvre, Chhaya Trehan, Gary Bennett, Valerie King, Shay Kutten, Paul Spirakis, Abhinav Aggarwal for the useful discussions and insights and to all others in our network who attempted to solve this rather easy to state puzzle.

1 Introduction

Consider the two well known graphs in Figure 1; the hypercube (cube in 3 dimensions) graph and the Petersen graph. Now, consider distributed networks where nodes follow the following simple flooding process as a communication primitive: A single node (origin) with a message M begins the process by sending M to all its neighbours in the first round. These nodes, in the second round, in parallel forward M to all the other neighbours except the origin and so on. Nodes do this forwarding in a mechanical manner not retaining any memory, thus, forwarding M again if they receive it again. Possibly, this process can go on indefinitely. We call this process *Amnesiac Flooding (AF)* and define it more formally in later discussion. How does Amnesiac flooding behave if the topology of the network is a hypercube or a Petersen graph? What about other topologies?

Consider AF on the hypercube first (Figure 1(c)) - it is easy to see that it stops after 3 rounds when the node diagonally opposite the origin gets M from all of its neighbours simultaneously in round 3 and hence, cannot forward the message further. On the Petersen graph (Figure 1(d)), though the process terminates, it takes 5 rounds and stops at the origin itself. If we consider the termination times in terms of graph diameter, it takes diameter time on the hypergraph but much longer (2 times diameter plus 1) for the Petersen graph. Thus, the following question: *Will AF terminate on other network topologies, and if so, how long will it take? Why does the time differ markedly on the Hypercube and the Petersen graphs though they are of similar sizes (in fact, the Petersen graph has a smaller diameter)?*

Flooding is among the most basic of distributed graph/network algorithms. To quote Apnes [1]: *Flooding is about the simplest of all distributed algorithms. It is dumb and expensive, but easy to implement, and gives you both a broadcast mechanism and a way*

to build rooted spanning trees. At a high level, flooding can be simply described as: *In a network, a node begins flooding by sending a message to all its neighbours and subsequently, every node, in parallel, forwards the same message to all their neighbours.*

Flooding is the simplest strategy to achieve broadcast i.e. have a message reach every node in the network, in quick time. Often flooding is implemented with a flag that is set when the message is seen for the first time to ensure termination (see e.g. [2]) We are interested in the variant of flooding which does not explicitly use such a flag or keep a record of having seen the message before. The node selectively sends the message only to the complement of its neighbours from whom it has just received the message and subsequently forgets about that activity. The process terminates if there is no node that forwards M in a round (and, therefore, subsequent rounds). We call this amnesiac flooding (AF for short) to account for the very short term memory of the node. We analyse this very simple and theoretically interesting deterministic process on graphs and derive a rather unexpected and surprising result. We show that synchronous AF (i.e. in the synchronous message passing model where nodes send messages in parallel in synchronised rounds) terminates on every finite graph in time optimal $O(d)$ rounds, where d is the diameter of the graph. We also show that, at least in one asynchronous model, an adversary can force AF to be non-terminating.

Besides being theoretically interesting, our results also have practical implications. AF is a natural variant minimising memory overhead with nodes simply forwarding messages in a rather dumb manner. Note that if there were multiple messages being flooded in the network, the memory requirement of keeping the historical flags (for every message being flooded) could be significant, especially for low memory devices (e.g. sensor networks). Our results show that if the objective of the flooding is broadcasting, this overhead maybe unnecessary. Of course, a spanning substructure could be constructed from the initial regular flooding and used for subsequent broadcast (as is often done). However, spanning substructures can be difficult to maintain if the network is changing. This would not be required if AF was being used for communication.

We speculate (though we have not studied this in detail) that AF may correspond to certain natural and social phenomena to whose understanding our results may contribute. Consider the following possibly contrived example as a thought experiment: There is an aggressive social media user that forwards every message it receives to all its contacts but is polite enough to not forward to those who had just forwarded it the message. Naturally, such users lose track of the messages they have been forwarding. A natural question is that will a message cease getting circulated.

These need to be investigated further.

1.1 Model, Problem Definition and Results

Let $G(V, E)$ be an undirected graph (with n vertices and m edges) representing a network where the vertices represent the nodes of the network and edges represent the connections between the nodes. We consider the process in a synchronous message passing network: computation proceeds in synchronous rounds where each round consists of every node receiving messages from all its neighbours, doing local computation and sending messages to all (or some of) its neighbours. No messages are lost in transit. We consider only flooding from a single source for now.

▶ **Definition 1.** ***Synchronous Amnesiac Flooding (Synchronous AF):*** *A distinguished node, say ℓ, sends a message (say, M) to all its neighbours in round 1. In subsequent rounds, every node receiving M forwards a copy of M to every, and only those, nodes it did not receive the message from in the previous round. Algorithm 1.1 presents the algorithm formally.*

Algorithm 1.1 Synchronous Amnesiac Flooding: A message M from a source node s is "flooded" over graph G.

1: **procedure** *Flooding*(G, s) {Flooding over graph G from source node s}
2: Let $N(v) \leftarrow$ Neighbours of $v \in G$
3: Node s sends message M to all its neighbours in G {Round 1: s "floods" a message M}
4: **for** Rounds $i = 1, 2, \ldots$ **do**
5: **for** For all nodes v in parallel **do**
6: Let $I(v, M) \leftarrow$ set of neighbours of v that sent M to v in round $i-1$ $\{I(v, M) \subseteq N(v)\}$
7: Send M to $N(v) \setminus I(v, M)$ {Send to all neighbours except those who sent the message to v in the previous round}

Note that this is an "amnesiac" process i.e. nodes do not retain memory of having received or sent the message in the previous (but one) rounds. We say that flooding *terminates* when no message (i.e. a copy of M) is being sent over any edge in the network. We address the following questions:

For every finite graph G, beginning from any arbitrary vertex, will amnesiac flooding always terminate? If so, how many rounds does it take?

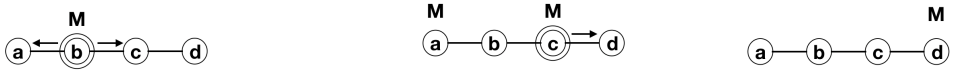
In Section 2, we answer the first part of the above question in the affirmative i.e. this flooding process will terminate for every G. For the second part of the question, in Section 3, we notice a sharp distinction between bipartite and non-bipartite graphs. Recall the standard definitions of *eccentricity* and *diameter*: eccentricity of a node is defined as the length of the longest of the shortest paths to other nodes in the graph, and diameter is the largest eccentricity of any node in the graph i.e. the longest distance between any two nodes in the graph. We show that flooding terminates in e rounds (i.e. at most d rounds), where e is the eccentricity of the source node and d the diameter of G, if and only if G is bipartite. Note that this is time optimal for broadcast. If the graph is non-bipartite, synchronous flooding takes longer: from a single source, flooding terminates in j rounds where $e < j \leq e + d + 1$.

Note that in this work, we only look at global termination i.e. the state when M stops circulating in the system. We do not discuss the related problem of individual nodes detecting that either global termination has happened or if they should stop participation in flooding. In some sense, this is even unnecessary since nodes do not need to maintain any additional state or history. There is no persistent overhead to keeping the simple amnesiac flooding process as a rule in the background.

1.1.1 Asynchronous Message Passing

For comparison, we also consider an asynchronous message passing model and show in Section 4 that an

adversary in this model can cause flooding to be non-terminating. We consider what we call as the *round-asynchronous model* where the computation still proceeds in global synchronous rounds but the adversary can decide the delay of message delivery on any link. The message cannot be lost and will be eventually delivered but the adversary can decide which round to deliver the message in. The adversary can decide on individual link delays for a round based on the state of the network for the present and previous rounds (i.e. node states, messages in transit and message history). Now, the flooding algorithm (*Asynchronous Amnesiac Flooding*) will exactly be same as Algorithm 1.1 except that the adversary decides which round a message transmitted on an edge reaches the other end. In Section 4, we show



(a) Round 1. **(b)** Round 2. **(c)** Round 3.

Figure 2 Amnesiac Flooding over a line network beginning with node b in 2 ($<$ diameter $= 3$) rounds. Circled nodes are sending M in that round.

(a) Round 1. **(b)** Round 2. **(c)** Round 3. **(d)** Round 4.

Figure 3 AF over a Triangle (Odd Cycle/Clique) network beginning with node b. Both node a and c send M to each other in round 2 and to b in round 3. Also, this is an odd (# nodes) cycle and termination takes $2d + 1$ time (d= diameter $= 1$).

that the adversary can force Asynchronous AF to be non-terminating by choosing link delays. Note that since the process is deterministic, the adversary can choose the link delays in advance (rather than needing to choose them adaptively).

We leave discussion of other asynchronous settings for future work.

1.2 Some illustrative examples

Figure 2 shows flooding over a line graph. The process begins with the node b and terminates at the ends of the graph and takes only 2 rounds, which is equal to the eccentricity of node b in the graph (which has diameter of 3). Note that a line is an example of a bipartite graph. The triangle graph is another interesting illustrative example (Figure 3) – here, termination takes 3 rounds, whereas, the diameter is only 1. Note that the triangle is also the smallest clique and the smallest non-trivial cycle with an odd number of nodes (an important topology for us). The even cycle is another interesting topology but here termination will happen in d rounds (as expected according to our bipartite graphs result). Of course, a graph can have far more complicated topology with cyclic and acyclic subgraphs.

1.3 Related work

A brief announcement of this work has appeared as [10]. The applications of flooding as a distributed algorithm are too numerous to be mentioned. It is one of the first algorithms to be introduced in distributed computing textbooks, often as the basic algorithm to solve leader election [12, 13] and set up graph substructures such as spanning trees [14, 2, 16, 19]. Flooding based algorithms (or flooding protocols) appear in areas ranging rom GPUs, High performance, shared memory and parallel computing to Mobile ad hoc networks(MANETs), Mesh Networks, Complex Networks etc [18]. In [17], Rahman et al show that flooding can even be adopted as a reliable and efficient routing scheme, comparable to sophisticated point-to-point forwarding schemes, in some ad-hoc wireless mobile network applications.

Termination is one of the most important properties a distributed algorithm requires. Since it is imperative to not have unnecessary messages circulating and clogging the network, explicit termination is desired and often enforced by using a flag to record if the node has

already participated in the flooding [1, 14, 2, 16, 19]. Otherwise, algorithms using broadcast for communication e.g. [9] (for high speed networks) use other explicit solutions to enforce termination. However, in some models such as population protocols, the low memory makes termination very difficult to achieve leading to research that tries to provide termination e.g. [15]. Our flooding algorithm has the advantage of being simple, using low memory, and being efficiently terminating as shown by our analysis. The idea of avoiding the most recently chosen node(s) has been used before in distributed protocols e.g. in social networks [6] and broadcasting [7] but we are not aware of this fundamental variant of flooding having been studied before. Lastly, processes such as random walks [4, 5, 8, 12, 13] and its deterministic variant Rotor-Router (or Propp) machine [11, 3] can be seen as restricted variants of flooding which possibly our work can provide some insight into.

2 Termination in a synchronous network

▶ **Definition 2.** *Let G be a graph. The round-sets $R_0, R_1, \ldots$ are defined as:*

R_0 *is the singleton containing an initial node,*
R_i *is the set of nodes which receive a message at round i* $(i \geq 1)$.

Clearly, if $R_j = \emptyset$ for some $j \geq 0$, then $R_i = \emptyset$ for all $i \geq j$. We shall refer to rounds R_i, where $R_i \neq \emptyset$, as *active* rounds.

▶ **Theorem 3.** *Any node $g \in G$ is contained in at most two distinct round-sets.*

Proof. Define $\mathcal{R}$ to be the set of finite sequences of consecutive round-sets of the form:

$$\underline{R} = R_s, \ldots, R_{s+d} \quad \textit{where } s \geq 0,\ d > 0, \textit{ and } R_s \cap R_{s+d} \neq \emptyset\ . \tag{1}$$

In (1), s is the *start-point* $s(\underline{R})$ and d is the *duration* $d(\underline{R})$ of $\underline{R}$. Note that, a node $g \in G$ belonging to R_s and R_{s+d} may also belong to other R_i in (1). If a node $g \in G$ occurs in three different round-sets R_{i_1}, R_{i_2} and R_{i_3}, then the duration between R_{i_1} and R_{i_2}, the duration between R_{i_2} and R_{i_3}, or the duration between R_{i_1} and R_{i_3} will be even. Consider the subset $\mathcal{R}^{EV}$ of $\mathcal{R}$ of sequences of the form (1) where d is even. To prove that no node is in three round-sets, it suffices to prove that $\mathcal{R}^{EV}$ is empty.

We assume that $\mathcal{R}^{EV}$ is non-empty and derive a contradiction.

Let $\mathcal{R}^{EV}_{\hat{d}}$ be the subset of $\mathcal{R}^{EV}$ comprising sequences of minimum (even) duration $\hat{d}$, i.e.

$$\mathcal{R}^{EV}_{\hat{d}} = \{\underline{R} \in \mathcal{R}^{EV} \mid \forall\, \underline{R}' \in \mathcal{R}^{EV}.\ d(\underline{R}') \geq d(\underline{R}) = \hat{d}\} \tag{2}$$

Clearly, if $\mathcal{R}^{EV}$ is non-empty then so is $\mathcal{R}^{EV}_{\hat{d}}$. Let $\underline{R}^* \in \mathcal{R}^{EV}_{\hat{d}}$ be the sequence with earliest start-point $\hat{s}$, i.e.

$$\underline{R}^* = R_{\hat{s}}, \ldots, R_{\hat{s}+\hat{d}} \tag{3}$$

where

$$\forall\, \underline{R}' \in \mathcal{R}^{EV}_{\hat{d}}\ .\ s(\underline{R}') \geq s(\underline{R}^*) = \hat{s} \tag{4}$$

By (1), there exists $g \in R_{\hat{s}} \cap R_{\hat{s}+\hat{d}}$. Choose node g' which sends a message to g in round $\hat{s} + \hat{d}$. As g' is a neighbour of g, either g' sends a message to g in round $\hat{s}$ or g sends a message to g' in round $\hat{s} + 1$. We show that each of these cases leads to a contradiction.

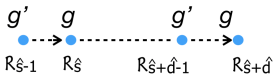


Figure 4 Node g' sends a message to node g in round $\hat{s}$: the first round of the minimum even length sequence (of length $\hat{d}$) in which g repeats.

Case (i): g' sends a message to g in round $\hat{s}$

Refer to Figure 4. In this case, there must be a round $\hat{s}-1$ which is either round 0 and g' is the initial node, or g' received a message in round $\hat{s}-1$. Thus, the sequence

$$\underline{R}^{*'} = R_{\hat{s}-1}, R_{\hat{s}}, \ldots, R_{\hat{s}+\hat{d}-1} \quad \textit{where } g' \in R_{\hat{s}-1} \cap R_{\hat{s}+\hat{d}-1} \tag{5}$$

has $d(\underline{R}^{*'}) = (\hat{s}+\hat{d}-1)-(\hat{s}-1) = \hat{d}$ which is even and so $\underline{R}^{*'} \in \mathcal{R}^{EV}_{\hat{d}}$. As $\underline{R}^{*'} \in \mathcal{R}^{EV}_{\hat{d}}$, by (4)

$$s(\underline{R}^{*'}) \geq s(\underline{R}^{*}) \tag{6}$$

But, from (5), $s(\underline{R}^{*'}) = \hat{s}-1$ and, from (4), $s(\underline{R}^{*}) = \hat{s}$. Thus, by (6),

$$\hat{s}-1 = s(\underline{R}^{*'}) \geq s(\underline{R}^{*}) = \hat{s}$$

which is a contradiction.

Case (ii): g sends a message to g' in round $\hat{s}+1$

Refer to Figure 5. By the definition of $\mathcal{R}^{EV}$, the smallest possible value of $\hat{d}$ is 2.

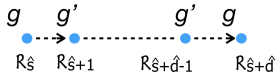


Figure 5 Node g sends a message to node g' in round $\hat{s}+1$: round $\hat{s}$ is the first round of the minimum even length sequence (of length $\hat{d}$) in which g repeats.

However, it is not possible to have $\hat{d}=2$ in this case as then

$$\underline{R}^{*} = R_{\hat{s}}, R_{\hat{s}+1}, R_{\hat{s}+2}$$

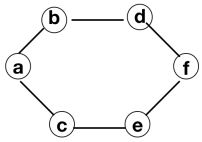
This would mean that g sends a message to g' in round $\hat{s}+1$. But, we chose g' to be such that g' sends a message to g in round $\hat{s}+\hat{d} = \hat{s}+2$. This cannot happen as g cannot send a message to g' and g' to g in consecutive rounds by the definition of rounds.

$$\underline{R}^{*} = R_{\hat{s}}, R_{\hat{s}+1}, \ldots, R_{\hat{s}+\hat{d}-1}, R_{\hat{s}+\hat{d}}$$

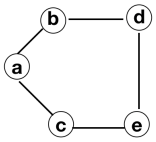
where $\hat{s}+1 < \hat{s}+\hat{d}-1$. Consider the sequence

$$\underline{R}^{*''} = R_{\hat{s}+1}, \ldots, R_{\hat{s}+\hat{d}-1} \tag{7}$$

As g' receives a message from g in round $\hat{s}+1$ and g' sends a message to g in round $\hat{s}+\hat{d}$, it is clear that $g' \in R_{\hat{s}+1} \cap R_{\hat{s}+\hat{d}-1}$. Thus, $\underline{R}^{*''} \in \mathcal{R}$. As $\hat{d}$ is even, so is



(a) Even Cycle Graph.



(b) Odd Cycle Graph.

Figure 6 Even cycle (6 nodes) and Odd cycle (5 nodes) graphs: Graphs show markedly different termination times. Consider AF from node b in both cases - In the 6-cycle it terminates in 3 rounds, but in the 5-cycle in 5 rounds.

$(\hat{s}+\hat{d}-1)-(\hat{s}+1)=\hat{d}-2$ and therefore $\underline{R}^{*''} \in \mathcal{R}^{EV}$. Now, $\underline{R}^{*} \in \mathcal{R}^{EV}_{\hat{d}}$ and so, as $\underline{R}^{*''} \in \mathcal{R}^{EV}$, we have, by (2),

$$d(\underline{R}^{*''}) \geq d(\underline{R}^{*}) \tag{8}$$

As $d(\underline{R}^{*''})=\hat{d}-2$ from (7) and $d(\underline{R}^{*})=\hat{d}$ from (3), we have, by (8),

$$\hat{d}-2=d(\underline{R}^{*''}) \geq d(\underline{R}^{*})=\hat{d}$$

This contradiction completes the proof. ◀

▶ **Definition 4.** *Given $g \in G$, we use a superscript 1 to indicate that g belongs to a round-set for the first time, and a superscript 2 to indicate that it belongs to a round-set for the second time, i.e.*

$$g^1 \in R_j$$

means that

$$g \in R_j \quad \textit{and} \quad g \notin R_i \quad \textit{for all } i \textit{ with } 0 \leq i < j.$$

and

$$g^2 \in R_j$$

means that

$$g \in R_j \quad \textit{and} \quad g \in R_i \quad \textit{for some } i \textit{ with } 0 \leq i < j.$$

Theorem 3 implies that $R_i = \emptyset$ for $i \geq 2n$, where n is the number of vertices of G, and therefore network flooding always terminates.

▶ **Corollary 5.** *Synchronous network flooding always terminates in fewer than $2n+1$ rounds.*

In the next section we give a greatly improved sharp upper bound for the number of rounds to termination, in terms of the eccentricity of the initial node and the diameter of G.

3 Time to termination

The question of termination of network flooding is non-trivial when cycles are present in G. The simple cases when G is an even cycle, as in Figure 6a and when G is an odd cycle, as in Figure 6b display quite different termination behaviours. The even cycle in Figure 6a

terminates after round e where e is the eccentricity of the initial node in G. On the other hand, flooding on the odd cycle in Figure 6b, returns a message to the initial node and terminates after round $2e+1$ resulting in a longer flooding process than the even cycle in Figure 6a despite having fewer nodes and a smaller value of e. In this section, we show that these observations can be largely generalized to arbitrary graphs. Specifically, we show that flooding on a graph G terminates after e rounds if and only if G is bipartite. If G is not bipartite, we show that flooding terminates after some round i where $e < i \leq e + d + 1$ and d is the diameter of G.

▶ **Definition 6.** *Let (G, E) be a graph with vertex set G and edge set E, and $g_0 \in G$ be an initial node. We will use the following definitions.*

(i) *For each $j \in \mathbb{N}$, the distance set D_j will denote the set of points which are a distance j from g_0. i.e.*

$$D_j = \{g \in G \ : \ d(g_0, g) = j\},$$

where d is the usual distance function in graph G.

(ii) *A node $g \in G$ is an equidistantly-connected node, abbreviated ec node, iff there there exists $g' \in G - \{g_0, g\}$ such that $d(g_0, g) = d(g_0, g')$ and $\{g, g'\} \in E$*

We have the following basic properties of distance sets D_j and *ec* nodes.

▶ **Lemma 7.** *Let G be a graph and $g_0 \in G$ be an initial node.*

(i) *For all $j \in \mathbb{N}$ and $i > j$, $D_j \subseteq R_j$ and $R_j \cap D_i = \emptyset$.*

(ii) *For all $j \in \mathbb{N}$, $g \in D_j$ and $g' \in D_{j+1}$ such that g and g' are neighbours, g sends a message to g' in round $j+1$, i.e. all nodes at a distance j from g_0 send to all their neighbours which are a distance $j+1$ in round $j+1$.*

(iii) *If $j \geq 1$ and $g \in D_j$ is an ec point, then $g^2 \in R_{j+1}$.*

Proof. For (i), clearly every node at a distance j from g_0 receives a message in round j and so $D_j \subseteq R_j$. Furthermore, every message received in round j will have travelled along j edges from g_0 and so could not have reached a node which is at a distance $i > j$ from g_0. Thus, $R_j \cap D_i = \emptyset$.

For (ii), we note that the only circumstance in which a node g in D_j ($\subseteq R_j$ by (i)) does not send to a neighbour g' in D_{j+1} in round $j+1$ is if g sent a message to g' in round j. This would need g to be in the round-set R_{j-1}, i.e. $g \in R_{j-1} \cap D_{j+1}$ which contradicts (i) which has $R_{j-1} \cap D_{j+1} = \emptyset$ as $j + 1 > j - 1$.

For (iii), if $j \geq 1$ and $g \in D_j$ is an *ec* point, then by Definition 6(ii) there is a point g' equidistant from the initial node g_0, i.e. $g' \in D_j$ such that g and g' are neighbours. By (i) of this lemma $D_j \subseteq R_j$, and so both g and g' receive messages in round j. Also by (i), neither sends a message in round j as $R_{j-1} \cap D_j = \emptyset$. Thus, g and g' send messages to each other in round $j+1$. As this will be the second time they receive messages we have that $g^2 \in R_{j+1}$. ◀

All nodes in a graph without *ec* nodes, belong to at most one round-set.

▶ **Lemma 8.** *Let G be a graph and let $g_0 \in G$ be an initial node. Then G has an ec node if and only if G has a node that is in two round-sets.*

Proof. Suppose that G has no *ec* nodes. Assume, on the contrary, that G has nodes that appear in two round-sets. Let R_j ($j \geq 1$) be the earliest round which contains a node g such that $g^2 \in R_j$ and $h \in R_{j-1}$ be a neighbour of g which sends to g in round j, so that

$h^1 \in R_{j-1}$. Then, $h \in D_i$ for some $i \geq 1$ and $h^1 \in R_i$ by Lemma i(i). Thus, $i = j - 1$ and so $g^2 \in R_{i+1}$. As g is a neighbour of h, $g \in D_i$, D_{i+1}, or D_{i-1}. If $g \in D_i$ then g and h are *ec* nodes contrary to our supposition that G has no *ec* nodes. If $g \in D_{i+1}$ then $g^1 \in R_{i+1}$ by Lemma i(i), which is contrary to the assertion that $g^2 \in R_j = R_{i+1}$. If $g \in D_{i-1}$ then $g \in R_{i-1}$, by Lemma i(i), and so $g^1 \in R_{i-1}$ as $g^2 \in R_{i+1}$. By Lemma ii(ii), g sends to h in round $i = j - 1$. This is contrary to h sending to g in round j. Thus, our assumption that G has nodes that appear in two round-sets is false.

Conversely, suppose that G has an *ec* node $g \in D_j$ where $j \geq 1$. Then $g^2 \in R_{j+1}$ by Lemma iii(iii). ◀

We note that bipartite graphs do not have any *ec* nodes.

▶ **Lemma 9.** *Let G be a graph and $g_0 \in G$ be an initial node. Then, G is bipartite iff it has no ec nodes.*

Proof. G is not bipartite iff there is a path from g_0 to an odd cycle in G. This is the case iff an edge of G connects two points equidistant from g_0, i.e G has an ec point. ◀

From Lemmas 8 and 9, we see that, in bipartite graphs, nodes only appear in one round-set. Thus, the time to termination can be determined by finding a bound on when each node belongs to a round-set.

▶ **Theorem 10.** *Let G be a graph and $g_0 \in G$ be an initial node with eccentricity e. Then, flooding will have terminated after round e if and only if G is bipartite.*

Proof.

G is bipartite	iff	G has no *ec* nodes	(by Lemma 9)
	iff	no node appears in 2 round-sets	(by Lemma 8)
	iff	R_e is the last non-empty round-set (as nodes a distance j from g_0 are only in R_j by Lemma i(i))	◀

To find the time to termination in general graphs we need to find a bound on when nodes can belong to a round-set for the second time. As nodes can only belong to at most two round-sets, by Theorem 3, this will give a bound for termination of flooding in general graphs. The following lemma relates the round-sets of second occurrences of neighbouring nodes.

▶ **Lemma 11.** *Let G be a graph and $g_0 \in G$ an initial node. If $h^2 \in R_j$ for some $j \in \mathbb{N}$, and if g is a neighbour of h, then*

$$g^2 \in R_{j-1} \text{ or } g^2 \in R_j \text{ or } g^2 \in R_{j+1}$$

Proof. Let i be the distance of h from g_0, i.e. $h \in D_i$. Then, as $h^2 \in R_j$, $j > i$ by Lemma i(i). As g is a neighbour of h, $g \in D_i$ or $g \in D_{i-1}$ or $g \in D_{i+1}$.

- *Case $g \in D_i$:* As $h, g \in D_i$ are neighbours they are both *ec* nodes. Thus, by Lemma iii(iii), $h^2 \in R_{i+1}$ and $g^2 \in R_{i+1}$. Therefore, $j = i + 1$ and $g^2 \in R_j$.
- *Case $g \in D_{i-1}$:* If $g \in R_j$ ($\neq R_{i-1}$ as $j > i$) then, as $g^1 \in D_{i-1} \subseteq R_{i-1}$ by Lemma i(i), it must be the case that $g^2 \in R_j$. If $g \notin R_j$ and $g \in R_{j-1}$ ($\neq R_{i-1}$ as $j > i$) then, as $g^1 \in R_{i-1}$ by Lemma i(i), it must be the case that $g^2 \in R_{j-1}$. If $g \notin R_j$ and $g \notin R_{j-1}$ then, as $h \in R_j$, h sends to g in round $j + 1$ and so $g \in R_{j+1}$ ($\neq R_{i-1}$ as $j > i$). As $g^1 \in R_{i-1}$, it must be the case that $g^2 \in R_{j+1}$.

- *Case $g \in D_{i+1}$, g does not send to h in round j:* In this case, as $h \in R_j$, h sends to g in round $j+1$. Thus, $g \in R_{j+1}$ ($\neq R_{i+1}$ as $j > i$) and therefore, as $g^1 \in D_{i+1} \subseteq R_{i+1}$ by Lemma i(i), it must be the case that $g^2 \in R_{j+1}$.
- *Case $g \in D_{i+1}$, g sends to h in round j:* In this case, $g \in R_{j-1}$. If $g^1 \in R_{j-1}$, then, by Lemma i(i) , $g^1 \in D_{i+1} \subseteq R_{i+1}$ and thus $j - 1 = i + 1$. Hence, by Lemma i(i), $h^1 \in D_i \subseteq R_i = R_{j-2}$. Also, $g \notin R_{j-3}$ as $g^1 \in R_{j-1}$.
  To summarize:

  $$g \notin R_{j-3}, \;\; h^1 \in R_{j-2}, \;\; g^1 \in R_{j-1}, \;\; h^2 \in R_j$$

  So, h sends to g in round $j-1$ and g sends to h in round j by the case assumption. This is a contradiction. Thus, $g^1 \notin R_{j-1}$ and, as $g \in R_{j-1}$, it follows that $g^2 \in R_{j-1}$.

This completes the proof. ◀

▶ **Theorem 12.** *Let G be a non-bipartite graph with diameter d and let $g_0 \in G$ be an initial node of eccentricity e. Then, flooding terminates after j rounds where j is in the range $e < j \leq e + d + 1$.*

Proof. If G is not bipartite it has an *ec* node g, by Lemma 9. By Lemma iii(iii), $g^2 \in R_k$ where $k = d(g_0, g) + 1$. Let h be an arbitrary node in G other than g. Then, there is a path

$$h_0 = g \longrightarrow h_1 \longrightarrow \ldots \longrightarrow h_l = h$$

where $l \leq d$. By repeated use of Lemma 11,

$$\begin{aligned}
&h_1^2 \in R_{j_1} \quad \text{where} \quad k - 1 \leq j_1 \leq k + 1,\\
&h_2^2 \in R_{j_2} \quad \text{where} \quad j_1 - 1 \leq j_2 \leq j_1 + 1,\\
&\ldots\\
&h_l^2 \in R_{j_l} \quad \text{where} \quad j_{l-1} - 1 \leq j_l \leq j_{l-1} + 1 \quad (l \geq 1).
\end{aligned}$$

Thus,

$$h_l^2 \in R_{j_l} \quad \textit{where} \quad k - l \leq j_l \leq k + l \tag{9}$$

Put $j = j_l$. From (19), as $k = d(g_o, g) + 1 \leq e + 1$ and as $l \leq d$,

$$h_l^2 \in R_j \quad \textit{where} \quad j \leq e + d + 1.$$

Thus, $j \leq e + d + 1$.
As G is not bipartite, $j > e$ by Theorem 10 and the proof is complete. ◀

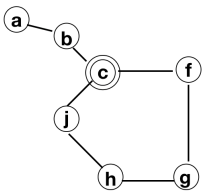
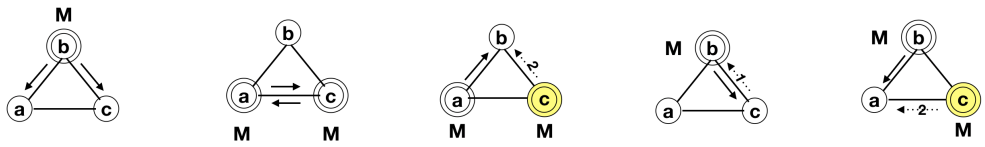


Figure 7 Flooding in the graph in the above figure starting from node c takes $e + d + 1$ rounds (the maximum as per our analysis), where e is eccentricity and d the diameter.

The upper bound in Theorem 12 is easily seen to be sharp - the flooding in the graph in Figure 7 starting from node c terminates after round $7 = 2 + 4 + 1 = e + d + 1$. Similar termination times hold for all nodes in the Petersen graph (Figure 1).



(a) Round 1 **(b)** Round 2 **(c)** Round 3 **(d)** Round 4 **(e)** Round 5

Figure 8 Asynchronous AF over a Triangle. Both node a and c send M to each other in round 2. In round 3, a sends M to b but the adversary makes c holds the message for one round (shaded node). In the next round, we have a round analogous to round 2 and so on.

4 Asynchronous Amnesiac Flooding

Non-termination in an adversarial asynchronous setting: Consider the *round-asynchronous* setting (as described in the model section). The scheduling adversary can choose the delay on every message edge i.e. which round to forward a message on.

An example suffices to prove non-termination. Consider round 3 in the triangle in Figure 8. The adversary delays M at node c but a continues and sends to b. In round 4, node b and c both send M so that the beginning of the next round is now identical to round 2 with nodes a and b interchanged. This process can now continue *ad infinitum* with the adversarial intervention.

5 Conclusion and Future Work

We studied a natural variant of the flooding algorithm where nodes do not retain any memory of the flooding beyond the previous round. We call this Amnesiac flooding (AF) and discussed the question of termination i.e. no copies of the initial message are being circulated anymore. We showed the surprising result that not only does this process terminate on all finite graphs but also accomplishes broadcast in almost optimal time and message overhead. There is a clear separation in complexity between bipartite and non-bipartite topologies. An interesting question is whether this separation can be exploited to devise distributed procedures to detect the topology of a graph given distance measures or vice versa. There is the question of multiple sources: what happens when multiple nodes start the flooding process with the same message M? We expect our method of proof can be extended to prove termination and obtain bounds in the case of multiple sources.

What about dynamic settings where nodes and edges change? It is easy to see that due to its simplicity, AF can be re-executed immediately after the graph has changed. However, what if the graph changes while messages are in circulation - under what conditions is termination/non-termination guaranteed?

Another important question is to look at flooding in asynchronous settings in more detail. We show one model where an adversary can force AF to be non-terminating. Since a completely asynchronous setting is event driven, this would also involve deciding what it means to receive messages simultaneously. Finally, one can see processes such as random walks, coalescing random walks and diffusion as probabilistic extremal variants of flooding. Are there any implications or connections of our result on these or intermediate probabilistic models? What about randomised variants of AF?

References

1 James Aspnes. Flooding, February 2019. URL: `http://www.cs.yale.edu/homes/aspnes/pinewiki/Flooding.html`.

2 Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. John Wiley & Sons, 2004.

3 JOSHUA N. COOPER and JOEL SPENCER. Simulating a random walk with constant error. *Combinatorics, Probability and Computing*, 15(6):815?822, 2006. `doi:10.1017/S0963548306007565`.

4 Atish Das Sarma, Danupon Nanongkai, and Gopal Pandurangan. Fast distributed random walks. In *PODC*, pages 161–170, 2009.

5 Atish Das Sarma, Danupon Nanongkai, Gopal Pandurangan, and Prasad Tetali. Efficient distributed random walks with applications. In *PODC*, 2010.

6 Benjamin Doerr, Mahmoud Fouz, and Tobias Friedrich. Social networks spread rumors in sublogarithmic time. *Electronic Notes in Discrete Mathematics*, 38:303–308, 2011. The Sixth European Conference on Combinatorics, Graph Theory and Applications, EuroComb 2011.

7 Robert Elsässer and Thomas Sauerwald. The power of memory in randomized broadcasting. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, pages 218–227, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

8 C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks: Algorithms and evaluation. *Performance Evaluation*, 63(3):241–263, 2006.

9 Ajei S. Gopal, Inder S. Gopal, and Shay Kutten. Fast broadcast in high-speed networks. *IEEE/ACM Trans. Netw.*, 7(2):262–275, 1999. `doi:10.1109/90.769773`.

10 Walter Hussak and Amitabh Trehan. On termination of a flooding process. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019.*, pages 153–155. ACM, 2019. `doi:10.1145/3293611.3331586`.

11 Adrian Kosowski and Dominik Pajak. Does adding more agents make a difference? A case study of cover time for the rotor-router. *J. Comput. Syst. Sci.*, 106:80–93, 2019. `doi:10.1016/j.jcss.2019.07.001`.

12 Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. On the complexity of universal leader election. *J. ACM*, 62(1):7:1–7:27, 2015. `doi:10.1145/2699440`.

13 Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. Sublinear bounds for randomized leader election. *Theoretical Computer Science*, 561(0):134–143, 2015. Special Issue on Distributed Computing and Networking. URL: `http://www.sciencedirect.com/science/article/pii/S0304397514001029`.

14 N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, San Mateo, CA, 1996.

15 Othon Michail and Paul G. Spirakis. Terminating population protocols via some minimal global knowledge assumptions. *Journal of Parallel and Distributed Computing*, 81-82:1–10, 2015. `doi:10.1016/j.jpdc.2015.02.005`.

16 David Peleg. *Distributed Computing: A Locality Sensitive Approach*. SIAM, 2000.

17 A. Rahman, W. Olesinski, and P. Gburzynski. Controlled flooding in wireless ad-hoc networks. In *In Proceedings of IWWAN'04*, pages 73–78, 2004.

18 Andrew Tanenbaum. *Computer networks*. Pearson Prentice Hall, Boston, 2011.

19 Gerard Tel. *Introduction to distributed algorithms*. Cambridge University Press, New York, NY, USA, 1994.

Generalised Pattern Matching Revisited

Bartłomiej Dudek
Institute of Computer Science, University of Wrocław, Poland
bartlomiej.dudek@cs.uni.wroc.pl

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland
gawry@cs.uni.wroc.pl

Tatiana Starikovskaya
DIENS, École normale supérieure, PSL Research University, Paris, France
tat.starikovskaya@gmail.com

Abstract

In the problem of Generalised Pattern Matching (GPM) [STOC'94, Muthukrishnan and Palem], we are given a text T of length n over an alphabet Σ_T, a pattern P of length m over an alphabet Σ_P, and a matching relationship $\subseteq \Sigma_T \times \Sigma_P$, and must return all substrings of T that match P (*reporting*) or the number of mismatches between each substring of T of length m and P (*counting*). In this work, we improve over all previously known algorithms for this problem:

- For $\mathcal{D}$ being the maximum number of characters that match a fixed character, we show two new Monte Carlo algorithms, a reporting algorithm with time $\mathcal{O}(\mathcal{D}\, n \log n \log m)$ and a $(1-\varepsilon)$-approximation counting algorithm with time $\mathcal{O}(\varepsilon^{-1}\mathcal{D}\, n \log n \log m)$. We then derive a $(1-\varepsilon)$-approximation deterministic counting algorithm for GPM with $\mathcal{O}(\varepsilon^{-2}\mathcal{D}\, n \log^6 n)$ time.
- For $\mathcal{S}$ being the number of pairs of matching characters, we demonstrate Monte Carlo algorithms for reporting and $(1-\varepsilon)$-approximate counting with running time $\mathcal{O}(\sqrt{\mathcal{S}}\, n \log m \sqrt{\log n})$ and $\mathcal{O}(\sqrt{\varepsilon^{-1}\mathcal{S}}\, n \log m \sqrt{\log n})$, respectively, as well as a $(1-\varepsilon)$-approximation deterministic algorithm for the counting variant of GPM with $\mathcal{O}(\varepsilon^{-1}\sqrt{\mathcal{S}} n \log^{7/2} n)$ time.
- Finally, for $\mathcal{I}$ being the total number of disjoint intervals of characters that match the m characters of the pattern P, we show that both the reporting and the counting variants of GPM can be solved exactly and deterministically in $\mathcal{O}(n\sqrt{\mathcal{I} \log m} + n \log n)$ time.

At the heart of our new deterministic upper bounds for $\mathcal{D}$ and $\mathcal{S}$ lies a faster construction of superimposed codes, which solves an open problem posed in [FOCS'97, Indyk] and can be of independent interest.

To conclude, we demonstrate first lower bounds for GPM. We start by showing that any deterministic or Monte Carlo algorithm for GPM must use $\Omega(\mathcal{S})$ time, and then proceed to show higher lower bounds for combinatorial algorithms. These bounds show that our algorithms are almost optimal, unless a radically new approach is developed.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases pattern matching, superimposed codes, conditional lower bounds

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.18

Related Version A full version of the paper is available at `https://arxiv.org/abs/2001.05976`.

Funding *Bartłomiej Dudek*: partially supported by the National Science Centre, Poland, grant number 2017/27/N/ST6/02719.

1 Introduction

Processing noisy data is a keystone of modern string processing. One possible approach to address this challenge is approximate pattern matching, where the task is to find all substrings of the text that are close to the pattern under some similarity measure, such as

Hamming or edit distance. The approximate pattern matching approach assumes that noise is arbitrary, i.e. that we can delete or replace any character of the pattern or of the text by any other character of the alphabet.

The assumption that the noise is completely arbitrary is not necessarily justified, as in practice we might have some predetermined knowledge about the structure of the errors. In this paper we focus on the Generalised Pattern Matching (GPM) problem that addresses this setting. We assume to be given a text T over an alphabet Σ_T, a pattern P over an alphabet Σ_P, and we allow each character of Σ_T to match a subset of characters of Σ_P. We must report all substrings of the text that match the pattern. This problem was introduced in STOC'94 [35] by Muthukrishnan and Palem to provide a unified approach for solving different extensions of the classical pattern matching question that has been considered as separate problems in the early 90s. Later, Muthukrishnan [34] considered a *counting variant* of GPM, where the task is to count the number of mismatches between substrings of the text and the pattern. Formally, the problem is defined as follows:

Generalised Pattern Matching (GPM)
Input: A text $T \in (\Sigma_T)^n$, a pattern $P \in (\Sigma_P)^m$, and a matching relationship $\subseteq \Sigma_T \times \Sigma_P$.
Output (Reporting): All $i \in [n-m+1]$ such that $T[i, i+m-1]$ matches P.
Output (Counting): For each $i \in [n-m+1]$, the number of positions $j \in [m]$ such that $T[i+j-1]$ does not match $P[j]$.

Muthukrishnan and Palem [35] and subsequent work [34, 36] considered three natural parameters describing the matching relationship ($\mathcal{D}, \mathcal{S}$) or the pattern ($\mathcal{I}$). Viewing the matching relationship as a bipartite graph with edges connecting pairs of matching characters from $\Sigma_T \times \Sigma_P$, $\mathcal{D}$ is the maximum degree of a node and $\mathcal{S}$ is the total number of edges in the graph. Next, the parameter $\mathcal{I}$ describes the pattern rather than the matching relationship. For each character $a \in \Sigma_P$, let $I(a)$ be the minimal set of disjoint sorted intervals that contain the characters that match a, and define $\mathcal{I} = \sum_{j \in [m]} |I(P[j])|$.

The maximum number of characters that match a fixed character, $\mathcal{D}$. For the reporting variant of GPM, Muthukrishnan [34] showed a Las Vegas algorithm with running time $\mathcal{O}(\mathcal{D}\, n \log n \log m)$. Indyk [27] used superimposed codes to show a deterministic algorithm with running time $\mathcal{O}(|\Sigma_P|\mathcal{D}^2 \log^2 n + \mathcal{D}\, n \log^3 n \log m)$. For the counting variant, Muthukrishnan [34] showed a $(\log m)$-approximation Las Vegas algorithm with time $\mathcal{O}(\mathcal{D}\, n \log n \log m)$. Indyk [27] gave a $(1-\varepsilon)$-approximation deterministic and Monte Carlo algorithm with running time $\mathcal{O}(\varepsilon^{-2}\mathcal{D}^2 n \log^3 n)$ and $\mathcal{O}(\varepsilon^{-2}\mathcal{D}\, n \log^3 n)$, respectively.

The number of matching pairs of characters, $\mathcal{S}$. Muthukrishnan and Ramesh [36] gave an $\mathcal{O}((\mathcal{S}\, m \log^2 m)^{1/3} n)$-time algorithm for the reporting variant of GPM.

The number of intervals of matching characters, $\mathcal{I}$. For this parameter, Muthukrishnan [34] gave an $\mathcal{O}(\mathcal{I} + (m\mathcal{I})^{1/3} n\sqrt{\log m})$-time algorithm[1].

1.1 Our Contribution

We improve existing randomised and deterministic upper bounds for GPM, and demonstrate matching lower bounds. At heart of our deterministic algorithms for the counting variant of GPM is a solution to an open problem of Indyk [27] on construction of superimposed codes.

[1] [34, Theorem 9] claims $\mathcal{O}(n + \mathcal{I} + \mathcal{I}^{1/3}(nm)^{2/3}\sqrt{\log m})$, but the first sentence of the proof states that for $n \le 2m$ the algorithm takes $\mathcal{O}(\mathcal{I} + \mathcal{I}^{1/3} m^{4/3}\sqrt{\log m})$ time, where the first term is the time that we need to read the input. For a longer text, one needs to apply it n/m times for overlapping blocks of length $2m$, making the total time $\mathcal{O}(\mathcal{I} + n/m \cdot \mathcal{I}^{1/3} m^{4/3}\sqrt{\log m}) = \mathcal{O}(\mathcal{I} + (m\mathcal{I})^{1/3} n\sqrt{\log m})$.

Data-dependent superimposed codes. A z-superimposed code is a set of binary vectors such that no vector is contained in a Boolean sum (i.e. bitwise OR) of z other vectors. Superimposed codes find their main application in information retrieval (e.g. in compressed representation of document attributes), and optimizing broadcasting on radio networks [30], and have also proved to be useful in graph algorithms [1, 25]. Indyk [27] extended the notion of superimposed codes to the so-called *data-dependent superimposed codes*, and asked for a deterministic construction for such codes with a certain additional property that makes them useful for counting mismatches (see Section 2 for a formal definition). We provide such a construction algorithm in Theorem 10. We briefly describe the high-level idea below.

We need the concept of discrepancy minimization. Given a universe U, each of its elements is assigned one of two colours, red or blue. The *discrepancy* of a subset of U is defined as the difference between the number of red and blue elements in it, and the discrepancy of a family $\mathcal{F}$ of subsets is defined as the maximum of the absolute values of discrepancies of the subsets in $\mathcal{F}$. Discrepancy minimization is a fundamental notion with numerous applications, including derandomization, computational geometry, numerical integration, understanding the limits of models of computation, and so on (see e.g. [13]). A recent line of work showed a series of algorithms for constructing colourings of low discrepancy in various settings [5–10, 32, 33]. For our applications, we need to work under the assumption that the size of each subset in $\mathcal{F}$ is bounded by a given parameter k. In Theorem 7, we describe a fast deterministic algorithm that returns a colouring of small discrepancy for this case. We follow the algorithm described by Chazelle [13] that can be roughly summarized as based on the method of conditional expectations tweaked as to allow for an efficient implementation. In more detail, Chazelle's construction assumes infinite precision of computation and does not immediately translate into an efficient algorithm working in the Word RAM model of computation, thus requiring resolving some technical issues to bound the required precision and the overall complexity.

We apply discrepancy minimization to design in Lemma 9 a procedure that, given a family $\mathcal{F}$ of subsets of U, partitions the universe U into not too many parts such that the intersection of each part and each of the subsets in $\mathcal{F}$ is small. The procedure follows the natural idea of colouring the universe with two colours, and then recursing on the elements with the same colour. Every step of such construction introduces some penalty that needs to be carefully controlled as to guarantee the desired property in the end. Because of this penalty, we are only able to guarantee that the intersections are small, but not constant. To finish the construction, we combine the partition with a hash function into the ring of polynomials. We stress that this part of the construction is new and not simply a modification of Chazelle's (or Indyk's) method.

Upper bounds for GPM. Similar to previous work, we assume that the alphabets' sizes are polynomial in n and that the matching relationship is given as a graph M on the set of vertices $\Sigma_T \cup \Sigma_P$. We also assume to have access to three oracles that can answer the following questions in $\mathcal{O}(1)$ time:

1. Is there an edge between $a \in \Sigma_T$ and $b \in \Sigma_P$ (in other words, do a and b match)?
2. What is the degree of a character $a \in \Sigma_T$ or $b \in \Sigma_P$ (in other words, what is the number of characters that match a given character)?
3. What is the k-th neighbor of $a \in \Sigma_T$ (in other words, what is the k-th character $b \in \Sigma_P$ matching a)? We assume an arbitrary (but fixed) order of neighbors of every node.

Under these assumptions, we show the following upper bounds summarized in Tables 1 and 2:

Table 1 Generalised Pattern Matching (reporting).

Time	Det./Rand.	
$\mathcal{O}(\lvert\Sigma_P\rvert\mathcal{D}^2\log^2 n + \mathcal{D}\, n\log^3 n\log m)$	Det.	[27]
$\mathcal{O}(\mathcal{D}\, n\log^6 n)$	Det.	This work
$\mathcal{O}(\mathcal{D}\, n\log n\log m)$	Rand.	[34]
$\mathcal{O}(\mathcal{D}\, n\log n\log m)$	Rand.	This work
$\mathcal{O}((\mathcal{S}\, m\log^2 m)^{1/3}n)$	Det.	[36]
$\mathcal{O}(\sqrt{\mathcal{S}}\, n\log^{7/2} n)$	Det.	This work
$\mathcal{O}(\sqrt{\mathcal{S}}\, n\log m\sqrt{\log n})$	Rand.	This work
$\mathcal{O}(\mathcal{I} + (m\mathcal{I})^{1/3}n\sqrt{\log m})$	Det.	[34]
$\mathcal{O}(n\sqrt{\mathcal{I}\log m} + n\log n)$	Det.	This work

Table 2 Generalised Pattern Matching (counting).

Time	Det./Rand.	Approx. factor	
$\mathcal{O}(\varepsilon^{-2}\mathcal{D}^2 n\log^3 n)$	Det.	$(1-\varepsilon)$	[27]
$\mathcal{O}(\varepsilon^{-2}\mathcal{D}\, n\log^6 n)$	Det.	$(1-\varepsilon)$	This work
$\mathcal{O}(\mathcal{D}\, n\log n\log m)$	Rand.	$\log m$	[34]
$\mathcal{O}(\varepsilon^{-2}\mathcal{D}\, n\log^3 n)$	Rand.	$(1-\varepsilon)$	[27]
$\mathcal{O}(\varepsilon^{-1}\mathcal{D}\, n\log n\log m)$	Rand.	$(1-\varepsilon)$	This work
$\mathcal{O}(\varepsilon^{-1}\sqrt{\mathcal{S}}\, n\log^{7/2} n)$	Det.	$(1-\varepsilon)$	This work
$\mathcal{O}(\sqrt{\varepsilon^{-1}\mathcal{S}}\, n\log m\sqrt{\log n})$	Rand.	$(1-\varepsilon)$	This work
$\mathcal{O}(\mathcal{I} + (m\mathcal{I})^{1/3}n\sqrt{\log m})$	Det.	–	[34]
$\mathcal{O}(n\sqrt{\mathcal{I}\log m} + n\log n)$	Det.	–	This work

1. We start by showing a new Monte Carlo algorithm for the parameter $\mathcal{D}$ with running time $\mathcal{O}(\mathcal{D}\, n\log m\log n)$ (Theorem 11). While its running time is the same as that of [34], it encapsulates a novel approach to the problem that serves as a basis for other algorithms. We then derive a Monte Carlo algorithm for the parameter $\mathcal{S}$ with running time $\mathcal{O}(\sqrt{\mathcal{S}}\, n\log m\sqrt{\log n})$ (Theorem 12). As a corollary, we show a $(1-\varepsilon)$-approximation Monte Carlo algorithm that solves the counting variant of GPM in time $\mathcal{O}(\min\{\varepsilon^{-1}\mathcal{D}\log n, \sqrt{\varepsilon^{-1}\mathcal{S}\log n}\}\cdot n\log m)$ (Corollary 13). All three algorithms have inverse-polynomial error probability.
2. Next, using the data-dependent superimposed codes, we construct $(1-\varepsilon)$-approximation deterministic algorithms for the counting variant of GPM. The first algorithm requires $\mathcal{O}(\varepsilon^{-2}\mathcal{D}\, n\log^6 n)$ time (Theorem 14), and the second algorithm $\mathcal{O}(\varepsilon^{-1}\sqrt{\mathcal{S}}\, n\log^{7/2} n)$ time (Theorem 15). By taking $\varepsilon = 1/2$, we immediately obtain deterministic algorithms for the reporting variant of the problem with the same complexities.
3. Finally, we show that both the reporting and the counting variants of GPM can be solved exactly and deterministically in $\mathcal{O}(n\sqrt{\mathcal{I}\log m} + n\log n)$ time (Theorem 17).

Lower bounds for GPM. We also show first lower bounds for GPM (see Section 4). We start with a simple adversary-based argument that shows that any deterministic algorithm or any Monte Carlo algorithm with constant error probability that solves GPM must use $\Omega(\mathcal{S})$ time (Lemma 19 and 20). We then proceed to show higher lower bounds for combinatorial

algorithms by reduction from Boolean matrix multiplication[2] parameterized by $\mathcal{D}, \mathcal{S}, \mathcal{I}$ (Lemma 21 and Corollary 23). All the lower bounds are presented for the reporting variant of GPM, so they immediately apply also to the counting variant. These bounds show that our algorithms are almost optimal, unless a radically new approach is developed.

1.2 Related Work

Degenerate string matching. A more general approach to dealing with noise in string data is degenerate string matching, where the set of matching characters is specified for every position of the text or of the pattern (as opposed to every character of the alphabets). Abrahamson [3] showed the first efficient algorithm for a degenerate pattern and a standard text. Later, several practically efficient algorithms were shown [26, 37].

Pattern matching with don't cares. In this problem, we assume $\Sigma_T = \Sigma_P = \Sigma$, where Σ contains a special character – "don't care". We assume that two characters of Σ match if either one of them is the don't care character, or they are equal. The study of this problem commenced in [21], where a $\mathcal{O}(n \log m \log |\Sigma|)$-time algorithm was presented. The time complexity of the algorithm was improved in subsequent work [18, 28, 29], culminating in an elegant $\mathcal{O}(n \log m)$-time deterministic algorithm of Clifford and Clifford [15]. Clifford and Porat [17] also considered the problem of identifying all alignments where the number of mismatching characters is at most k.

Threshold pattern matching. In the threshold pattern matching problem, we are given a parameter δ, and we say that two characters a, b match if $|a - b| < \delta$. The threshold pattern matching problem has been studied both in reporting and counting variants [4, 11, 12, 16, 19, 20, 22, 38]. The best algorithm for the reporting variant of the threshold pattern matching problem is deterministic and takes linear time (after the pattern has been preprocessed). The best deterministic algorithm for the counting variant of threshold pattern matching has time $\mathcal{O}((\log \delta + 1)n\sqrt{m \log m})$, while the best randomised algorithm has time $\mathcal{O}((\log \delta + 1)n \log m)$ [38].

In threshold pattern matching the matching relationship is described with a single interval per character, so $\mathcal{I} = m$. Hence from Theorem 17 immediately follows a faster deterministic algorithm for the counting variant of the threshold pattern matching problem (Corollary 18).

2 Data-Dependent Superimposed Codes

We start by solving the open problem posed by Indyk [27]: provide a deterministic algorithm for construction of a variant of data-dependent superimposed codes that is particularly suitable for the counting variant of GPM. The algorithm that we present is rather involved, a reader more interested in pattern matching applications can skip this section on the first reading.

▶ **Definition 1.** *Let $S_1, \ldots, S_z$ be subsets of a universe U. A family of sets $\mathcal{C} = \{C_1, \ldots, C_{|U|}\}$, where $C_u \subseteq [\ell]$ and $|C_u| = w$ for $u \in U$ is called an $(\{S_i\}, \tau)$-superimposed code if for every S_i and $u \notin S_i$ we have $|C_u - \bigcup_{v \in S_i} C_v| \geq \tau$. We call ℓ and w respectively the length and the weight of the code $\mathcal{C}$.*

[2] It is not clear what combinatorial means precisely. However, FFT and Boolean convolution often used in algorithms on strings are considered not to be combinatorial.

Suppose that the size of each S_i is at most k, where k is some fixed integer. Indyk asked if there exists a deterministic $\tilde{\mathcal{O}}((zk)/\varepsilon^{\mathcal{O}(1)})$-time algorithm that computes an $(\{S_i\}, (1-\varepsilon)w)$-superimposed code of some weight w and length $\ell = \mathcal{O}(k \operatorname{polylog}(zk))$. It can be seen that we cannot hope to construct such a code with ℓ independent of ε. In the following lemma we show that even if we restrict to the case of $k = 1$ we still need that ℓ significantly depends on ε.

▶ **Lemma 2.** *For every constant $\delta \in (0,1)$, function $f(z) = \mathcal{O}(\operatorname{polylog} z)$, and z large enough, there exists a family of singleton sets $S_1, S_2, \ldots, S_z$ and $0 < \varepsilon < 1$ such that any $(\{S_i\}, (1-\varepsilon)w)$-superimposed code of weight w must have length length $\ell > f(z)/\varepsilon^{\delta}$.*

Proof. Consider sets $S_i = \{i\}$ for $i \in [z]$, where z will be determined later. Let $\varepsilon = 1/(2f(z))^{\frac{1}{1-\delta}}$ and suppose that there is a $(\{S_i\}, (1-\varepsilon)w)$-superimposed code $\mathcal{C}$. Then, by definition of superimposed codes and from $w \leq \ell$, for $i \neq j$ it holds

$$\begin{cases} |C_i - C_j| \geq (1-\varepsilon)w = w - \varepsilon w \geq w - \varepsilon\ell, \\ \varepsilon\ell \leq \varepsilon f(z)/\varepsilon^{\delta} = \varepsilon^{1-\delta} f(z) = 1/2 \end{cases}$$

so $|C_i - C_j| > w - 1$. Hence, $|C_i - C_j| = w$ and every C_i and C_j must be disjoint, and therefore $\ell \geq zw \geq z$. Assume towards a contradiction that $\ell \leq f(z)/\varepsilon^{\delta}$. We obtain

$$\ell \leq f(z)/\varepsilon^{\delta} = f(z) \cdot (2f(z))^{\frac{\delta}{1-\delta}} = f(z)^{\frac{1}{1-\delta}} \cdot 2^{\frac{\delta}{1-\delta}} = \mathcal{O}(\operatorname{polylog} z) \cdot 2^{\frac{\delta}{1-\delta}} < z$$

where the last inequality holds for sufficiently large z. This leads to a contradiction and the claim follows. ◀

Therefore, one should allow $\ell = \mathcal{O}(k \operatorname{polylog}(zk)/\varepsilon^{\mathcal{O}(1)})$. We give a positive answer to this natural relaxation. We start by showing an efficient deterministic algorithm for discrepancy minimization that will play an essential role in our approach.

2.1 Discrepancy Minimization

Let us start with a formal definition of discrepancy.

▶ **Definition 3** (Discrepancy). *Consider a family $\mathcal{F}$ of z sets $S_i \subseteq U$, $i \in [z]$. We call a function $\chi : U \to \{-1, +1\}$* a colouring. *The* discrepancy *of a set S_i is defined as $\chi(S_i) = \sum_{u \in S_i} \chi(u)$, and the* discrepancy *of $\mathcal{F}$ is defined as $\max_{i \in [z]} |\chi(S_i)|$.*

In [13, Section 1.1], Chazelle presented a construction of a colouring of small discrepancy assuming infinite precision of computation. Our deterministic algorithm will follow the outline of this construction (although crucial modifications are required in order to overcome the infinite precision assumption), so we quickly restate Chazelle's construction below. The main idea is to assign colours so as to minimize the value of an objective function $G = G(\chi, \{S_i\})$ defined as follows: let ε be chosen so that $\log \frac{1+\varepsilon}{1-\varepsilon} = \alpha \cdot \sqrt{\log(3z)/k}$ for some constant $\alpha > 2$, and let p_i (respectively, n_i) be the number of $u \in S_i$ such that $\chi(u) = +1$ (respectively, $\chi(u) = -1$) for $i \in [z]$. Define

$$G_i = (1+\varepsilon)^{p_i}(1-\varepsilon)^{n_i} + (1+\varepsilon)^{n_i}(1-\varepsilon)^{p_i} \quad \text{and} \quad G = \sum_{i \in [z]} G_i$$

Chazelle's construction assigns colours to one element of U at a time, without ever backtracking. To assign a colour to an element u, it performs the following three simple steps. First, it computes G^+, the value of G assuming $\chi(u) = +1$. Second, it computes G^-, the

value of G assuming $\chi(u) = -1$. Finally, if $G^+ \leq G^-$, it sets $\chi(u) = +1$ and $G = G^+$, and otherwise it sets $\chi(u) = -1$ and $G = G^-$. Note that for each $i \in [z]$, we have

$$(1+\varepsilon)^{p_i+1}(1-\varepsilon)^{n_i} + (1+\varepsilon)^{n_i}(1-\varepsilon)^{p_i+1} + (1+\varepsilon)^{p_i}(1-\varepsilon)^{n_i+1} + (1+\varepsilon)^{n_i+1}(1-\varepsilon)^{p_i} = 2 \cdot ((1+\varepsilon)^{p_i}(1-\varepsilon)^{n_i} + (1+\varepsilon)^{n_i}(1-\varepsilon)^{p_i})$$

and therefore the value of G can only decrease. This implies an important property of Chazelle's construction: since at initialization we have $n_i = p_i = 0$ for all $i \in [z]$ and therefore $G = 2z$, we have $G_i \leq G \leq 2z$ for $i \in [z]$ at any moment of the construction. Let us show that small values of G_i's imply small discrepancy. In order to do this, we follow the outline of [13], but use a slightly higher bound for G_i's to be able to apply this lemma later.

▶ **Lemma 4** ([13]). *If after all elements of U have been assigned a colour we have $G_i \leq 3z$ for all $i \in [z]$, then the discrepancy of the resulting colouring is at most $\alpha \cdot \sqrt{k \log(3z)}$ for any constant $\alpha > 2$.*

We will show a deterministic algorithm that computes a colouring for which the values G_i are bounded by $3z$. By Lemma 4, we therefore obtain that the discrepancy is bounded by $\alpha \cdot \sqrt{k \log(3z)}$. We must overcome several crucial issues: first, we must explain how to compute ε. Second, we must design an algorithm that uses only multiplications and additions so as to be able to control the accumulated precision error. And finally, we must explain how to remove the assumption of infinite precision and to ensure that we never operate on numbers that are too small.

▶ **Proposition 5.** *Assume $k > \log(3z)$. There is a deterministic algorithm that computes $\varepsilon \in (0,1)$ such that $\log \frac{1+\varepsilon}{1-\varepsilon} = \alpha \cdot \sqrt{\log(3z)/k}$ for some constant $\alpha > 2$ in $\mathcal{O}(\log(zk))$ time. Both ε and $1-\varepsilon$ are bounded from below by $1/(kz)^{\mathcal{O}(1)}$.*

We can implement Chazelle's construction to use only multiplications and additions via segment trees.

▶ **Proposition 6.** *Assume that $(1+\varepsilon)$ and $(1-\varepsilon)$ are known. Chazelle's construction can be implemented via $\mathcal{O}(zk \log z)$ addition and multiplication operations.*

Proof. We maintain a complete binary tree on top of $\{1, 2, \ldots, 2^t\}$, where $2^{t-1} < z \leq 2^t$. At any moment, the $(2i-1)$-th leaf stores $(1+\varepsilon)^{p_i}(1-\varepsilon)^{n_i}$ and the $(2i)$-th leaf stores $(1+\varepsilon)^{n_i}(1-\varepsilon)^{p_i}$ for all $i \in [z]$, while all the other leaves store value 0. Each internal node stores the sum of the values in the leaves of its subtree. In particular, the root stores the value G. To update G after setting $\chi(u)$ for $u \in U$, we must update the values stored in the $(2i-1)$-th and $(2i)$-th leaves for all i such that $u \in S_i$, as well as the sums in the $\mathcal{O}(\log z)$ internal nodes above these leaves. For each leaf, we use one multiplication operation (we must multiply the value by $(1+\varepsilon)$ or $(1-\varepsilon)$ as appropriate), and for each internal node we use one addition operation. In total, we need $\mathcal{O}(\sum_{i \in [z]} |S_i| \log z) = \mathcal{O}(zk \log z)$ addition and multiplication operations. ◀

We are now ready to remove the infinite precision assumption and to show the final result of this section. Our algorithm will follow the outline of Proposition 6, but the addition and the multiplication operations will be implemented with precision Δ. Moreover, we will guarantee that the algorithm only works with values in $[\Delta, \mathcal{O}(z)]$, which will imply that both arithmetic operations can be performed in constant time and that the algorithm takes $\mathcal{O}(zk \log z)$ time.

▶ **Theorem 7.** *Given a family of z sets $S_i \subseteq U$ where $|S_i| \leq k$ and $|U| = zk$, one can find deterministically in $\mathcal{O}(zk \log z)$ time a colouring $\chi : U \to \{-1, +1\}$ such that $\max_{i \in [z]} |\chi(S_i)| \leq \alpha \cdot \sqrt{k \log(3z)}$ for some constant $\alpha > 2$.*

Theorem 7 can be used to partition the universe U into a small number of subsets such that the intersection of every subset of the partition and every set S_i is small. We start with a simple technical lemma.

▶ **Lemma 8.** *Consider a process that starts with $x_0 = x$, and keeps computing $x_{i+1} := \lfloor x_i(1/2 + 1/\sqrt{x_i}) \rfloor$ as long as $x_i > 4$. The process ends after at most $\log x + \mathcal{O}(\log^* x)$ steps.*

▶ **Lemma 9.** *Given a family of z sets $S_i \subseteq U$ where $|S_i| \leq k$ and $|U| = zk$, one can construct deterministically in $\mathcal{O}(|U| \log z \log k)$ time a function $f : U \to [k \cdot 2^{\mathcal{O}(\log^* k)}]$ such that for each $c \in [k \cdot 2^{\mathcal{O}(\log^* k)}]$ and for each S_i, the intersection of $\{u \in U \mid f(u) = c\}$ and S_i contains $\mathcal{O}(\log z)$ elements.*

Proof. We can reformulate the statement of the lemma as follows. We must show that there is a partitioning of U into subsets $X_c = \{u \in U : f(u) = c\}$ such that for every S_i, the intersection $X_c \cap S_i$ has size at most $\mathcal{O}(\log z)$.

We partition U recursively using the procedure from Theorem 7. We start with a single set $X = U$. Suppose that after several steps we have a partitioning of U into sets X_c such that $|S_i \cap X_c| \leq y$ for all i and c and some integer y. We then apply Theorem 7 to the sets X_c. Using the colouring output by the lemma, we partition each set X_c into sets X_{c_0} and X_{c_1}, where the former contains all the elements of X_c of colour -1 and the latter all the elements of X_c of colour $+1$. For $j \in \{0, 1\}$ we choose c_j (and also the value of $f(x)$ for $x \in X_{c_j}$) so that its binary representation equals the binary representation of c appended with j. By Theorem 7, there is a constant α such that

$$|S_i \cap X_{c_0}|, |S_i \cap X_{c_1}| \leq y/2 + \frac{1}{2}\alpha \cdot \sqrt{y \log(3z)} \leq y(1/2 + 1/\sqrt{y/\alpha^2 \log(3z)}).$$

We continue this process until $|S_i \cap X_c| \leq 4\alpha^2 \log(3z)$ for all i and c.

It remains to bound the number of iterations. By setting $x = k/\alpha^2 \log(3z)$ in Lemma 8, we obtain that we need at most $\log x + \mathcal{O}(\log^* x) \leq \log k + \mathcal{O}(1) + \mathcal{O}(\log^* k) = \log k + \mathcal{O}(\log^* k) = t$ recursive applications of the partition procedure implemented with Theorem 7 to ensure that every set S_i has at most $4\alpha^2 \log(3z) = \mathcal{O}(\log z)$ elements in common with every X_c. Therefore, the size of the image of f is bounded by $2^t = k2^{\mathcal{O}(\log^* k)}$. The overall construction time is $\mathcal{O}(|U| \log z \log k)$. ◀

2.2 Superimposed Codes

We are now ready to show an efficient construction algorithm for data-dependent superimposed codes (see Definition 1). At a high level, we will construct a family of functions which, combined with the partition f from Lemma 9, will give us the superimposed code.

▶ **Theorem 10.** *Given a family of z sets $S_i \subseteq U$ where $|S_i| \leq k$ and $|U| = zk$, one can construct an $(\{S_i\}, (1 - \varepsilon)w)$-superimposed code of weight $w = \mathcal{O}(\varepsilon^{-1} \log^2 |U|)$ and $\ell = \mathcal{O}(\varepsilon^{-2} k \log^5 |U|)$ in $\mathcal{O}(\varepsilon^{-1}|U| \log^2 |U|)$ time and space.*

Proof. By applying Lemma 9, we obtain in $\mathcal{O}(|U| \log z \log k) = \mathcal{O}(|U| \log^2 |U|)$ time a function $f : U \to [k \cdot 2^{\mathcal{O}(\log^* k)}]$ which gives a partitioning of U into subsets $X_c = \{u \in U \mid f(u) = c\}$, such that for some constant α, for every c and i holds $|X_c \cap S_i| \leq \alpha \log z$.

Consider the ring of polynomials $\mathbb{Z}_2[x]$. Let $U = \{u_1, u_2, \ldots, u_{zk}\}$. We define a mapping $\textsc{pol} : U \to \mathbb{Z}_2[x]$ as follows. Let $u = u_q$ and $q = \overline{q_t q_{t-1} \ldots q_0}$ be the binary representation of q, where $t = \lfloor \log |U| \rfloor$, then $\textsc{pol}(u) = \sum_{i=0}^{t} q_i x^i$.

Let $\mathcal{H}(U, d)$ be the family of functions $h_p : U \to \mathbb{F}_{2^d}$ of the form $h_p(u) = (\textsc{pol}(u) \bmod p)$ for all irreducible polynomials p of degree d. By Gauss's formula [14, 23], there are $\Theta(2^d/d)$ irreducible polynomials of degree d over $\mathbb{Z}_2$, and so is the size of the family $\mathcal{H}(U, d)$. Consider two distinct polynomials x, y of degree t. Observe that there are at most t/d irreducible polynomials p that hash both x and y to the same value $h_p(x) = h_p(y)$, because $\mathbb{Z}_2[x]$ is a unique factorization domain [23]. We choose d in such a way that the probability that x, y are hashed to the same value while choosing a hash function uniformly at random from $\mathcal{H}(U, d)$ is bounded by $\varepsilon/(\alpha \log z)$: $\frac{t/d}{\Theta(2^d/d)} \leq \frac{\varepsilon}{\alpha \log z}$ and hence we can choose $d = \Theta(\log \frac{t \log z}{\varepsilon})$.

If $d > t$, then $\varepsilon < \frac{\log^2 |U|}{|U|}$ and we can take $\ell = |U|, w = 1$ and set $C_{u_q} = \{q\}$. From now on, assume $d \leq t$. Let f be as in Lemma 9. Consider $u \in U$ such that $u \in X_c$, where $c = f(u) \in [k \cdot 2^{\mathcal{O}(\log^* k)}]$. We define C_u as follows:

$$C_u = \{H_p(u) = \textsc{num}(h_p(u)) + 2^d \cdot \textsc{num}(p) + 4^d \cdot c \mid h_p \in \mathcal{H}(U, d)\},$$

where the mapping $\textsc{num}(q)$ treats a polynomial $q = \sum_{i=0}^{d-1} q_i x^i$ as a d-bit number $\overline{q_{d-1} \ldots q_0}$. Clearly, $w = |C_u| = \mathcal{O}(2^d/d) = \mathcal{O}(2^d) = \mathcal{O}(\frac{t \log z}{\varepsilon}) = \mathcal{O}(\varepsilon^{-1} \log^2 |U|)$ and $C_u \subseteq [l]$ where:

$$\ell = 2^d \cdot 2^d \cdot k 2^{\mathcal{O}(\log^* k)} = \frac{t^2 \log^2 z}{\varepsilon^2} \cdot k \cdot 2^{\mathcal{O}(\log^* k)} = \mathcal{O}(\varepsilon^{-2} k \log^5 |U|).$$

We claim that the obtained code is a $(\{S_i\}, (1-\varepsilon)w)$-superimposed code. Consider any S_i and $u \notin S_i$. We count elements of C_u that do not belong to any C_v, for $v \in S_i$. Let $c = f(u) \in [k \cdot 2^{\mathcal{O}(\log^* k)}]$ and so $u \in X_c$. By construction, $|X_c \cap S_i| \leq \alpha \log z$. Thus, by the union bound, the probability that $h_p(u) = h_p(x)$ for some $x \in X_c \cap S_i$ is at most ε for h_p chosen uniformly at random from $\mathcal{H}(U, d)$. Recall that C_u consists of elements $H_p(u) = \textsc{num}(h_p(u)) + 2^d \cdot \textsc{num}(p) + 4^d \cdot c$ for $h_p \in \mathcal{H}(U, d)$. The number of irreducible polynomials p such that $H_p(u) = H_p(x)$ for some $x \in X_c \cap S_i$ is at most $\varepsilon \cdot w$. Consequently, at least $w - \varepsilon \cdot w = (1-\varepsilon)w$ elements of C_u do not belong to any C_v, for $v \in S_i$.

We now show that we can construct the above superimposed codes in $\mathcal{O}(|U|w)$ time. To this end, we need to generate all irreducible polynomials of degree d and to explain how we compute remainders modulo these polynomials. Note first that as we only operate on polynomials of degree $\leq t = \mathcal{O}(\log |U|)$, they fit in a machine word and hence we can subtract two polynomials or multiply a polynomial by any power of x in constant time. We can now use this to generate the irreducible polynomials and compute the sets C_u at the same time. We maintain a bit vector I that for each polynomial p of degree $\leq d$ stores an indicator bit equal to 1 iff p, i.e. iff its remainder modulo any polynomial of degree smaller than $\deg(p)$ is not zero. We consider the polynomials of degree $0, 1, 2, \ldots, d$ in order. For every irreducible polynomial p, we compute a table $\texttt{Mod}_p[q] = (q \bmod p)$ for all polynomials q of degree $\leq t$ in overall $\mathcal{O}(|U|)$ time using dynamic programming with the following recursive formula:

$$\texttt{Mod}_p[q] = \begin{cases} q, & \text{if } \texttt{deg}(q) < \texttt{deg}(p) \\ \texttt{Mod}_p[q - p \cdot x^{\texttt{deg}(q) - \texttt{deg}(p)}], & \text{otherwise} \end{cases}$$

We use the table to compute $H_p(u)$ for all $u \in U$. Also, if for a polynomial q the remainder is zero, we zero out the corresponding bit in I. Here we use the fact that $d \leq t$ to guarantee that we will find all irreducible polynomials of degree $\leq d$ in this way.

As there are w irreducible polynomials, in total we spend $\mathcal{O}(|U|w) = \mathcal{O}(\varepsilon^{-1}|U| \log^2 |U|)$ time. At any moment, we use $\mathcal{O}(|U|)$ space to store the table and $\mathcal{O}(\varepsilon^{-1}|U| \log^2 |U|)$ space to store the codes. ◀

3 Upper Bounds for Generalised Pattern Matching

In this section, we present new algorithms for the parameters $\mathcal{D}$, $\mathcal{S}$ and $\mathcal{I}$. Our algorithms for the parameters $\mathcal{D}$ and $\mathcal{S}$ share similar ideas, so we present them together in Section 3.1. The algorithm for $\mathcal{I}$ is presented in Section 3.2.

We start by recalling the formal statement of the Pattern Matching with Don't Cares problem that will be used throughout this section.

> Pattern Matching with Don't Cares (counting, binary alphabet)
> **Input:** A text $T \in \{0,1,?\}^n$ and a pattern $P \in \{0,1,?\}^m$, where "?" is a *don't care character* that matches any character of the alphabet.
> **Output:** For each $i \in [n-m+1]$, the number of positions $j \in [m]$ such that $T[i+j-1]$ does not match $P[j]$.

Clifford and Clifford [15] showed that this problem can be solved in $\mathcal{O}(n \log m)$ time.

3.1 Parameters $\mathcal{D}$ and $\mathcal{S}$

We first show Monte Carlo algorithms for the reporting and counting variants of GPM, and then de-randomise them using the data-dependent superimposed codes of Section 2.

3.1.1 Randomised Algorithms

We start by presenting a new reporting algorithm for the parameter $\mathcal{D}$. It does not improve over the algorithm of [34], but encapsulates a novel idea that will be used by all our algorithms for the parameters $\mathcal{D}$ and $\mathcal{S}$. Essentially, we use hashing to reduce Σ_T to a smaller set of characters of size $p = \Theta(\mathcal{D})$ while preserving occurrences of the pattern in the text with constant probability, and then show that this smaller instance of GPM can be reduced to $p = \Theta(\mathcal{D})$ instances of Pattern Matching with Don't Cares.

▶ **Theorem 11.** *Let $\mathcal{D}$ be the maximum degree in the matching graph M and c be any constant fixed in advance. There is a Monte Carlo algorithm that solves the reporting variant of GPM in $\mathcal{O}(\mathcal{D}\, n \log m \log n)$ time. The error is one-sided (only false positives are allowed), and the error probability is at most $1/n^c$.*

Proof. If $\mathcal{D} > m$, we can use a naive algorithm that compares the pattern and each m-length substring of the text character-by-character and uses $\mathcal{O}(mn) = \mathcal{O}(\mathcal{D}\, n)$ time in total. Below we assume $\mathcal{D} \leq m$. We can also assume $|\Sigma_T| \leq n$.

We first choose a 2-wise independent hash function $h : \Sigma_T \to [2\mathcal{D}]$ of the form $h(x) = ((a \cdot x + b) \mod p) \mod (2\mathcal{D}) + 1$, where $p \geq |\Sigma_T|$ is a prime, and a, b are chosen independently and uniformly from $\mathbb{F}_p$. Note that we can find a prime p such that $n \leq p \leq 2n$, in $\mathcal{O}(n)$ time. Consider a matching graph M' on the set of vertices $[p] \cup \Sigma_P$. For every character $b = P[j]$ and for every character $a \in \Sigma_T$ in the adjacency list of b, we add an edge $(h(a), b)$ to M'. Overall, it takes $\mathcal{O}(\mathcal{D}\, m) = \mathcal{O}(\mathcal{D}\, n)$ time.

We claim that if M does not contain an edge (a, b), then the probability of M' to contain an edge $(h(a), b)$ is at most $1/2$. By definition, if $(h(a), b)$ belongs to M', then there exists a character $a' \in \Sigma_T$ such that (a', b) is in M and $h(a') = h(a)$. Since h is 2-wise independent, for a fixed character a' the probability of $h(a') = h(a)$ is $1/(2\mathcal{D})$. Because the degree of b is at most $\mathcal{D}$, the probability of such event is at most $1/2$ by the union bound.

Consider a text T', where $T'[i] = h(T[i])$. If $T[i, i+m-1]$ does not match P under M, then $T'[i, i+m-1]$ does not match P under M' with probability $\geq 1/2$. Indeed, suppose that for some $j \in [m]$, $T[i+j-1]$ and $P[j]$ do not match under M, or equivalently, an

edge $(T[i+j-1], P[j])$ does not belong to M. From above, with probability at least $1/2$, $h(T[i+j-1])$ and $P[j]$ do not match under M'. It follows that we can use the GPM algorithm for M', T', and P to eliminate every non-occurrence of P in T with probability at least $1/2$. We can amplify the probability in a standard way, i.e. by independently repeating the algorithm $c \log n$ times.

It remains to explain how to solve GPM for M', T', and P. We use the fact that the size of the alphabet of T' is $\mathcal{O}(\mathcal{D})$. For every $a \in [2\mathcal{D}]$ we create a new text $T'_a[1,n]$ and a new pattern $P_a[1,m]$ as follows:

$$T'_a[j] = \begin{cases} \texttt{0} & \text{if } T'[j] = a, \\ \texttt{?} & \text{otherwise.} \end{cases} \qquad P_a[j] = \begin{cases} \texttt{0} & \text{if } a \text{ matches } P[j] \text{ under } M', \\ \texttt{1} & \text{otherwise.} \end{cases}$$

We can construct T'_a and P_a in $\mathcal{O}(n+m) = \mathcal{O}(n)$ time, or in $\mathcal{O}(\mathcal{D}\,n)$ total time for all $a \in [2\mathcal{D}]$. It is not hard to see that $T'[i, i+m-1]$ matches P if and only if $T'_a[i, i+m-1]$ matches P_a for all $a \in [2\mathcal{D}]$. Therefore, to solve GPM for M', T', and P, it suffices to solve the $2\mathcal{D}$ instances of Pattern Matching with Don't Cares. By [15], this can be done in total $\mathcal{O}(\mathcal{D}\,n \log m)$ time. As we repeat the algorithm $c \log n$ times, the theorem follows. ◀

We now show a new randomised algorithm for the parameter $\mathcal{S}$. At a high level, we divide Σ_P into heavy and light characters based on their degree in M (a character of Σ_P is called heavy when it matches many characters of Σ_T, and light otherwise). The number of heavy characters is relatively small, and we can eliminate all substrings of T that do not match P because of heavy characters by running an instance of Pattern Matching with Don't Cares for each of them. For light characters, we apply Theorem 11.

▶ **Theorem 12.** *Let $\mathcal{S}$ be the number of edges in the matching graph M and c be any constant fixed in advance. There is a Monte Carlo algorithm that solves the reporting variant of GPM in $\mathcal{O}(\sqrt{\mathcal{S}}\,n \log m \sqrt{\log n})$ time. The error is one-sided (only false positive are allowed), and the error probability is at most $1/n^c$.*

Combining the techniques of Theorems 11, 12 and the approach of Kopelowitz and Porat [31], we obtain the following corollary.

▶ **Corollary 13.** *Let c be any constant fixed in advance, $\mathcal{D}$ be the maximum degree and $\mathcal{S}$ be the number of edges in the matching graph M. There is a $(1-\varepsilon)$-approximation Monte Carlo algorithm that solves the counting variant of GPM in $\mathcal{O}(\min\{\varepsilon^{-1}\mathcal{D}\log n, \sqrt{\varepsilon^{-1}\mathcal{S}\log n}\} \cdot n \log m)$ time. The error probability is at most $1/n^c$.*

3.1.2 Deterministic Algorithms

We are now ready to give $(1-\varepsilon)$-approximation deterministic algorithms for the counting variant of GPM for the parameters $\mathcal{D}$ and $\mathcal{S}$. By taking $\varepsilon = 1/2$, the algorithms for the reporting variant follow immediately. We first remind the definition of superimposed codes, which we will use throughout this section.

▶ **Definition 1.** *Let $S_1, \ldots, S_z$ be subsets of a universe U. A family of sets $\mathcal{C} = \{C_1, \ldots, C_{|U|}\}$, where $C_u \subseteq [\ell]$ and $|C_u| = w$ for $u \in U$ is called an $(\{S_i\}, \tau)$-superimposed code if for every S_i and $u \notin S_i$ we have $|C_u - \bigcup_{v \in S_i} C_v| \geq \tau$. We call ℓ and w respectively the length and the weight of the code $\mathcal{C}$.*

▶ **Theorem 14.** *Let $\mathcal{D}$ be the maximum degree in the matching graph M. There is an $(1-\varepsilon)$-approximation deterministic algorithm that solves the counting variant of GPM in $\mathcal{O}(\varepsilon^{-2}\mathcal{D}\,n \log^6 n)$ time.*

Proof. First, note that we can assume $\mathcal{D} \leq m$ and $\varepsilon \geq 1/m$. If this is not the case, we can run a naive algorithm that compares each m-length substring of the text T and the pattern character-by-character in $\mathcal{O}(mn) = \mathcal{O}(\mathcal{D}\, n)$ time.

For each distinct character b of the pattern P, consider a set S_b containing all characters in Σ_T that match b. By definition, $|S_b| \leq \mathcal{D}$. We define the universe $U = (\bigcup_{b\in\Sigma_P} S_b) \cup \{\$\}$, where $\$ \notin \Sigma_T$ is a special character that we will need later, $|U| = \mathcal{O}(n)$. We apply Theorem 10 that constructs $(\{S_b\}, (1-\varepsilon)w)$-superimposed code for the universe U and sets S_b in $\mathcal{O}(\varepsilon^{-1} n \log^2 n)$ time, where the weight $w = \mathcal{O}(\varepsilon^{-1}\log^2 n)$ and the length $\ell = \mathcal{O}(\varepsilon^{-2}\mathcal{D}\log^5 n)$.

We define the code of a character $a \in U$ to be a binary vector of length ℓ such that its j-th bit equals 1 if C_a contains j, and 0 otherwise. For a character $a' \in \Sigma_T \setminus U$, we define its code to be equal to the code of \$. We define the code of a character $b \in \Sigma_P$ to be a binary vector of length ℓ such that its j-th bit equals 1 if $\bigcup_{a\in S_b} C_a$ contains j, and 0 otherwise. Next, we create a text $T'[1, n\ell]$ and a pattern $P'[1, m\ell]$ by replacing the characters in respectively T and P by their codes. To finish this step, we replace each 1 in P' with the don't care character and run the algorithm of Clifford and Clifford [15] for T' and P' that takes $\mathcal{O}(n\ell\log(m\ell)) = \mathcal{O}(\varepsilon^{-2}\mathcal{D}\, n\log^6 n)$ time (here we use $\varepsilon \geq 1/m$).

Let h' be the number of mismatching characters between P' and $T'[(i-1)\cdot\ell+1, (i+m-1)\cdot\ell]$, and h be the number of mismatches between P and $T[i, i+m-1]$. We claim that $(1-\varepsilon)wh \leq h' \leq wh$. Indeed, if $P[j]$ matches $T[i+j-1]$, then $C_{T[i+j-1]}$ is a subset of $\bigcup_{a\in S_{P[j]}} C_a$. Therefore, if the code of $T[i+j-1]$ contains 1 in position k, the code of $P[j]$ will have 1 in position k as well. By replacing all 1s in P' with the don't care characters, we ensure that the corresponding fragments of P' and T' match. On the other hand, if $P[j]$ does not match $T[i+j-1]$, then from the definition of the code it follows that the distance between the corresponding chunks of P' and T' will be at least $(1-\varepsilon)w$ and at most w. ◀

To show a deterministic algorithm for the parameter $\mathcal{S}$, we again consider the partition of the alphabet Σ_P into heavy and light characters. To count the mismatches caused by some heavy character, we create an instance of Pattern Matching with Don't Cares. As the number of heavy characters is small, the total number of the created instances is small as well. For light characters, we use the superimposed codes similarly as in Theorem 14.

▶ **Theorem 15.** *Let $\mathcal{S}$ be the number of edges in the matching graph M. There is an $(1-\varepsilon)$-approximation deterministic algorithm that solves the counting variant of GPM in $\mathcal{O}(\varepsilon^{-1}\sqrt{\mathcal{S}}\, n\log^{7/2} n)$ time.*

3.2 Parameter $\mathcal{I}$

In this section, we show a deterministic GPM algorithm for the parameter $\mathcal{I}$. The algorithm solves the counting variant of the problem exactly, and we can immediately derive an algorithm for the reporting version with the same complexities as a corollary. We will need the following technical lemma.

▶ **Lemma 16.** *Let b be a parameter, $S = \{x_1, x_2, \ldots, x_\ell\}$ be a sequence of integers, and $s = \sum_{i\in[\ell]} x_i$. Then S can be partitioned into $\mathcal{O}(s/b+1)$ ranges $S_1, S_2, \ldots$ such that, for every i, either S_i is a singleton or the sum of all elements in S_i is at most b.*

We are now ready to show the main result of the section.

▶ **Theorem 17.** *For each character $a \in \Sigma_P$ consider a minimal set $I(a)$ of disjoint sorted intervals that contain the characters that match a, and define $\mathcal{I} = \sum_{j\in[m]} |I(P[j])|$. There is a deterministic algorithm that solves the counting version of GPM in $\mathcal{O}(n\sqrt{\mathcal{I}\log m} + n\log n)$ time.*

Proof. If $\mathcal{I} > m^2$, we can use the naive algorithm that compares each m-length substring with the pattern character-by-character and takes $\mathcal{O}(mn)$ time in total.

We first make a pass over T and retrieve the set of distinct characters $a_1, a_2, \ldots, a_l$ of Σ_T that occur in it, as well as their frequencies. This can be done in $\mathcal{O}(n \log n)$ time using a binary search tree. We partition $a_1, a_2, \ldots, a_l$ into ranges as follows. Let $\mathsf{count}(c)$, for $c \in \Sigma_T$, be the frequency (i.e. the number of occurrences) of c in T. We apply Lemma 16 for $b > 1$ that will be specified later and the sequence $\mathsf{count}(a_1), \mathsf{count}(a_2), \ldots, \mathsf{count}(a_l)$ which sums up to n.

Let Σ'_T be a new alphabet obtained by creating a character for every range in the partition, where $|\Sigma'_T| = \mathcal{O}(n/b + 1)$. For $c \in \Sigma'_T$ we denote by $\mathsf{range}(c)$ the range of Σ_T corresponding to c, and for $a \in \Sigma_T$ we denote by $\mathsf{range}^{-1}(a)$ the character of Σ'_T corresponding to the range containing a. We create a new text $T'[1, n]$ and pattern $P'[1, m]$ as follows. For every $i \in [n]$, we set $T'[i] = \mathsf{range}^{-1}(T[i])$. For every $j \in [m]$, we set $P'[j] = \{c \in \Sigma'_T \mid \mathsf{range}(c) \text{ contains a character that matches } P[j]\}$. As the number of the ranges is $\mathcal{O}(n/b + 1)$, the size of the set $P'[j]$ is $\mathcal{O}(n/b + 1)$. We represent it as a binary vector of length $\mathcal{O}(n/b + 1)$. Furthermore, we can construct T' in $\mathcal{O}(n)$ time, and P' in $\mathcal{O}(\mathcal{I} + m(n/b + 1))$ time.

After this initial step the algorithm consists of two phases. First, we solve the Subset Pattern Matching for T' and P' that consists of counting, for every $i \in [n - m + 1]$, all positions $j \in [m]$ such that $T'[i + j - 1] \notin P'[j]$. To this end, we create an instance of Pattern Matching with Don't Cares for every $c \in \Sigma'_T$, namely, we create a text $T'_c[1, n]$ and a pattern $P'_c[1, m]$ as follows:

$$T'_c[i] = \begin{cases} \mathtt{0} & \text{if } T'[i] = c, \\ \mathtt{?} & \text{otherwise.} \end{cases} \qquad P'_c[j] = \begin{cases} \mathtt{0} & \text{if } c \in P'[j], \\ \mathtt{1} & \text{otherwise.} \end{cases}$$

We can solve all these instances in $\mathcal{O}(|\Sigma'_T| n \log m) = \mathcal{O}((n/b+1)n \log m)$ time [15]. Summing up the results, we obtain the result for the subset matching problem.

In the second phase, we slightly adjust the results obtained for Subset Pattern Matching to obtain the results for GPM. Consider a substring $T[i, i + m - 1]$ that does not match P because of a mismatch in position j of the pattern, i.e. $T[i + j - 1]$ does not match $P[j]$. We have two possible cases. The first case is when $T'[i + j - 1] \notin P'[j]$. In this case, the mismatch is detected by the Subset Pattern Matching algorithm. The second case is when $T'[i + j - 1] \in P'[j]$. Observe that in this case, $\mathsf{range}(T'[i + j - 1])$ cannot be a singleton and must contain an endpoint of some interval of characters that match $P[j]$.

To detect such mismatches, we run the following algorithm. For each $j \in [m]$, we consider the intervals $I(P[j])$ of the characters that match $P[j]$. For every endpoint $c \in \Sigma_T$ of the intervals in $I(P[j])$, we iterate over all $a \in \mathsf{range}^{-1}(c)$ such that a does not match $P[j]$ and all occurrences of a in the text. Summing over all j and a, there are in total $\mathcal{O}(\mathcal{I} \cdot b)$ of the occurrences due to the properties of the partition and the fact that $\mathsf{range}(T'[i + j - 1])$ is not a singleton. We can find the occurrences in $\mathcal{O}(\mathcal{I} \cdot b + n + mn/b)$ time as follows. First we find the ranges containing the endpoints in $\mathcal{O}(\mathcal{I} + m(n/b+1))$ time similarly to above, and we can generate the lists of occurrences of every character $a \in \Sigma_T$ in T by one pass over T in $\mathcal{O}(n \log n)$ time. For each such occurrence $T[k] = a$ that does not match $P[j]$, we increment the number of mismatches for the substring $T[k - j + 1, k + m - j]$. This correctly detects every mismatch that has not been accounted for in the first phase, and hence allows counting all mismatches in $\mathcal{O}(\mathcal{I} + m(n/b + 1) + (n/b + 1)n \log m + n \log n + \mathcal{I} \cdot b) = \mathcal{O}(n^2 \log(m)/b + n \log n + \mathcal{I} \cdot b)$ total time. Substituting $b = n\sqrt{\log m/\mathcal{I}}$ gives us the claim of the theorem. ◀

▶ **Corollary 18.** *There is a deterministic algorithm that solves the counting variant of the threshold pattern matching problem in* $\mathcal{O}(n(\sqrt{m \log m} + \log n))$ *time.*

4 Lower Bounds for GPM

In this section we give lower bounds for GPM algorithms. All the lower bounds are presented for the reporting variant of GPM, so they immediately apply also to the counting variant. Recall that we assume to have access to three oracles that can answer the following questions about the matching graph M in $\mathcal{O}(1)$ time:

1. Is there an edge between $a \in \Sigma_T$ and $b \in \Sigma_P$?
2. What is the degree of a character $a \in \Sigma_T$ or $b \in \Sigma_P$?
3. What is the k-th neighbor of $a \in \Sigma_T$?

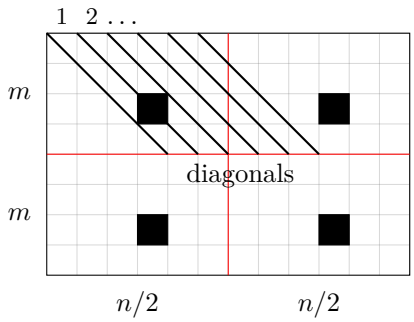


Figure 1 The adjacency matrix of the matching graph M. We show diagonals (solid lines) and a quadruple of related cells (black). Note that among any quadruple of related cells, only one can belong to a diagonal.

We first use an adversary-based argument to show an $\Omega(\mathcal{S})$ time lower bound.

▶ **Lemma 19.** *Any deterministic algorithm for GPM requires* $\Omega(\mathcal{S})$ *time.*

Proof. We will show that any deterministic algorithm checking if there exists at least one occurrence needs to inspect $\Omega(\mathcal{S})$ entries of M in the worst case by an adversary-based argument. In particular, this implies a lower bound of $\Omega(nm)$ when $\mathcal{S} = \Theta(nm)$. The main difficulty in the argument is to design the input so that the second oracle is essentially useless.

It will be convenient for us to think in terms of the adjacency matrix of the matching graph M that we denote by $\mathcal{M}$. Let us assume that $n \geq 2m$ is even, $\Sigma_T = [n]$, and $\Sigma_P = [2m]$. We split both alphabets into halves. For every $a \in [n/2]$ and $b \in [m]$ we will choose one of the following two possibilities:

1. $\mathcal{M}[a, b] = \mathcal{M}[n/2 + a, m + b] = 1$ and $\mathcal{M}[n/2 + a, b] = \mathcal{M}[a, m + b] = 0$,
2. $\mathcal{M}[a, b] = \mathcal{M}[n/2 + a, m + b] = 0$ and $\mathcal{M}[n/2 + a, b] = \mathcal{M}[a, m + b] = 1$.

We call $\mathcal{M}[a, b]$, $\mathcal{M}[n/2 + a, b]$, $\mathcal{M}[a, m + b]$ and $\mathcal{M}[n/2 + a, m + b]$ *related*. Observe that, irrespectively of all such choices, the second oracle returns the same number for every $b \in \Sigma_P$ and every $a \in \Sigma_T$, and so the algorithm only needs to query the first oracle.

We choose the text $T = 1\ 2 \ldots n/2\ 1\ 2 \ldots n/2$ and the pattern $P = 1\ 2 \ldots m$. Clearly, P occurs in T when, for some $a \in [n/2]$, we have $M[1 + (a + b - 2) \bmod n/2, b] = 1$ for every $b \in [m]$. We call the set of corresponding entries of M a *diagonal* (see Fig. 1).

Note that among any quadruple of related entries exactly one can belong to the diagonals. Furthermore, suppose that an algorithm retrieves the values in a quadruple of related entries $\mathcal{M}[a,b]$, $\mathcal{M}[n/2+a,m+b]$, $\mathcal{M}[n/2+a,b]$, $\mathcal{M}[a,m+b]$. This can be done by one of the following queries: ask for the value of any of these four entries, or retrieve the particular neighbor of one of the nodes a, $n/2+a$, b, or $m+b$. In both cases, we retrieve only the related entries and spend $\Omega(1)$ time for any of the retrieved quadruples.

The adversary proceeds as follows. If the algorithm retrieves a quadruple containing $\mathcal{M}[a,b]$, for $a \in [n/2]$ and $b \in [m]$, such that the value of $\mathcal{M}[a,b]$ is not yet determined, the adversary checks if setting $\mathcal{M}[a,b]=1$ would result in creating a diagonal containing only 1s. If so, the adversary sets $\mathcal{M}[a,b]=0$, and otherwise the adversary sets $\mathcal{M}[a,b]=1$. In other words, the adversary sets $M[a,b]=0$ when it is the last undecided entry on its diagonal.

The algorithm can report an occurrence only after having verified that the corresponding diagonal contains only 1s, and the adversary makes sure that this is never the case. On the other hand, if the algorithm terminates without having reported an occurrence while there exists a diagonal that has not been fully verified then the adversary could set its remaining entries to 1s and obtain an instance that does contain an occurrence. Consequently, the algorithm needs to retrieve all the entries in all the diagonals, and as we showed, it requires $\Omega(mn)=\Omega(\mathcal{S})$ time.

Note that above $\mathcal{S}=nm/2$. The proof can be extended to $\mathcal{S}<nm/2$ as follows. If $\mathcal{S}\geq m$ we set $n'=\lfloor \mathcal{S}/m \rfloor$ and choose the text to be the prefix of length n of $(1\ 2\ldots n')^\infty$ (the string $1\ 2\ldots n'$ repeated infinitely many times). Then the above argument shows that any algorithm needs to inspect $n'm \geq \mathcal{S}/2$ entries of $\mathcal{M}$. If $\mathcal{S}<m$ we choose the pattern to be the prefix of length m of $(1\ 2\ldots \mathcal{S})^\infty$ (the string $1\ 2\ldots \mathcal{S}$ repeated infinitely many times), the text to be 1^n (1 repeated n times) and proceed as above to argue that one must inspect $\Omega(\mathcal{S})$ entries of $\mathcal{M}$. ◀

A similar argument can be used to show that this bound holds for Monte Carlo algorithms with constant error probability as well.

▶ **Lemma 20.** *Any Monte Carlo algorithm for GPM with constant error probability $\varepsilon < 1/2$ requires $\Omega(\mathcal{S})$ time.*

We now show lower bounds for GPM conditional on hardness of Boolean matrix multiplication.

▶ **Conjecture** ([2]). *For any $\alpha,\beta,\gamma,\varepsilon>0$, there is no combinatorial[3] algorithm for multiplying two Boolean matrices of size $N^\alpha \times N^\beta$ and $N^\beta \times N^\gamma$ in time $\mathcal{O}(N^{\alpha+\beta+\gamma-\varepsilon})$.*

A simple adaptation of the folklore lower bound for computing the Hamming distance (cf. [24]) yields the following lower bounds.

▶ **Lemma 21.** *For any $\alpha \geq 1$, and $1 \geq \beta,\varepsilon > 0$, there is no combinatorial algorithm that solves GPM in time $\mathcal{O}(\mathcal{S}^{0.5-\varepsilon}n)$, for $n=\Theta(m^{(1+\alpha)/2})$ and $\mathcal{S}=\Theta(m^\beta)$.*

Proof. We show a reduction from Boolean matrix multiplication. Consider a matrix A of size $x \times y$ and a matrix B of size $y \times z$, where $x=N^\alpha$, $y=N^\beta$, $z=N$. We transform the matrix A by replacing every 1 by the number of the column it belongs to and every 0 by the don't care character ?. Similarly, we replace each 1 in B by the number of the row it belongs to and every 0 by the don't care character ?.

[3] It is not clear what combinatorial means precisely, but fast matrix multiplication is definitely non-combinatorial. Arguably neither is FFT used in our algorithms, thus making them non-combinatorial.

▶ **Example 22.** Consider $A = ((0,0,1),(1,0,1),(0,1,0))$ and $B = ((1,0,1),(0,1,0),(1,1,0))$. After the transform, they become $((?,?,3),(1,?,3),(?,2,?))$ and $((1,?,1),(?,2,?),(3,3,?))$, respectively.

We define the text $T = ?^{z^2} A_1 ?^{z-y+1} A_2 ?^{z-y+1} \dots ?^{z-y+1} A_x ?^{z^2}$, where A_i is the i-th row of A, and the pattern $P = B_1 ?^{z-y} B_2 ?^{z-y} \dots ?^{z-y} B_z$, where B_j is the j-th column of the matrix B. The length of T is $n = 2z^2 + (x-1)(z-y+1) + xy = \mathcal{O}(N^{1+\alpha})$, and the length of P is $m = yz + (z-y)(z-1) = \mathcal{O}(N^2)$. Next, we define the matching relationship as follows. Every character different than the don't care is defined to match all characters of the alphabet but itself, and the don't care character matches all characters of the alphabet. Consequently, the alphabet has size $y+1$ and the matching relationship matrix contains $\mathcal{S} = \Theta(y^2) = \Theta(N^{2\beta})$ set bits.

Let $C = A \times B$. By definition, $C[i,j] = 1$ iff $\bigvee_{k=1}^{y} (A_i[k] \wedge B_j[k]) = 1$. We claim that this is the case iff, aligning A_i in the text and B_j in the pattern does not yield an occurrence of the pattern. Suppose first that $\bigvee_{k=1}^{y} (A_i[k] \wedge B_j[k]) = 1$. Then there is k_0 such that $A_i[k_0] = B_j[k_0] = 1$. In the text and in the pattern they are both encoded by the same $k_0 \neq ?$ and aligned, and k_0 does not match itself. Therefore, we do not have an occurrence. Assume otherwise. We need to show that for every character $a \neq ?$, a is not aligned with itself. For B_j it follows from the fact that $\bigvee_{k=1}^{y} (A_i[k] \wedge B_j[k]) \neq 1$. For other columns of B it follows from the shift caused by the don't care characters.

It follows that a combinatorial algorithm that correctly outputs all occurrences of P in T in $\mathcal{O}(\mathcal{S}^{0.5-\varepsilon} n)$ time implies a combinatorial algorithm for Boolean matrix multiplication of matrices of size $N^\alpha \times N^\beta$ and $N^\beta \times N$ in time $\mathcal{O}(\mathcal{S}^{0.5-\varepsilon} n) = \mathcal{O}(N^{1+\alpha+2\beta(0.5-\varepsilon)}) = \mathcal{O}(N^{\alpha+1+\beta-2\varepsilon\beta})$, which contradicts the combinatorial matrix multiplication conjecture. The lower bound follows. ◀

▶ **Corollary 23.** *For any $\alpha \geq 1$, and $1 \geq \beta, \varepsilon > 0$, there is no combinatorial algorithm that solves GPM in time $\mathcal{O}(\mathcal{D}^{1-\varepsilon} n)$, for $n = \Theta(m^{(1+\alpha)/2})$ and $\mathcal{D} = \Theta(m^\beta)$. For any $\alpha \geq 1$, and $1 \geq \varepsilon > 0$, there is no combinatorial algorithm that solves GPM in time $\mathcal{O}(\mathcal{I}^{0.5-\varepsilon} n)$, for $n = \Theta(m^{(1+\alpha)/2})$ and $\mathcal{I} = \Theta(m)$.*

Proof. To show the first part of the claim, note that in the constructed instance of generalized pattern matching $\mathcal{D} = \Theta(m^{\beta/2})$. For the second part, we take $\beta = 1$. Then $\mathcal{I} = \mathcal{O}(m)$, and therefore a combinatorial algorithm that correctly outputs all occurrences of P in T in $\mathcal{O}(\mathcal{I}^{0.5-\varepsilon} n)$ time implies a combinatorial algorithm for Boolean matrix multiplication of matrices of size $N^\alpha \times N$ and $N \times N$ in time $\mathcal{O}(\mathcal{I}^{0.5-\varepsilon} n) = \mathcal{O}(N^{1+\alpha+2(0.5-\varepsilon)}) = \mathcal{O}(N^{\alpha+2-2\varepsilon})$, which contradicts the combinatorial matrix multiplication conjecture. ◀

References

1 Amir Abboud, Loukas Georgiadis, Giuseppe F. Italiano, Robert Krauthgamer, Nikos Parotsidis, Ohad Trabelsi, Przemyslaw Uznanski, and Daniel Wolleb-Graf. Faster algorithms for all-pairs bounded min-cuts. In *Proceedings of the International Colloquium on Automata, Languages, and Programming*, ICALP, pages 7:1–7:15, 2019. doi:10.4230/LIPIcs.ICALP.2019.7.

2 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of the 2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, FOCS, pages 434–443. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.53.

3 Karl R. Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, 1987. doi:10.1137/0216067.

4 Mikhail J. Atallah and Timothy W. Duket. Pattern matching in the Hamming distance with thresholds. *Information Processing Letters*, 111(14):674–677, 2011. doi:10.1016/j.ipl.2011.04.004.

5 Nikhil Bansal. Constructive algorithms for discrepancy minimization. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, FOCS, pages 3–10, 2010. doi:10.1109/FOCS.2010.7.

6 Nikhil Bansal, Moses Charikar, Ravishankar Krishnaswamy, and Shi Li. Better algorithms and hardness for broadcast scheduling via a discrepancy approach. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 55–71, 2014. doi:10.1137/1.9781611973402.5.

7 Nikhil Bansal, Daniel Dadush, and Shashwat Garg. An algorithm for Komlós conjecture matching Banaszczyk's bound. *SIAM J. Comput.*, 48(2):534–553, 2019. doi:10.1137/17M1126795.

8 Nikhil Bansal, Daniel Dadush, Shashwat Garg, and Shachar Lovett. The Gram-Schmidt walk: A cure for the Banaszczyk blues. In *Proceedings of the Annual ACM SIGACT Symposium on Theory of Computing*, STOC, pages 587–597, 2018. doi:10.1145/3188745.3188850.

9 Nikhil Bansal and Shashwat Garg. Algorithmic discrepancy beyond partial coloring. In *Proceedings of the Annual ACM SIGACT Symposium on Theory of Computing*, STOC, pages 914–926, 2017. doi:10.1145/3055399.3055490.

10 Nikhil Bansal and Joel Spencer. Deterministic discrepancy minimization. *Algorithmica*, 67(4):451–471, 2013. doi:10.1007/s00453-012-9728-1.

11 Emilios Cambouropoulos, Maxime Crochemore, Costas S. Iliopoulos, Laurent Mouchard, and Yoan J. Pinzon. Algorithms for computing approximate repetitions in musical sequences. *International Journal of Computer Mathematics*, 79(11):1135–1146, 2002. doi:10.1080/00207160213939.

12 Domenico Cantone, Salvatore Cristofaro, and Simone Faro. An efficient algorithm for δ-approximate matching with α-bounded gaps in musical sequences. In *Proceedings of the International Conference on Experimental and Efficient Algorithms*, WEA, pages 428–439, 2005. doi:10.1007/11427186_37.

13 Bernard Chazelle. *The discrepancy method - randomness and complexity*. Cambridge University Press, 2001. doi:10.1017/CBO9780511626371.

14 Sunil Chebolu and Jan Minac. Counting irreducible polynomials over finite fields using the inclusion-exclusion principle. *Mathematics Magazine*, 84(5):369–371, 2011. doi:10.4169/math.mag.84.5.369.

15 Peter Clifford and Raphaël Clifford. Simple deterministic wildcard matching. *Information Processing Letters*, 101(2):53–54, 2007. doi:10.1016/j.ipl.2006.08.002.

16 Peter Clifford, Raphaël Clifford, and Costas Iliopoulos. Faster algorithms for δ,γ-matching and related problems. In *Proceedings on the Annual Symposium on Combinatorial Pattern Matching*, CPM, pages 68–78, 2005. doi:10.1007/11496656_7.

17 Raphaël Clifford and Ely Porat. A filtering algorithm for k-mismatch with don't cares. *Information Processing Letters*, 110(22):1021–1025, 2010. doi:10.1016/j.ipl.2010.08.012.

18 Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, STOC, pages 592–601, 2002. doi:10.1145/509907.509992.

19 Richard Cole, Costas Iliopoulos, Thierry Lecroq, Wojciech Plandowski, and Wojciech Rytter. On special families of morphisms related to δ-matching and don't care symbols. *Information Processing Letters*, 85(5):227–233, 2003. doi:10.1016/S0020-0190(02)00430-1.

20 Maxime Crochemore, Costas S. Iliopoulos, Thierry Lecroq, Yoan J. Pinzon, Wojciech Plandowski, and Wojciech Rytter. Occurrence and substring heuristics for δ-matching. *Fundamenta Informaticae*, 56(1,2):1–21, October 2002.

21 Michael John Fischer and Michael Stewart Paterson. String-matching and other products. Technical report, Massachusetts Institute of Technology, 1974.

22 Kimmo Fredriksson and Szymon Grabowski. Efficient algorithms for (δ, γ, α) and $(\delta, k_\delta, \alpha)$-matching. *International Journal of Foundations of Computer Science*, 19(01):163–183, 2008. doi:10.1142/S0129054108005607.

23 Carl Friedrich Gauss. *Untersuchungen über höhere Arithmetik. (Disquisitiones arithmeticae. Theorematis arithmetici demonstratio nova. Summatio quarundam serierum singularium ó.)*. Deutsch hrsg. von H. Mas, Berlin, 1889.

24 Paweł Gawrychowski and Przemysław Uznański. Towards unified approximate pattern matching for Hamming and L_1 distance. In *Procedings of the International Colloquium on Automata, Languages and Programming*, ICALP, pages 62:1–62:13, 2018. doi:10.4230/LIPIcs.ICALP.2018.62.

25 Loukas Georgiadis, Daniel Graf, Giuseppe F. Italiano, Nikos Parotsidis, and Przemyslaw Uznanski. All-pairs 2-reachability in $O(n^w \log n)$ time. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming*, ICALP, pages 74:1–74:14, 2017. doi:10.4230/LIPIcs.ICALP.2017.74.

26 Jan Holub, William F. Smyth, and Shu Wang. Fast pattern-matching on indeterminate strings. *J. of Discrete Algorithms*, 6(1):37–50, March 2008. doi:10.1016/j.jda.2006.10.003.

27 Piotr Indyk. Deterministic superimposed coding with applications to pattern matching. In *Proceedings of the Annual Symposium on Foundations of Computer Science*, FOCS, pages 127–136, 1997. doi:10.1109/SFCS.1997.646101.

28 Piotr Indyk. Faster algorithms for string matching problems: Matching the convolution bound. In *Proceedings of the Annual Symposium on Foundations of Computer Science*, FOCS, pages 166–173, 1998. doi:10.1109/SFCS.1998.743440.

29 Adam Kalai. Efficient pattern-matching with don't cares. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 655–656, 2002.

30 William Kautz and Richard Singleton. Nonrandom binary superimposed codes. *IEEE Trans. Inf. Theor.*, 10(4):363–377, September 2006. doi:10.1109/TIT.1964.1053689.

31 Tsvi Kopelowitz and Ely Porat. A simple algorithm for approximating the text-to-pattern Hamming distance. In *Proceedings of the SIAM Symposium on Simplicity in Algorithms*, volume 61 of *OASICS*, pages 10:1–10:5, 2018. doi:10.4230/OASIcs.SOSA.2018.10.

32 Kasper Green Larsen. Constructive discrepancy minimization with hereditary L2 guarantees. In *Proceedings of the International Symposium on Theoretical Aspects of Computer Science*, STACS, pages 48:1–48:13, 2019. doi:10.4230/LIPIcs.STACS.2019.48.

33 Shachar Lovett and Raghu Meka. Constructive discrepancy minimization by walking on the edges. *SIAM Journal on Computing*, 44(5):1573–1582, 2015. doi:10.1137/130929400.

34 Shan Muthukrishnan. New results and open problems related to non-standard stringology. In *Proceedings of the Annual Symposium on Combinatorial Pattern Matching*, CPM, pages 298–317, 1995. doi:10.1007/3-540-60044-2_50.

35 Shan Muthukrishnan and Krishna Palem. Non-standard stringology: Algorithms and complexity. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, STOC, pages 770–779. ACM, 1994. doi:10.1145/195058.195457.

36 Shan Muthukrishnan and Hariharan Ramesh. String matching under a general matching relation. *Information and Computation*, 122(1):140–148, 1995. doi:10.1007/3-540-56287-7_118.

37 Gonzalo Navarro. NR-grep: A fast and flexible pattern-matching tool. *Softw. Pract. Exper.*, 31(13):1265–1312, October 2001. doi:10.1002/spe.411.

38 Peng Zhang and Mikhail J. Atallah. On approximate pattern matching with thresholds. *Information Processing Letters*, 123:21–26, 2017. doi:10.1016/j.ipl.2017.03.001.

Parameterized Pre-Coloring Extension and List Coloring Problems

Gregory Gutin
Royal Holloway, University of London, UK
gutin@cs.rhul.ac.uk

Diptapriyo Majumdar
Royal Holloway, University of London, UK
diptapriyo.majumdar@rhul.ac.uk

Sebastian Ordyniak
University of Sheffield, UK
sordyniak@gmail.com

Magnus Wahlström
Royal Holloway, University of London, UK
Magnus.Wahlstrom@rhul.ac.uk

Abstract

Golovach, Paulusma and Song (Inf. Comput. 2014) asked to determine the parameterized complexity of the following problems parameterized by k: (1) Given a graph G, a clique modulator D (a *clique modulator* is a set of vertices, whose removal results in a clique) of size k for G, and a list $L(v)$ of colors for every $v \in V(G)$, decide whether G has a proper list coloring; (2) Given a graph G, a clique modulator D of size k for G, and a pre-coloring $\lambda_P : X \to Q$ for $X \subseteq V(G)$, decide whether λ_P can be extended to a proper coloring of G using only colors from Q. For Problem 1 we design an $\mathcal{O}^*(2^k)$-time randomized algorithm and for Problem 2 we obtain a kernel with at most $3k$ vertices. Banik et al. (IWOCA 2019) proved the following problem is fixed-parameter tractable and asked whether it admits a polynomial kernel: Given a graph G, an integer k, and a list $L(v)$ of exactly $n - k$ colors for every $v \in V(G)$, decide whether there is a proper list coloring for G. We obtain a kernel with $\mathcal{O}(k^2)$ vertices and colors and a compression to a variation of the problem with $\mathcal{O}(k)$ vertices and $\mathcal{O}(k^2)$ colors.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized Algorithms, W-hardness, Kernelization, Graph Coloring, List Coloring

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.19

Related Version A full version of this paper is available at `https://arxiv.org/abs/1907.12061`.

1 Introduction

Graph coloring is a central topic in Computer Science and Graph Theory due to its importance in theory and applications. Every text book in Graph Theory has at least a chapter devoted to the topic and the monograph of Jensen and Toft [21] is completely devoted to graph coloring problems focusing especially on more than 200 unsolved ones. There are many survey papers on the topic including recent ones such as [10, 18, 25, 27].

For a graph G, a *proper coloring* is a function $\lambda : V(G) \to \mathbb{N}_{\geq 1}$ such that for no pair u, v of adjacent vertices of G, $\lambda(u) = \lambda(v)$. In the widely studied Coloring problem, given a graph G and a positive integer p, we are to decide whether there is a proper coloring $\lambda : V(G) \to [p]$, where henceforth $[p] = \{1, \ldots, p\}$. In this paper, we consider two extensions of Coloring: the Pre-Coloring Extension problem and the List Coloring problem.

In the Pre-Coloring Extension problem, given a graph G, a set Q of colors, and a *pre-coloring* $\lambda_P : X \to Q$, where $X \subseteq V(G)$, we are to decide whether there is a proper coloring $\lambda : V(G) \to Q$ such that $\lambda(x) = \lambda_P(x)$ for every $x \in X$. In the List Coloring problem, given a graph G and a list $L(u)$ of possible colors for every vertex u of G, we are to decide whether G has a proper coloring λ such that $\lambda(u) \in L(u)$ for every vertex u of G. Such a coloring λ is called a *proper list coloring*. Clearly, Pre-Coloring Extension is a special case of List Coloring, where all lists of vertices $x \in X$ are singletons.

The p-Coloring problem is a special case of Coloring when p is fixed (i.e., not part of input). When $Q \subseteq [p]$ ($L(u) \subseteq [p]$, respectively), Pre-Coloring Extension (List Coloring, respectively) are called p-Pre-Coloring Extension (List p-Coloring, respectively). In classical complexity, it is well-known that p-Coloring, p-Pre-Coloring Extension and List p-Coloring are polynomial-time solvable for $p \leq 2$, and the three problems become NP-complete for every $p \geq 3$ [23, 25]. In this paper, we solve several open problems about pre-coloring extension and list coloring problems, which lie outside classical complexity, so-called parameterized problems. We provide basic notions on parameterized complexity in the next section. For more information on parameterized complexity, see recent books [11, 15, 17].

The first two problems we study are the following ones stated by Golovach et al. [19] (see also [24]) who asked to determine their parameterized complexity. These questions were motivated by a result of Cai [8] who showed that Coloring Clique Modulator (the special case of Pre-Coloring Extension Clique Modulator when $X = \emptyset$) is fixed-parameter tractable (FPT). Note that a *clique modulator* of a graph G is a set D of vertices such that $G - D$ is a clique. When using the size of a clique modulator as a parameter we will for convenience assume that the modulator is given as part of the input. Note that this assumption is not necessary (however it avoids having to repeat how to compute a clique modulator) as we will show in Section 2.1 that computing a clique modulator of size k is FPT and can be approximated to within a factor of two.

List Coloring Clique Modulator parameterized by k

Input: A graph G, a clique modulator D of size k for G, and a list $L(v)$ of colors for every $v \in V(G)$.

Problem: Is there a proper list coloring for G?

Pre-Coloring Extension Clique Modulator parameterized by k

Input: A graph G, a clique modulator D of size k for G, and a pre-coloring $\lambda_P : X \to Q$ for $X \subseteq V(G)$ where Q is a set of colors.

Problem: Can λ_P be extended to a proper coloring of G using only colors from Q?

In Section 3 we show that List Coloring Clique Modulator is FPT. We first show a randomized $\mathcal{O}^*(2^{k \log k})$-time algorithm, then we improve the running time to $\mathcal{O}^*(2^k)$ using more refined approaches. Note that all our randomized algorithms are one-sided error algorithms having a constant probability of being wrong, when the algorithm outputs no.

We note that the time $\mathcal{O}^*(2^k)$ matches the best known running time of $\mathcal{O}^*(2^n)$ for Chromatic Number (where $n = |V(G)|$) [5], while applying to a more powerful parameter. It is a long-open problem whether Chromatic Number can be solved in time $\mathcal{O}(2^{cn})$ for some $c < 1$ and Cygan et al. [12] ask whether it is possible to show that such algorithms are impossible assuming the Strong Exponential Time Hypothesis (SETH).

We conclude Section 3 by showing that List Coloring Clique Modulator does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly. The reduction used to prove this result allows us to observe that if List Coloring Clique Modulator could be solved in time $\mathcal{O}(2^{ck}n^{\mathcal{O}(1)})$ for some $c < 1$, then the well-known Set Cover problem could be solved in time $\mathcal{O}(2^{c|U|}|\mathcal{F}|^{\mathcal{O}(1)})$, where U and $\mathcal{F}$ are universe and family of subsets, respectively. The existence of such an algorithm is open, and it has been conjectured that no such algorithm is possible under SETH; see Cygan et al. [12]. Thus, up to the assumption of this conjecture (called *Set Cover Conjecture* [22]) and SETH, our $\mathcal{O}^*(2^k)$-time algorithm for List Coloring Clique Modulator is best possible w.r.t. its dependency on k.

In Section 4, we consider Pre-Coloring Extension Clique Modulator, which is a subproblem of List Coloring Clique Modulator and prove that Pre-Coloring Extension Clique Modulator, unlike List Coloring Clique Modulator, admits a polynomial kernel: a linear kernel with at most $3k$ vertices. This kernel builds on a known, but counter-intuitive property of bipartite matchings (see Proposition 2), which was previously used in kernelization by Bodlaender et al. [6].

In Section 5, we study an open problem stated by Banik et al. [3]. In a classic result, Chor et al. [9] showed that Coloring has a linear vertex kernel parameterized by $k = n - p$, i.e., if the task is to "save k colors". Arora et al. [2] consider the following as a natural extension to list coloring, and show that it is in XP. Banik et al. [3] show that the problem is FPT, but leave as an open question whether it admits a polynomial kernel.

$(n-k)$-Regular List Coloring parameterized by k

Input: A graph G on n vertices, an integer k, and a list $L(v)$ of exactly $n-k$ colors for every $v \in V(G)$.

Problem: Is there a proper list coloring for G?

We answer this question in affirmative by giving a kernel with $\mathcal{O}(k^2)$ vertices and colors, as well as a compression to a variation of the problem with $\mathcal{O}(k)$ vertices, encodable in $\mathcal{O}(k^2 \log k)$ bits. We note that this compression is asymptotically almost tight, as even 4-Coloring does not admit a compression into $\mathcal{O}(n^{2-\varepsilon})$ bits for any $\varepsilon > 0$ unless the polynomial hierarchy collapses [20].

This kernel is more intricate than the above. Via known reduction rules from Banik et al. [3], we can compute a clique modulator of at most $2k$ vertices (hence our result for List Coloring Clique Modulator also solves $(n-k)$-Regular List Coloring in $2^{O(k)}$ time). However, the usual "crown rules" (as in [9] and in Section 4) are not easily applied here, due to complications with the color lists. Instead, we are able to show a set of $\mathcal{O}(k)$ vertices whose colorability make up the "most interesting" part of the problem, leading to the above-mentioned compression and kernel.

Finally, in Section 6, we consider further natural pre-coloring and list coloring variants of the "saving k colors" problem of Chor et al. [9]. We show that the known fixed-parameter tractability and linear kernelizability [9] carries over to a natural pre-coloring generalization but fails for a more general list coloring variant. Since $(n-k)$-Regular List Coloring was originally introduced in [2] as a list coloring variant of the "saving k colors" problem, it is natural to consider other such variants. We conclude the paper in Section 7, where in particular a number of open questions are discussed.

Omitted proofs are marked by $(\star)$ and can be found in the full version of this paper.

2 Preliminaries

2.1 Graphs, Matchings, and Clique Modulator

We consider finite simple undirected graphs. For basic terminology on graphs, we refer to a standard textbook [13]. Let $H = (V, E)$ be an undirected bipartite graph with bi-partition (A, B). We say that a set C is a *Hall set* for A or B if $C \subseteq A$ or $C \subseteq B$, respectively, and $|N_H(C)| < |C|$. We will need the following well-known properties for matchings.

▶ **Proposition 1** (Hall's Theorem [13]). *Let G be an undirected bipartite graph with bi-partition (A, B). Then G has a matching saturating A if and only if there is no Hall set for A, i.e., for every $A' \subseteq A$, it holds that $|N(A')| \geq |A'|$.*

▶ **Proposition 2** ([6, Theorem 2]). *Let G be a bipartite graph with bi-partition (X, Y) and let X_M be the set of all vertices in X that are endpoints of a maximum matching M of G. Then, for every $Y' \subseteq Y$, it holds that G contains a matching that covers Y' if and only if so does $G[X_M \cup Y]$.*

Clique Modulator. Let G be an undirected graph. We say that a set $D \subseteq V(G)$ is a *clique modulator* for G if $G - D$ is a clique. Since we will use the size of a smallest clique modulator as a parameter for our coloring problems, it is natural to ask whether the following problem can be solved efficiently.

Clique Modulator parameterized by k

Input:	A graph G and an integer k
Problem:	Does G have a clique modulator of size at most k?

The following proposition shows that this is indeed the case. Namely, Clique Modulator is both FPT and can be approximated within a factor of two. The former is important for our FPT algorithms and the later for our kernelization algorithms as it allows us to not depend on a clique modulator given as part of the input.

▶ **Proposition 3.** ($\star$) *Clique Modulator is fixed-parameter tractable (in time $\mathcal{O}^*(1.2738^k)$) and can be approximated within a factor of two.*

2.2 Parameterized Complexity

An instance of a parameterized problem Π is a pair (I, k) where I is the *main part* and k is the *parameter*; the latter is usually a non-negative integer. A parameterized problem is *fixed-parameter tractable* (FPT) if there exists a computable function f such that instances (I, k) can be solved in time $\mathcal{O}(f(k)|I|^c)$ where $|I|$ denotes the size of I and c is an absolute constant. The class of all fixed-parameter tractable decision problems is called FPT and algorithms which run in the time specified above are called FPT algorithms. As in other literature on FPT algorithms, we will often omit the polynomial factor in $\mathcal{O}(f(k)|I|^c)$ and write $\mathcal{O}^*(f(k))$ instead. To establish that a problem under a specific parameterization is not in FPT we prove that it is W[1]-hard as it is widely believed that FPT$\neq$W[1].

A *reduction rule* R for a parameterized problem Π is an algorithm A that given an instance (I, k) of a problem Π returns an instance (I', k') of the *same* problem. The reduction rule is said to be *safe* if it holds that $(I, k) \in \Pi$ if and only if $(I', k') \in \Pi$. If A runs in polynomial time in $|I| + k$ then R is a *polynomial-time reduction rule*. Often we omit the adjectives "safe" and "polynomial-time" in "safe polynomial-time reduction rule" as we consider only such reduction rules.

A *kernelization* (or, a *kernel*) of a parameterized problem Π is a reduction rule such that $|I'| + k' \leq f(k)$ for some computable function f. Note that a decidable parameterized problem is FPT if and only if it admits a kernel [11, 15, 17]. The function f is called the *size* of the kernel, and we have a *polynomial kernel* if $f(k)$ is polynomially bounded in k.

A kernelization can be generalized by considering a reduction (rule) from a parameterized problem Π to another parameterized problem Π'. Then instead of a kernel we obtain a *generalized kernel* (also called a bikernel [1] in the literature). If the problem Π' is not parameterized, then a reduction from Π to Π' (i.e., (I, k) to I') is called a *compression*, which is *polynomial* if $|I'| \leq p(k)$, where p is a fixed polynomial in k. If there is a polynomial compression from Π to Π' and Π' is polynomial-time reducible back to Π, then combining the compression with the reduction gives a polynomial kernel for Π.

3 List Coloring Clique Modulator

The following lemma is often used in the design of randomized algorithms.

▶ **Lemma 4.** *(Schwartz-Zippel [26, 30]). Let $P(x_1, \ldots, x_n)$ be a multivariate polynomial of total degree at most d over a field $\mathbb{F}$, and assume that P is not identically zero. Pick $r_1, \ldots, r_n$ uniformly at random from $\mathbb{F}$. Then $Pr[P(r_1, \ldots, r_n) = 0] \leq d/|\mathbb{F}|$.*

Both parts of the next lemma will be used in this section. The part for fields of characteristic two was proved by Wahlström [28]. The part for reals can be proved similarly.

▶ **Lemma 5.** *Let $P(x_1, \ldots, x_n)$ be a polynomial over a field of characteristic two (over reals, respectively), and $J \subseteq [n]$ a set of indices. For a set $I \subseteq [n]$, define $P_{-I}(x_1, \ldots, x_n) = P(y_1, \ldots, y_n)$, where $y_i = 0$ for $i \in I$ and $y_i = x_i$, otherwise. Define*

$$Q(x_1, \ldots, x_n) = \sum_{I \subseteq J} P_{-I}(x_1, \ldots, x_n)$$

$$(Q(x_1, \ldots, x_n) = \sum_{I \subseteq J} (-1)^{|I|} P_{-I}(x_1, \ldots, x_n), \text{ respectively}).$$

Then for any monomial T divisible by $\Pi_{i \in J} x_i$ we have $\mathrm{coef}_Q T = \mathrm{coef}_P T$, and for every other monomial T we have $\mathrm{coef}_Q T = 0$.

Using the lemmas, we can prove the following:

▶ **Theorem 6.** *List Coloring Clique Modulator can be solved by a randomized algorithm in time $\mathcal{O}^*(2^{k \log k})$.*

Proof. Let $L = \bigcup_{V \in V(G)} L(v)$ and $C = G - D$. We say that a proper list coloring λ for G is compatible with $(\mathcal{D}, \mathcal{D}')$ if:

- $\mathcal{D} = \{D_1, \ldots, D_p\}$ is the partition of all vertices in D that do not reuse colors used by λ in C into color classes given by λ and
- $\mathcal{D} = \{D'_1, \ldots, D'_t\}$ is the partition of all vertices in D that do reuse colors used by λ in C into color classes given by λ.

Note that $\{D_1, \ldots, D_p, D'_1, \ldots, D'_t\}$ is the partition of D into color classes given by λ.

For a given pair $(\mathcal{D}, \mathcal{D}')$, we will now construct a bipartite graph B (with weights on its edges) such that B has a perfect matching satisfying certain additional properties if and only if G has a proper list coloring that is compatible with $(\mathcal{D}, \mathcal{D}')$. B has bi-partition $(C \cup \{D_1, \ldots, D_p\}, L)$ and an edge between a vertex $c \in C$ and a vertex $\ell \in L$ if and only if $\ell \in L(u)$. Moreover, B has an edge between a vertex D_i and a vertex $\ell \in L$ if and only if

$\ell \in \bigcap_{d \in D_i} L(d)$. Finally, if $c \in C$ and $\ell \in L$, then assign the edge $c\ell$ weight $\sum_{j \in J} x_j$, where x_j's are variables and $j \in J$ if and only if $\ell \in (\bigcap_{d \in D'_j} L(d)) \cap L(c)$ and c is not adjacent to any vertex in D'_j. All other edges in B are given weight 1. In the following we will assume that B is balanced; if this is not the case then we simply add the right amount of dummy vertices to the smaller side and make them adjacent (with an edge of weight 1) to all vertices in the opposite side. Note that B has a perfect matching M such that there is a bijection α between $[t]$ and t edges in M such that for every $i \in [t]$, the weight of the edge $\alpha(i)$ contains the term x_i if and only if G has a proper list coloring that is compatible with $(\mathcal{D}, \mathcal{D}')$.

Let M be the weighted incidence matrix of B, i.e., M is an $|V(B)/2| \times |V(B)/2|$ matrix such that its entries $L_{i,j}$ equal to the weight of the edge between the i-th vertex on one side and the j-th vertex on the other side of B if it exists and $L_{i,j} = 0$ otherwise.

Note that the permanent $\mathrm{per}(M)$ of M equals to the sum of the products of entries of M, where each product corresponds to a perfect matching Q of B and is equal to the product of the entries of M corresponding to the edges of Q. Some of the entries of M contain sums of variables x_j, $j \in [t]$ and thus $\mathrm{per}(M)$ is a polynomial in these variables.

Now it is not hard to see that $\mathrm{per}(M)$ contains the monomial $\prod_{j=1}^{t} x_j$ if and only if B has a perfect matching M such that there is a bijection α between $[t]$ and t edges in M such that for every $i \in [t]$, the weight of the edge $\alpha(i)$ contains the term x_i, which in turn is equivalent to G having a proper list coloring that is compatible with $(\mathcal{D}, \mathcal{D}')$.

Hence, deciding whether G has a proper list coloring that is compatible with $(\mathcal{D}, \mathcal{D}')$ boils down to deciding whether the permanent of M contains the monomial $\prod_{j=1}^{t} x_j$. For any evaluation of variables x_j, we can compute $\mathrm{per}(M)$ over the field of characteristic two by replacing permanent with determinant, which can be computed in polynomial-time [7].

Now let $P(x_1, \ldots, x_t) = \det(M)$ and $Q(x_1, \ldots, x_t) = \sum_{I \subseteq [t]} P_{-I}(x_1, \ldots, x_t)$. Note that $Q(x_1, \ldots, x_t) \neq 0$ if and only if $\det(M)$ contains the monomial $\prod_{j=1}^{t} x_j$. Moreover, using Lemmas 4 and 5 (with P and Q just defined), we can verify in time $\mathcal{O}^*(2^t)$ whether $Q(x_1, \ldots, x_t) = 0$ (i.e. whether $\det(M)$ contains the monomial $\prod_{j=1}^{t} x_j$) with probability at least $1 - \frac{t}{|\mathbb{F}|} \geq 1 - \frac{1}{t}$ for a field $\mathbb{F}$ of characteristic 2 such that $|\mathbb{F}| \geq t^2$.

Our algorithm sets $t = k$ and for every pair $(\mathcal{D}, \mathcal{D}')$, where $\mathcal{D} \cup \mathcal{D}'$ is a partition of D into independent sets, constructs graph B and matrix M. It then verifies in time $\mathcal{O}^*(2^t)$ whether $Q(x_1, \ldots, x_t) = 0$ and if $Q(x_1, \ldots, x_t) \neq 0$ it returns "Yes" and terminates. If the algorithm runs to the end, it returns "No".

Note that the time complexity of the algorithm is dominated by the number of choices for $(\mathcal{D}, \mathcal{D}')$, which is in turn dominated by $\mathcal{O}^*(\mathcal{B}_k)$, where $\mathcal{B}_k$ is the k-th Bell number. By Berend and Tassa [4], $\mathcal{B}_k < (\frac{0.792k}{\ln(k+1)})^k$, and thus $\mathcal{O}^*(\mathcal{B}_k) = \mathcal{O}^*(2^{k \log k})$. ◀

3.1 A faster FPT algorithm

We now show a faster FPT algorithm, running in time $\mathcal{O}^*(2^k)$. It is a variation on the same algebraic sieving technique as above, but instead of guessing a partition of the modulator it works over a more complex matrix. We begin by defining the matrix, then we show how to perform the sieving step in $\mathcal{O}^*(2^k)$ time.

3.1.1 Matrix definition

As before, let $L = \bigcup_{v \in V(G)} L(v)$ be the set of all colors, and let $C = G - D$. Define an auxiliary bipartite graph $H = (U_H \cup V_H, E_H)$ where initially $U_H = V(G)$ and $V_H = L$, and where $v\ell \in E_H$ for $v \in V(G)$, $\ell \in L$ if and only if $\ell \in L(v)$. Additionally, introduce a set $L' = \{\ell'_d \mid d \in D\}$ of k artificial colors, add L' to V_H, and for each $d \in D$ connect ℓ'_d to d but

to no other vertex. Finally, pad U_H with $|V_H| - |U_H|$ artificial vertices connected to all of V_H; note that this is a non-negative number, since otherwise $|L| < |V(C)|$ and we may reject the instance.

Next, we associate with every edge $v\ell \in E_H$ a set $S(v\ell) \subseteq 2^D$ as follows.

- If $v \in V(C)$, then $S(v\ell)$ contains all sets $S \subseteq D$ such that the following hold: 1. S is an independent set in G, 2. $N(v) \cap S = \emptyset$, 3. $\ell \in \bigcap_{s\in S} L(s)$.
- If $v \in D$ and $\ell \in L$, then $S(v\ell)$ contains all sets $S \subseteq D$ such that the following hold: 1. $v \in S$, 2. S is an independent set in G, 3. $\ell \in \bigcap_{s\in S} L(s)$.
- If v or ℓ is an artificial vertex – in particular, if $\ell = \ell'_d$ for some $d \in D$ – then $S(v\ell) = \{\emptyset\}$.

Finally, define a matrix A of dimensions $|U_H| \times |V_H|$, with rows labeled by U_H and columns labeled by V_H, whose entries are polynomials as follows. Define a set of variables $X = \{x_d \mid d \in D\}$ corresponding to vertices of D, and additionally a set $Y = \{y_e \mid e \in E_H\}$. Then for every edge $v\ell$ in H, $v \in U_H$, $\ell \in V_H$ we define $P(v\ell) = \sum_{S\in S(v\ell)} \prod_{s\in S} x_s$, where as usual an empty product equals 1. Then for each edge $v\ell \in E_H$ we let $A[v, \ell] = y_{v\ell}P(v\ell)$, and the remaining entries of A are 0. We argue the following. (Expert readers may note although the argument can be sharpened to show the existence of a multilinear term, we do not wish to argue that there exists such a term with odd coefficient. Therefore we use the simpler sieving of Lemma 5 instead of full multilinear detection, cf. [11].)

▶ **Lemma 7.** *Let A be defined as above. Then $\det A$ (as a polynomial) contains a monomial divisible by $\prod_{x\in X} x$ if and only if G is properly list colorable.*

Proof. We first note that no cancellation happens in $\det A$. Note that monomials of $\det A$ correspond (many-to-one) to perfect matchings of H, and thanks to the formal variables Y, two monomials corresponding to distinct perfect matchings never interact. On the other hand, if we fix a perfect matching M in H, then the contributions of M to $\det A$ equal $\sigma_M \prod_{e\in M} y_e P(e)$, where $\sigma_M \in \{1, -1\}$ is a sign term depending only on M. Since the polynomials $P(e)$ contain only positive coefficients, no cancellation occur, and every selection of a perfect matching M of H and a factor from every polynomial $P(e)$, $e \in M$ results (many-to-one) to a monomial with non-zero coefficient in $\det A$.

We now proceed with the proof. On the one hand, let c be a proper list coloring of G. Define an ordering $\prec$ on $V(G)$ such that $V(C)$ precedes D, and define a matching M as follows. For every vertex $v \in V(C)$, add $vc(v)$ to M. For every vertex $v \in D$, add $vc(v)$ to M if v is the first vertex according to $\prec$ that uses color $c(v)$, otherwise add $v\ell'_v$ to M. Note that M is a matching in H of $|V(G)|$ edges. Pad M to a perfect matching in H by adding arbitrary edges connected to the artificial vertices in U_H; note that this is always possible. Finally, for every edge $v\ell \in M$ with $\ell \in L$ we let $D_{v\ell} = D \cap c^{-1}(\ell)$. Observe that for every edge $v\ell$ in M, $D_{v\ell} \in S(v\ell)$; indeed, this holds by construction of $S(v\ell)$ and since c is a proper list coloring. Further let $p_{v\ell} = \prod_{v\in D_{v\ell}} x_v$; thus $p_{v\ell}$ is a term of $P(v\ell)$. It follows, by the discussion in the first paragraph of the proof, that

$$\alpha\sigma_M \prod_{v\ell\in M} y_{v\ell}p_{v\ell}$$

is a monomial of $\det A$ for some constant $\alpha > 0$, where $\sigma_M \in \{1, -1\}$ is the sign term for M. It remains to verify that every variable $x_d \in X$ occurs in some term $p_{v\ell}$. Let $\ell = c(d)$ and let v be the earliest vertex according to $\prec$ such that $c(v) = \ell$. Then $v\ell \in M$ and x_d occurs in $p_{v\ell}$. This finishes the first direction of the proof.

On the other hand, assume that $\det A$ contains a monomial T divisible by $\prod_{x\in X} x$, and let M be the corresponding perfect matching of H. Let $T = \alpha \prod_{e\in M} y_e p_e$ for some constant

factor α, where p_e is a term of $P(e)$ for every $e \in M$. Clearly such a selection is possible; if it is ambiguous, make the selection arbitrarily. Now define a mapping $c\colon V(G) \to L$ as follows. For $v \in V(C)$, let $v\ell \in M$ be the unique edge connected to v, and set $c(v) = \ell$. For $v \in D$, let v' be the earliest vertex according to $\prec$ such that x_v occurs in $p_{v'\ell}$, where $v'\ell \in M$. Set $c(v) = \ell$. We verify that c is a proper list coloring of G. First of all, note that $c(v)$ is defined for every $v \in V(G)$ and that $c(v) \in L(v)$. Indeed, if $v \in V(C)$ then $c(v) \in L(v)$ since $vc(v) \in E_H$; and if $v \in D$ then $c(v) \in L(v)$ is verified in the creation of the term $p_{vc(v)}$ in $P(vc(v))$. Next, consider two vertices $u, v \in V(G)$ with $c(u) = c(v)$. If $u, v \in D$, then u and v are represented in the same term $p_{v'c(v)}$ for some v', hence u and v form an independent set; otherwise assume $u \in V(C)$. Note that $u, v \in V(C)$ is impossible since otherwise the matching M would contain two edges $uc(u)$ and $vc(u)$ which intersect. Thus $v \in D$, and v is represented in the term $p_{uc(u)}$. Therefore $uv \notin E(G)$, by construction of $P(uc(u))$. We conclude that c is a proper coloring respecting the lists $L(v)$, i.e., a proper list coloring. ◀

3.1.2 Fast evaluation

By the above description, we can test for the existence of a list coloring of G using 2^k evaluations of $\det A$, as in Theorem 6; and each evaluation can be performed in $\mathcal{O}^*(2^k)$ time, including the time to evaluate the polynomials $P(v\ell)$, making for a running time of $\mathcal{O}^*(4^k)$ in total (or $\mathcal{O}^*(3^k)$ with more careful analysis). We show how to perform the entire sieving in time $\mathcal{O}^*(2^k)$ using fast subset convolution.

For $I \subseteq D$, let us define A_{-I} as A with all occurrences of variables x_i, $i \in I$ replaced by 0, and for every edge $v\ell$ of H, let $P(v\ell)_{-I}$ denote the polynomial $P(v\ell)$ with x_i, $i \in I$ replaced by 0. Then a generic entry (v, ℓ) of A_{-I} equals $A_{-I}[v, \ell] = y_{v\ell}P_{-I}(v\ell)$, and in order to construct A_{-I} it suffices to pre-compute the value of $P_{-I}(v\ell)$ for every edge $v\ell \in E_H$, $I \subseteq D$. For this, we need the *fast zeta transform* of Yates [29], which was introduced to exact algorithms by Björklund et al. [5].

▶ **Lemma 8** ([29, 5])**.** *Given a function $f : 2^N \to R$ for some ground set N and ring R, we may compute all values of $\hat{f} : 2^N \to R$ defined as $\hat{f}(S) = \sum_{A \subseteq S} f(A)$ using $\mathcal{O}^*(2^{|N|})$ ring operations.*

We show the following lemma, which is likely to have analogs in the literature, but we provide a short proof for the sake of completeness.

▶ **Lemma 9.** *Given an evaluation of the variables X, the value of $P_{-I}(v\ell)$ can be computed for all $I \subseteq D$ and all $v\ell \in E_H$ in time and space $\mathcal{O}^*(2^k)$.*

Proof. Consider an arbitrary polynomial $P_{-I}(v\ell)$. Recalling $P(v\ell) = \sum_{S \in S(v\ell)} \prod_{s \in S} x_s$, we have

$$P_{-I}(v\ell) = \sum_{S \in S(v\ell)} [S \cap I = \emptyset] \prod_{s \in S} x_s = \sum_{S \subseteq (D-I)} [S \in S(v\ell)] \prod_{s \in S} x_s,$$

using Iverson bracket notation.[1] Using $f(S) = [S \in S(v\ell)] \prod_{s \in S} x_s$, this clearly fits the form of Lemma 8, with $\hat{f}(D - I) = P_{-I}(v\ell)$. Hence we apply Lemma 8 for every edge $v\ell \in E_H$, for $\mathcal{O}^*(2^k)$ time per edge, making $\mathcal{O}^*(2^k)$ time in total to compute all values. ◀

Having access to these values, it is now easy to complete the algorithm.

[1] Recall that for a logical proposition P, $[P] = 1$ if P is true and 0, otherwise.

▶ **Theorem 10.** *List Coloring Clique Modulator can be solved by a randomized algorithm in time $\mathcal{O}^*(2^k)$.*

Proof. Let A be the matrix defined above (but do not explicitly construct it yet). By Lemma 7, we need to check whether $\det A$ contains a monomial divisible by $\prod_{x\in X} x$, and by Lemma 5 this is equivalent to testing whether $\sum_{I\subseteq D}(-1)^{|I|}\det A_{-I} \not\equiv 0$. By the Schwartz-Zippel lemma, it suffices to randomly evaluate the variables X and Y occurring in A and evaluate this sum once; if G has a proper list coloring and if the values of X and Y are chosen among sufficiently many values, then with high probability the result is non-zero, and if not, then the result is guaranteed to be zero. Thus the algorithm is as follows.

1. Instantiate variables of X and Y uniformly at random from $[N]$ for some sufficiently large N. Note that for an error probability of $\varepsilon > 0$, it suffices to use $N = \Omega(n^2(1/\varepsilon))$.
2. Use Lemma 9 to fill in a table with the value of $P_{-I}(v\ell)$ for all I and $v\ell$ in time $\mathcal{O}^*(2^k)$.
3. Compute $\sum_{I\subseteq D}(-1)^{|I|}\det A_{-I}$, constructing A_{-I} from the values $P_{-I}(v\ell)$ in polynomial time in each step.
4. Answer YES if the result is non-zero, NO otherwise.

Clearly this runs in total time and space $\mathcal{O}^*(2^k)$ and the correctness follows from the arguments above. ◀

3.2 Refuting Polynomial Kernel

In this section, we prove that List Coloring Clique Modulator does not admit a polynomial kernel. We prove this result by a polynomial parameter transformation from Hitting Set where the parameter is the number of sets, which is known not to have a polynomial kernel [14].

▶ **Theorem 11.** (⋆) *List Coloring Clique Modulator parameterized by k does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly.*

We note here that the reduction also shows that if List Coloring Clique Modulator could be solved in time $\mathcal{O}(2^{\epsilon k}n^{\mathcal{O}(1)})$ for some $\epsilon < 1$, then Hitting Set could be solved in time $\mathcal{O}(2^{\epsilon|\mathcal{F}|}|U|^{\mathcal{O}(1)})$, which in turn would imply that any instance I with universe U and set family $\mathcal{F}$ of the well-known Set Cover problem could be solved in time $\mathcal{O}(2^{\epsilon|U|}|\mathcal{F}|^{\mathcal{O}(1)})$. The existence of such an algorithm is open, and it has been conjectured that no such algorithm is possible under SETH (the strong exponential-time hypothesis); see Cygan et al. [12]. Thus, up to the assumption of this conjecture and SETH, the algorithm for List Coloring Clique Modulator given in Theorem 10 is best possible w.r.t. its dependency on k.

4 Polynomial kernel for Pre-Coloring Extension Clique Modulator

In the following let $(G, D, k, \lambda_P, X, Q)$ be an instance of Pre-Coloring Extension Clique Modulator, let $C = G - D$, let D_P be the set of all pre-colored vertices in D, and let $D' = D \setminus D_P$.

▶ **Reduction Rule 1.** *Remove any vertex $v \in D'$ that has less than $|Q|$ neighbors in G.*

The proof of the following lemma is obvious and thus omitted.

▶ **Lemma 12.** *Reduction Rule 1 is safe and can be implemented in polynomial time.*

Note that if Reduction Rule 1 can no longer be applied, then every vertex in D' has at least $|Q|$ neighbors, which because of $|Q| \geq |C|$ implies that every such vertex has at most $|D| \leq k$ non-neighbors in G and hence also in C. Let C_N be the set of all vertices in C that are not adjacent to all vertices in D' and let $C' = C - C_N$. Note that $|C_N| \leq |D||D| \leq k^2$.

We show next how to reduce the size of C_N to k. Note that this step is optional if our aim is solely to obtain a polynomial kernel, however, it allows us to reduce the number of vertices in the resulting kernel from $\mathcal{O}(k^2)$ to $\mathcal{O}(k)$. Let J be the bipartite graph with partition (C_N, D) having an edge between $c \in C_N$ and $d \in D$ if $\{c, d\} \notin E(G)$.

▶ **Reduction Rule 2.** *If $A \subseteq C_N$ is an inclusion-wise minimal set satisfying $|A| > |N_J(A)|$, then remove the vertices in $D' \cap N_J(A)$ from G.*

Note that after the application of Reduction Rule 2, the vertices in A are implicitly removed from C_N and added to C' since all their non-neighbors in D' (i.e. the vertices in $D' \cap N_J(A)$) are removed from the graph.

▶ **Lemma 13.** *Reduction Rule 2 is safe and can be implemented in polynomial time.*

Proof. It is clear that the rule can be implemented in polynomial-time. Towards showing the safeness of the rule, it suffices to show that G has a coloring extending λ_P using only colors from Q if and only if so does $G \setminus (D' \cap N_J(A))$. Since $G \setminus (D' \cap N_J(A))$ is a subgraph of G, the forward direction of this statement is trivial. So assume that $G \setminus (D' \cap N_J(A))$ has a coloring λ extending λ_P using only colors from Q. Because the set A is inclusion-minimal, we obtain from Proposition 1, that there is a (maximum) matching, say M, between $N_J(A)$ and A in J that saturates $N_J(A)$. Moreover, it follows from the definition of J that every vertex in A is adjacent to every vertex in $D \setminus N_J(A)$ in the graph G. Hence, we obtain that every color in $\lambda(A)$ appears exactly once. Hence, we can extend λ into a coloring λ' for G by coloring the vertices in $D' \cap N_J(A)$ according to the matching M. More formally, let $\lambda_{D' \cap N_J(A)}$ be the coloring for the vertices in $D' \cap N_J(A)$ by setting $\lambda_{D' \cap N_J(A)}(v) = \lambda(u)$ for every $v \in D' \cap N_J(A)$, where $\{v, u\} \in M$. Then, we obtain λ' by setting: $\lambda'(v) = \lambda(v)$ for every $v \in V(G) \setminus (D' \cap N_J(A))$ and $\lambda'(v) = \lambda_{D' \cap N_J(A)}(v)$ for every vertex $v \in D' \cap N_J(A)$. ◀

Note that because of Proposition 1, we obtain that there is a set $A \subseteq C_N$ with $|A| > |N_J(A)|$ as long as $|C_N| > |D|$. Moreover, since $N_J(A) \cap D' \neq \emptyset$ for every such set A (due to the definition of C_N), we obtain that Reduction Rule 2 is applicable as long as $|C_N| > |D|$. Hence after an exhaustive application of Reduction Rule 2, we obtain that $|C_N| \leq |D'| \leq k$.

We now introduce our final two reduction rules, which allow us to reduce the size of C'.

▶ **Reduction Rule 3.** *Let $v \in V(C')$ be a pre-colored vertex with color $\lambda_P(v)$. Then remove $\lambda_P^{-1}(\lambda_P(v))$ from G and $\lambda_P(v)$ from Q.*

▶ **Lemma 14.** *Reduction Rule 3 is safe and can be implemented in polynomial time.*

Proof. Because $v \in V(C')$, it holds that only vertices in D_P can have color $\lambda_P(v)$, but these are already pre-colored. Hence in any coloring for G that extends λ_P, the vertices in $\lambda_P^{-1}(\lambda_P(v))$ are the only vertices that obtain color $\lambda_P(v)$, which implies the safeness of the rule. ◀

Because of Reduction Rule 3, we can from now on assume that no vertex in C' is pre-colored. Note that the only part of G, whose size is not yet bounded by a polynomial in the parameter k is C'. To reduce the size of C', we need will make use of Proposition 2. Let $P = \lambda_P(D_P)$ and H be the bipartite graph with bi-partition (C', P) containing an edge between $c' \in C'$ and $p \in P$ if and only if c' is not adjacent to a vertex pre-colored by p in G.

▶ **Reduction Rule 4.** *Let M be a maximum matching in H and let C_M be the endpoints of M in C'. Then remove all vertices in $C_{\overline{M}} := C' \setminus C_M$ from G and remove an arbitrary set of $|C_{\overline{M}}|$ colors from $Q \setminus \lambda_P(X)$. (Recall that $\lambda_P : X \to Q$.)*

In the following let C_M and $C_{\overline{M}}$ be as defined in the above reduction rule for an arbitrary maximum matching M of H. To show that the reduction rule is safe, we need the following auxiliary lemma, which shows that if a coloring for G reuses colors from P in C', then those colors can be reused solely on the vertices in C_M.

▶ **Lemma 15.** *If there is a coloring λ for G extending λ_P using only colors in Q, then there is a coloring λ' for G extending λ_P using only colors in Q such that $\lambda'(C_{\overline{M}}) \cap P = \emptyset$.*

Proof. Let C_P be the set of all vertices v in C' with $\lambda(v) \in P$. If $C_P \cap C_{\overline{M}} = \emptyset$, then setting λ' equal to λ satisfies the claim of the lemma. Hence assume that $C_P \cap C_{\overline{M}} \neq \emptyset$. Let N be the matching in H containing the edges $\{v, \lambda(v)\}$ for every $v \in C_P$; note that N is indeed a matching in H, because C_P is a clique in G. Because of Proposition 2, there is a matching N' in $H[C_M \cup P]$ such that N' has exactly the same endpoints in P as N. Let $C_M[N']$ be the endpoints of N' in C_M and let λ_A be the coloring of the vertices in $C_M[N']$ corresponding to the matching N', i.e., a vertex v in $C_M[N']$ obtains the unique color $p \in P$ such that $\{v, p\} \in N'$. Finally, let α be an arbitrary bijection between the vertices in $(V(N) \cap C') \setminus C_M[N']$ and the vertices in $C_M[N'] \setminus (V(N) \cap C')$, which exists because $|N| = |N'|$. We now obtain λ' from λ by setting $\lambda'(v) = \lambda_A(v)$ for every $v \in C_M[N']$, $\lambda'(v) = \lambda(\alpha(v))$ for every vertex $v \in (V(N) \cap C') \setminus C_M[N']$, and $\lambda'(v) = \lambda(v)$ for every other vertex. To see that λ' is a proper coloring note that $\lambda'(C') = \lambda(C')$. Moreover, all the colors in $\lambda(C') \setminus P$ are "universal colors" in the sense that exactly one vertex of G obtains the color and hence those colors can be freely moved around in C'. Finally, the matching N' in H ensures that the vertices in $C_M[N']$ can be colored using the colors from P. ◀

▶ **Lemma 16.** *Reduction Rule 4 is safe and can be implemented in polynomial time.*

Proof. Note first that the reduction can always be applied since if $Q \setminus \lambda_P(X)$ contains less than $|C_{\overline{M}}|$ colors, then the instance is a no-instance. It is clear that the rule can be implemented in polynomial time using any polytime algorithm for finding a maximum matching. Moreover, if the reduced graph has a coloring extending λ_P using only the colors in Q, then so does the original graph, since the vertices in $C_{\overline{M}}$ can be colored with the colors removed from the original instance.

Hence, it remains to show that if G has a coloring, say λ, extending λ_P using only colors in Q, then $G \setminus C_{\overline{M}}$ has a coloring extending λ_P that uses only colors in $Q' := Q \setminus Q_{\overline{M}}$, where $Q_{\overline{M}}$ is the set of $|C_{\overline{M}}|$ colors from $Q \setminus \lambda_P(X)$ that have been removed from Q.

Because of Lemma 15, we may assume that $\lambda(C_{\overline{M}}) \cap P = \emptyset$. Let B be the set of all vertices v in $G - C_{\overline{M}}$ with $\lambda(v) \in Q_{\overline{M}}$. If $B = \emptyset$, then λ is a coloring extending λ_P using only colors from Q'. Hence assume that $B \neq \emptyset$. Let A be the set of all vertices v in $C_{\overline{M}}$ with $\lambda(v) \in Q'$. Then $\lambda(A) \cap \lambda_P(X) = \emptyset$, which implies that every color in $\lambda(A)$ appears only in $C_{\overline{M}}$ (and exactly once in $C_{\overline{M}}$). Moreover, $|\lambda(A)| \geq |\lambda(B)|$. Let α be an arbitrary bijection between $\lambda(B)$ and an arbitrary subset of $\lambda(A)$ (of size $|B|$) and let λ' be the coloring obtained from λ by setting $\lambda'(v) = \alpha(\lambda(v))$ for every $v \in B$, $\lambda'(v) = \alpha^{-1}(\lambda(v))$ for every $v \in A$, and $\lambda'(v) = \lambda(v)$, otherwise. Then λ' restricted to $G - C_{\overline{M}}$ is a coloring for $G - C_{\overline{M}}$ extending λ_P using only colors from Q'. Note that λ' is a proper coloring because the colors in $\lambda(A)$ are not in P and hence do not appear anywhere else in G and moreover the colors in $\lambda(B)$ do not appear in $\lambda(C_{\overline{M}})$. ◀

Note that after the application of Reduction Rule 4, it holds that $|C'| = |C_M| \leq |P| \leq |D_P| \leq |D| \leq k$. Together with the facts that $|D| \leq k$, $|C_N| \leq k$, we obtain that the reduced graph has at most $3k$ vertices.

▶ **Theorem 17.** *Pre-Coloring Extension Clique Modulator admits a polynomial kernel with at most $3k$ vertices.*

5 Polynomial kernel and Compression for $(n-k)$-Regular List Coloring

We now show our polynomial kernel and compression for $(n-k)$-Regular List Coloring, which is more intricate than the one for Pre-Coloring Extension Clique Modulator. Let (G, k, L) be an input of $(n-k)$-Regular List Coloring. We begin by noting that we can assume that G has a clique-modulator of size at most $2k$.

▶ **Lemma 18** ([3])**.** *In polynomial-time either we can either solve (G, k, L) or compute a clique-modulator for G of size at most $2k$.*

Henceforth, we let $V(G) = C \cup D$ where $G[C]$ is a clique and D is a clique modulator, $|D| \leq 2k$. Let $T = \bigcup_{v \in V(G)} L(v)$. We note one further known reduction rules for $(n-k)$-Regular List Coloring. Consider the bipartite graph H_G with bi-partition $(V(G), T)$ having an edge between $v \in V(G)$ and $t \in T$ if and only if $t \in L(v)$.

▶ **Reduction Rule 5** ([3])**.** *Let T' be an inclusion-wise minimal subset of T such that $|N_{H_G}(T')| < |T'|$, then remove all vertices in $N_{H_G}(T')$ from G.*

Note that after an exhaustive application of Reduction Rule 5, it holds that $|T| \leq |V(G)|$ since otherwise Proposition 1 would ensure the applicability of the reduction rule. Hence in the following we will assume that $|T| \leq |V(G)|$.

With this preamble handled, let us proceed with the kernelization. We are not able to produce a direct "crown reduction rule" for List Coloring, as for Pre-Coloring Extension (e.g., we do not know of a useful generalization of Reduction Rule 2). Instead, we need to study more closely which list colorings of $G[D]$ extend to list colorings of G. For this purpose, let $H = H_G - D$ be the bipartite graph with bi-partition (C, T) having an edge $\{c, t\}$ with $c \in C$ and $t \in T$ if and only if $t \in L(c)$. Say that a partial list coloring $\lambda_0 \colon A \to T$ is *extensible* if it can be extended to a proper list coloring λ of G. If $D \subseteq A$, then a sufficient condition for this is that $H - (A \cup \lambda_0(A))$ admits a matching saturating $C \setminus A$. (This is not a necessary condition, since some colors used in $\lambda_0(D)$ could be reused in $\lambda(C \setminus A)$, but this investigation will point in the right direction.) By Proposition 1, this is characterized by Hall sets in $H - (A \cup \lambda_0(A))$.

A Hall set $S \subseteq U$ in a bipartite graph G' with bi-partition (U, W) is *trivial* if $N(S) = W$. We start by noting that if a color occurs in sufficiently many vertex lists in H, then it behaves uniformly with respect to extensible partial colorings λ_0 as above.

▶ **Lemma 19.** *Let $\lambda_0 \colon A \to T$ be a partial list coloring where $|A \cap C| \leq p$ and let $t \in T$ be a color that occurs in at least $k + p$ lists in C. Then t is not contained in any non-trivial Hall set of colors in $H - (A \cup \lambda_0(A))$.*

Proof. Let $H' = H - (A \cup \lambda_0(A))$. Consider any Hall set of colors $S \subset (T \setminus \lambda_0(A))$ and any vertex $v \in C \setminus (A \cup N_{H'}(S))$ (which exists assuming S is non-trivial). Then $S \subseteq T \setminus L(v)$, hence $|S| \leq k$, and by assumption $|N_{H'}(S)| < |S|$. But for every $t' \in S$, we have $N_H(t') \subseteq N_{H'}(S) \cup (A \cap C)$, hence t' occurs in at most $|N_{H'}(S) \cup (A \cap C)| < k + p$ vertex lists in C. Thus $t \notin S$. ◀

In the following, we will assume that $n \geq 11k$.[2] This is safe, since otherwise (by Reduction Rule 5) we already have a kernel with a linear number of vertices and colors. We say that a color $t \in T$ is *rare* if it occurs in at most $6k$ lists of vertices in C.

▶ **Lemma 20.** *If $n \geq 11k$, then there are at most $3k$ rare colors.*

Proof. Let $S = \{t \in T \mid d_H(t) < 6k\}$. For every $t \in S$, there are $|C| - 6k$ "non-occurrences" (i.e., vertices $v \in C$ with $t \notin L(v)$), and there are $|C|k$ non-occurrences in total. Thus

$$|S| \cdot (|C| - 6k) \leq |C|k \quad \Rightarrow \quad |S| \leq \frac{|C|}{|C| - 6k}k = (1 + \frac{6k}{|C| - 6k})k,$$

where the bound is monotonically decreasing in $|C|$ and maximized (under the assumption that $n \geq 11k$ and hence $|C| \geq 9k$) for $|C| = 9k$ yielding $|S| \leq 3k$. ◀

Let $T_R \subseteq T$ be the set of rare colors. Define a new auxiliary bipartite graph H^* with bi-partition $(C, D \cup T_R)$ having an edge between a vertex $c \in C$ and a vertex $d \in D$ if $\{c, d\} \notin E(G)$ and an edge between a vertex $c \in C$ and a vertex $t \in T_R$ if $t \in L(c)$. Let X be a minimum vertex cover of H^*. Refer to the colors $T_R \setminus X$ as *constrained* rare colors. Note that constrained rare colors only occur on lists of vertices in $D \cup (C \cap X)$. Let $T' = T \setminus (T_R \setminus X)$, $V' = (D \setminus X) \cup (C \cap X)$, and set $q = |T'| - |C \setminus X|$. Before we continue, we want to provide some useful observations about the sizes of the considered sets and numbers.

▶ **Observation 1.** *It holds that:*

- $|X| \leq |D| + |T_R| \leq 5k$,
- $|V'| \leq |D| + |X| \leq 7k$,
- $q \leq |T| - |C| + |C \cap X| \leq |D| + |X| \leq 7k$; *this holds because* $|T| \leq |V| = |C| + |D|$.

▶ **Lemma 21.** *Assume $n \geq 11k$. Then G has a list coloring if and only if there is a partial list coloring $\lambda_0 \colon V' \to T$ that uses at most $q = |T'| - |C \setminus X|$ colors from T'.*

Proof. The number of colors usable in $C \setminus X$ is $|T'| - p$ where p is the number counted above (since constrained rare colors cannot be used in $C \setminus X$ even if they are unused in λ_0). Thus it is a requirement that $|T'| - p \geq |C \setminus X|$. That is, $p \leq |T'| - |C \setminus X| = q$. Thus necessity is clear. We show sufficiency as well. That is, let λ_0 be a partial list coloring with scope $V' = (C \cap X) \cup (D \setminus X)$ which uses at most q colors of T'. We modify and extend λ_0 to a list coloring of G.

First let H_0 be the bipartite graph with bi-partition $(V, T_R \setminus X)$ and let M_0 be a matching saturating $T_R \setminus X$; note that this exists by reduction rule 5. We modify λ_0 to a coloring λ_0' so that every constrained rare color is used by λ_0', by iterating over every color $t \in T_R \setminus X$; for every t, if t is not yet used by λ_0', then let $vt \in M_0$ and update λ_0' with $\lambda_0'(v) = t$. Note that the scope of λ_0' after this modification is contained in $(C \cap X) \cup D$. Next, let M be a maximum matching in H^*. We use M to further extend λ_0' in stages to a partial list coloring λ which colors all of D and uses all rare colors. In phase 1, for every color $t \in T_R \cap X$ which is not already used, let $vt \in M$ be the edge covering t and assign $\lambda(v) = t$. Note that M matches every vertex of X in H^* with a vertex not in X, thus the edge vt exists and v has not yet been assigned in λ. Hence, at every step we maintain a partial list coloring, and at the end of the phase all rare colors have been assigned. Finally, as phase 2, for

[2] The constants $11k$ and $6k$ in this paragraph are chosen to make the arguments work smoothly. A smaller kernel is possible with a more careful analysis and further reduction rules.

every vertex $v \in D \cap X$ not yet assigned, let $uv \in M$ where $u \in C$; necessarily $u \in C \setminus X$ and u is as of yet unassigned in λ. The number of colors assigned in λ thus far is at most $|X| + |D| \leq |T_R| + 2|D| \leq 7k$, whereas $|L(u) \cap L(v)| \geq n - 2k \geq 9k$, hence there always exists an unused shared color that can be mapped to $\lambda(u) = \lambda(v)$. Let λ be the resulting partial list coloring. We claim that λ can be extended to a list coloring of G.

Let A be the scope of λ and let $H' = H - (A \cap \lambda(A))$. Note that $A \cap C \subseteq V(M)$, hence $|A \cap C| \leq |D| + |T_R| \leq 5k$. Thus by Lemma 19, no non-trivial Hall set in H' can contain a rare color. However, all rare colors are already used in λ. Thus H' contains no non-trivial Hall set of colors. Thus the only possibility that λ is not extensible is that H' has a trivial Hall set, i.e., $|T \setminus \lambda(A)| < |C \setminus A|$. However, every modification after λ'_0 added one vertex to A and one color to $\lambda(A)$, keeping the balance between the two sides. Thus already the partial coloring λ'_0 leaves behind a trivial Hall set. However, λ'_0 colors precisely $C \cap X$ in C and leaves at least $|T'| - q$ colors remaining. By design this is at least $|C \setminus X|$, yielding a contradiction. Thus we find that H' contains no Hall set, and λ is a list coloring of G. ◀

Before we give our compression , we need the following auxiliary lemma.

▶ **Lemma 22.** *T' contains at least $|T'| - |V'|k$ colors that are universal to all vertices in V'.*

Proof. The list of every vertex $v \in V'$ misses at most k colors from T'. Hence all but at most $|V'|k$ colors in T' are universal to all vertices in V'. ◀

For clarity, let us define the output problem of our compression explicitly.

Budget-Constrained List Coloring

Input: A graph G, a set T of colors, a list $L(v) \subseteq T$ for every $v \in V(G)$, and a pair (T', q) where $T' \subseteq T$ and $q \in \mathbb{N}$.

Problem: Is there a proper list coloring for G that uses at most q distinct colors from T'?

▶ **Theorem 23.** *$(n-k)$-Regular List Coloring admits a compression into an instance of Budget-Constrained List Coloring with at most $11k$ vertices and $\mathcal{O}(k^2)$ colors, encodable in $\mathcal{O}(k^2 \log k)$ bits.*

Proof. Lemma 21 shows that the existence of a list coloring in G is equivalent to the existence of a list coloring in $G[V']$ that uses at most q colors from T'. Since $|V'| \leq 7k$, it only remains to reduce the number of colors in $T_R \cup T'$. Clearly, if $|T'| < |V'|k + q$, then $|T_R \cup T'| \leq 3k + (7k)k \in \mathcal{O}(k^2)$ and there is nothing left to show. So suppose that $|T'| \geq |V'|k + q$. Then, it follows from Lemma 22 that T' contains at least q colors that are universal to the vertices in V' and we obtain an equivalent instance by removing all but exactly q universal colors from T', which leaves us with an instance with at most $|T_R| + q \leq 3k + 7k^2 \in \mathcal{O}(k^2)$ colors, as required. Finally, to describe the output concisely, note that $G[V']$ can be trivially described in $\mathcal{O}(k^2)$ bits, and the lists $L(v)$ can be described by enumerating $T \setminus L(v)$ for every vertex v, which is k colors per vertex, each color identifiable by $\mathcal{O}(\log k)$ bits. ◀

Note that the compression is asymptotically essentially optimal, since even the basic 4-Coloring problem does not allow a compression in $\mathcal{O}(n^{2-\varepsilon})$ bits for any $\varepsilon > 0$ unless the polynomial hierarchy collapses [20]. For completeness, we also give a proper kernel, which can be obtained in a similar manner to the compression given in Theorem 23.

▶ **Theorem 24.** *$(n-k)$-Regular List Coloring admits a kernel with $\mathcal{O}(k^2)$ vertices and colors.*

Proof. We distinguish two cases depending on whether or not $|T'| < |V'|k + q$. If $|T'| < |V'|k + q$, then $|T| \leq |T_R| + |T'| < 3k + |V'|k + q \leq 3k + (7k)(k + 1) \in \mathcal{O}(k^2)$. Since a list coloring requires at least one distinct color for every vertex in C, it holds that $|C| \leq |T| \leq 3k + (7k)(k + 1)$ and hence $|V(G)| \leq (3 + 7k)k + 2k \in \mathcal{O}(k^2)$, implying the desired kernel.

If on the other hand, $|T'| \geq |V'|k+q$, then, because of Lemma 22 it holds that T' contains a set U of exactly q colors that are universal to the vertices in V'. Recall that Lemma 21 shows that the existence of a list coloring in G is equivalent to the existence of a list coloring in $G[V']$ that uses at most $q = |T'| - |C \setminus X|$ colors from T'. It follows that the graph $G[V']$ has a list coloring using only colors in $(T_R \setminus X) \cup U$ if and only if G has a list coloring. Hence, it only remains to restore the regularity of the instance. We achieve this as follows. First we add a set T_N of $|(T_R \setminus X) \cup U|$ novel colors. We then add these colors (arbitrarily) to the color lists of the vertices in V' such that the size of every list (for any vertex in V') is $|(T_R \setminus X) \cup U|$. This clearly already makes the instance regular, however, now we also need to ensure that no vertex in V' can be colored with any of the new colors in T_N. To achieve this we add a set C_N of $|T_N|$ novel vertices to $G[V']$, which we connect to every vertex in $(C \cap X) \cup C_N$ and whose lists all contain all the new colors in T_N. It is clear that the constructed instance is equivalent to the original instance since all the new colors in T_N are required to color the new vertices in C_N and hence no new color can be used to color a vertex in V'. Moreover, D is still a clique modulator and the number k' of missing colors (in each list of the constructed instance) is equal to $|D| + |C \cap X| \leq 2k + 5k$ because the instance is $(n - |D| - |C \cap X|)$-regular. Finally, the instance has at most $|V' \cup C_N| \leq 7k + 3k + 7k = 17k \in \mathcal{O}(k)$ vertices and at most $2(|T_R| + |U|) \leq 2(3k + 7k) = 20k \in \mathcal{O}(k)$ colors, as required. ◀

6 Saving k colors: Pre-coloring and List Coloring Variants

In this section, we consider natural pre-coloring and list coloring variants of the "saving k colors" problem, which given a graph on n vertices and an integer k asks whether G has a proper coloring with at most $n - k$ colors. This problem is known to be FPT (it even allows for a linear kernel) [9], when parameterized by k. Notably the problem provided the main motivation for the introduction of $(n - k)$-Regular List Coloring in [3, 2].

We consider the following (pre-coloring and list coloring) extensions of $(n-k)$-Coloring.

$(n - |Q|)$-Pre-Coloring Extension parameterized by $n - |Q|$

Input: A graph G with n vertices and a pre-coloring $\lambda_P : X \to Q$ for $X \subseteq V(G)$ where Q is a set of colors.

Problem: Can λ_P be extended to a proper coloring of G using only colors from Q?

List Coloring with $n - k$ colors parameterized by k

Input: A graph G on n vertices with a list $L(v)$ of colors for every $v \in V(G)$ and an integer k.

Problem: Is there a proper list coloring of G using at most $n - k$ colors?

Interestingly, we show that $(n-|Q|)$-Pre-Coloring Extension is FPT and even allows a linear kernel. Thus, we generalize the above-mentioned result of Chor et al. [9] (set $Q=[n-k]$ and $X=\emptyset$). However, List Coloring with $n-k$ colors is easily seen to be NP-hard (even for $k=0$) using a trivial reduction from 3-Coloring.

▶ **Theorem 25.** ($\star$) *$(n-|Q|)$-Pre-Coloring Extension (parameterized by $n-|Q|$) has a kernel with at most $6(n-|Q|)$ vertices and is hence fixed-parameter tractable.*

7 Conclusions

We have shown several results regarding the parameterized complexity of List Coloring and Pre-Coloring Extension problems. We showed that List Coloring, and hence also Pre-Coloring Extension, parameterized by the size of a clique modulator admits a randomized FPT algorithm with a running time of $\mathcal{O}^*(2^k)$, matching the best known running time of the basic Chromatic Number problem parameterized by the number of vertices. This answers open questions of Golovach et al. [19]. Note that also that ListColoring is already W[1]-hard parameterized by vertex cover [19], i.e., modulator to an independent set, which excludes even quite simple generalizations of our result to, e.g., a modulator to a disjoint union of cliques. Additionally, we showed that Pre-Coloring Extension under the same parameter admits a linear vertex kernel with at most $3k$ vertices and that $(n-k)$-Regular List Coloring admits a compression into a problem we call Budget-Constrained List Coloring, into an instance with at most $11k$ vertices, encodable in $\mathcal{O}(k^2 \log k)$ bits. The latter also admits a proper kernel with $\mathcal{O}(k^2)$ vertices and colors. This answers an open problem of Banik et al. [3].

One obvious open question is whether it is possible to derandomize our algorithm for List Coloring. This seems, however, very challenging as it would require a derandomization of Lemma 4, which has been an open problem for some time. It might, however, be possible (and potentially more promising) to consider a different approach than ours. Another open question is to optimize the bound $11k$ on the number of vertices in the $(n-k)$-Regular List Coloring compression, and/or show a proper kernel with $\mathcal{O}(k)$ vertices. Finally, another set of questions is raised by Escoffier [16], who studied the Max Coloring problem from a "saving colors" perspective. In addition to the questions explicitly raised by Escoffier, it is natural to ask whether his problems Saving Weight and Saving Color Weights admit FPT algorithms with a running time of $2^{\mathcal{O}(k)}$ and/or polynomial kernels.

References

1 Noga Alon, Gregory Z. Gutin, Eun Jung Kim, Stefan Szeider, and Anders Yeo. Solving MAX-r-SAT above a tight lower bound. *Algorithmica*, 61(3):638–655, 2011.

2 Pranav Arora, Aritra Banik, Vijay Kumar Paliwal, and Venkatesh Raman. Some (in)tractable parameterizations of coloring and list-coloring. In Jianer Chen and Pinyan Lu, editors, *Frontiers in Algorithmics - 12th International Workshop, FAW 2018, Proceedings*, volume 10823 of *Lecture Notes in Computer Science*, pages 126–139. Springer, 2018.

3 Aritra Banik, Ashwin Jacob, Vijay Kumar Paliwal, and Venkatesh Raman. Fixed-parameter tractability of $(n-k)$ List Coloring. In C.J. Colbourn, R. Grossi, and N. Pisani, editors, *IWOCA 2019, Proceedings*, volume 11638 of *Lecture Notes in Computer Science*, pages 61–69, 2019. doi:10.1007/978-3-030-25005-8_6.

4 D. Berend and T. Tassa. Improved bounds on bell numbers and on moments of sums of random variables. *Probability and Mathematical Statistics*, 30:185–205, 2010.

5 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009.

6 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernel bounds for path and cycle problems. *Theor. Comput. Sci.*, 511:117–136, 2013.

7 James R. Bunch and John E. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28(125):231–236, 1974.

8 Leizhen Cai. Parameterized complexity of vertex colouring. *Discrete Applied Mathematics*, 127(3):415–429, 2003.

9 Benny Chor, Mike Fellows, and David Juedes. Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. In *Proceedings of the 30th International Conference on Graph-Theoretic Concepts in Computer Science*, WG'04, pages 257–269. Springer-Verlag, 2004.

10 Maria Chudnovsky. Coloring graphs with forbidden induced subgraphs. In *2014 International Congress of Mathematicians*, volume IV, pages 291–302, 2014.

11 M. Cygan, F.V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

12 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3):41:1–41:24, 2016.

13 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

14 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Kernelization Lower Bounds Through Colors and IDs. *ACM Trans. Algorithms*, 11(2):13:1–13:20, 2014.

15 R.G. Downey and M.R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.

16 Bruno Escoffier. Saving colors and max coloring: Some fixed-parameter tractability results. *Theor. Comput. Sci.*, 758:30–41, 2019. doi:10.1016/j.tcs.2018.08.002.

17 F.V. Fomin, D. Lokshtanov, S. Saurabh, and M. Zehavi. *Kernelization*. Cambridge Univ. Press, 2019.

18 Petr A. Golovach, Matthew Johnson, Daniël Paulusma, and Jian Song. A survey on the computational complexity of coloring graphs with forbidden subgraphs. *Journal of Graph Theory*, 84(4):331–363, 2017.

19 Petr A. Golovach, Daniël Paulusma, and Jian Song. Closing complexity gaps for coloring problems on h-free graphs. *Inf. Comput.*, 237:204–214, 2014.

20 Bart M. P. Jansen and Astrid Pieterse. Sparsification upper and lower bounds for graph problems and not-all-equal SAT. *Algorithmica*, 79(1):3–28, 2017.

21 Tommy R. Jensen and Bjarne Toft. *Graph Coloring Problems*. Wiley & Sons, 1994.

22 Robert Krauthgamer and Ohad Trabelsi. The set cover conjecture and subgraph isomorphism with a tree pattern. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, volume 126 of *LIPIcs*, pages 45:1–45:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.45.

23 L. Lovász. Coverings and coloring of hypergraphs. In *4th Southeastern Conference on Combinatorics, Graph Theory, and Computing, Utilitas Math*, pages 3–12, 1973.

24 Daniël Paulusma. Open problems on graph coloring for special graph classes. In Ernst W. Mayr, editor, *Graph-Theoretic Concepts in Computer Science - 41st International Workshop, WG 2015, Garching, Germany, June 17-19, 2015, Revised Papers*, volume 9224 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2015.

25 B. Randerath and I. Schiermeyer. Vertex coloring and forbidden subgraphs - a survey. *Graphs Comb.*, 20:1–40, 2004.

26 J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, October 1980.

27 Z. Tuza. Graph colorings with local restrictions - a survey. *Discussiones Mathematicae Graph Theory*, 17:161–228, 1997.

28 Magnus Wahlström. Abusing the Tutte matrix: An algebraic instance compression for the K-set-cycle problem. In Natacha Portier and Thomas Wilke, editors, *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 - March 2, 2013, Kiel, Germany*, volume 20 of *LIPIcs*, pages 341–352. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

29 F. Yates. The design and analysis of factorial experiments. Technical Report Technical Communication no. 35, Commonwealth Bureau of Soils, 1937.

30 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposiumon on Symbolic and Algebraic Computation*, EUROSAM '79, pages 216–226, 1979.

Oracle Complexity Classes and Local Measurements on Physical Hamiltonians

Sevag Gharibian
Paderborn University, Paderborn Germany
sevag.gharibian@upb.de

Stephen Piddock
University of Bristol, Bristol, UK
stephen.piddock@bristol.ac.uk

Justin Yirka
The University of Texas at Austin, Austin, TX, USA
yirka@utexas.edu

Abstract

The canonical hard problems for NP and its quantum analogue, Quantum Merlin-Arthur (QMA), are MAX-k-SAT and the k-local Hamiltonian problem (k-LH), the quantum generalization of MAX-k-SAT, respectively. In recent years, however, an arguably even more physically motivated problem than k-LH has been formalized – the problem of simulating local measurements on ground states of local Hamiltonians (APX-SIM). Perhaps surprisingly, [Ambainis, CCC 2014] showed that APX-SIM is likely harder than QMA. Indeed, [Ambainis, CCC 2014] showed that APX-SIM is $\mathrm{P}^{\mathrm{QMA[log]}}$-complete, for $\mathrm{P}^{\mathrm{QMA[log]}}$ the class of languages decidable by a P machine making a logarithmic number of adaptive queries to a QMA oracle. In this work, we show that APX-SIM is $\mathrm{P}^{\mathrm{QMA[log]}}$-complete even when restricted to physically motivated Hamiltonians, obtaining as intermediate steps a variety of related complexity-theoretic results.

Specifically, we first give a sequence of results which together yield $\mathrm{P}^{\mathrm{QMA[log]}}$-hardness for APX-SIM on well-motivated Hamiltonians such as the 2D Heisenberg model:

- We show that for NP, StoqMA, and QMA oracles, a logarithmic number of adaptive queries is equivalent to polynomially many parallel queries. Formally, $\mathrm{P}^{\mathrm{NP[log]}} = \mathrm{P}^{||\mathrm{NP}}$, $\mathrm{P}^{\mathrm{StoqMA[log]}} = \mathrm{P}^{||\mathrm{StoqMA}}$, and $\mathrm{P}^{\mathrm{QMA[log]}} = \mathrm{P}^{||\mathrm{QMA}}$. (The result for NP was previously shown using a different proof technique.) These equalities simplify the proofs of our subsequent results.
- Next, we show that the hardness of APX-SIM is preserved under Hamiltonian simulations (à la [Cubitt, Montanaro, Piddock, 2017]) by studying a seemingly weaker problem, $\forall$-APX-SIM. As a byproduct, we obtain a full complexity classification of APX-SIM, showing it is complete for $\mathrm{P}, \mathrm{P}^{||\mathrm{NP}}, \mathrm{P}^{||\mathrm{StoqMA}}$, or $\mathrm{P}^{||\mathrm{QMA}}$ depending on the Hamiltonians employed.
- Leveraging the above, we show that APX-SIM is $\mathrm{P}^{\mathrm{QMA[log]}}$-complete for any family of Hamiltonians which can efficiently simulate spatially sparse Hamiltonians. This implies APX-SIM is $\mathrm{P}^{\mathrm{QMA[log]}}$-complete even on physically motivated models such as the 2D Heisenberg model.

Our second focus considers 1D systems: We show that APX-SIM remains $\mathrm{P}^{\mathrm{QMA[log]}}$-complete even for local Hamiltonians on a 1D line of 8-dimensional qudits. This uses a number of ideas from above, along with replacing the "query Hamiltonian" of [Ambainis, CCC 2014] with a new "sifter" construction.

2012 ACM Subject Classification Theory of computation → Quantum computation theory; Theory of computation → Oracles and decision trees; Theory of computation → Problems, reductions and completeness; Theory of computation → Quantum complexity theory

Keywords and phrases Quantum Merlin Arthur (QMA), simulation of local measurement, local Hamiltonian, oracle complexity class, physical Hamiltonians

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.20

Related Version A full version of the paper is available at `https://arxiv.org/abs/1909.05981`.

37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020).
Editors: Christophe Paul and Markus Bläser; Article No. 20; pp. 20:1–20:37

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

Funding *Sevag Gharibian*: SG acknowledges support from NSF grants CCF-1526189 and CCF-1617710.
Stephen Piddock: SP was supported by EPSRC.
Justin Yirka: Part of this work was completed while JY was supported by a Virginia Commonwealth University Presidential Scholarship. JY acknowledges QIP 2019 student travel funding (NSF CCF-1840547).

Acknowledgements We are grateful to Thomas Vidick for helpful discussions which helped initiate this work. We also thank an anonymous referee for [22] (written by two of the present authors) for the suggestion to think about 1D systems.

1 Introduction

In analogy with MAX-k-SAT playing a central role in the theory of NP-completeness, the *k-local Hamiltonian problem* (denoted k-LH, and which generalizes Boolean constraint satisfaction) is the canonical complete [31] problem for the quantum analogue of NP, Quantum Merlin Arthur (QMA). Roughly, in k-LH the input is a set of $2^k \times 2^k$ Hermitian matrices $\{ H_i \}$, where each H_i is a "local quantum constraint" acting on a subset of k out of n qubits. The output is the smallest eigenvalue of $H = \sum_i H_i$, known as the *ground state energy* of H, which we denote $\lambda(H)$. (For clarity, in the sum $H = \sum_i H_i$, each H_i is implicitly in tensor product with an identity on all qubits which H_i does not act on.) In words, the ground state energy encodes the energy of the quantum system corresponding to H when cooled into its lowest energy configuration. This remarkable connection between physics and complexity theory (i.e. Kitaev's proof that k-LH is QMA-complete [31]) spawned the field of Quantum Hamiltonian Complexity (QHC) (see, e.g., [37, 6, 18]), which has since explored the complexity of computing properties of ground spaces (i.e. "solution spaces" of k-LH instances) *beyond* estimating ground state energies [9, 44, 19, 21, 24, 20, 41, 3, 14, 22, 30, 7, 13].

Approximate Simulation

Despite the role of k-LH as a "posterchild" for Quantum Hamiltonian Complexity, in 2014 Ambainis formalized [3] the arguably even more natural physical problem of simulating local measurements on low-energy states of a local Hamiltonian, denoting it Approximate Simulation (APX-SIM). Intuitively, in APX-SIM one is given a local Hamiltonian H and local measurement A, and asked to estimate the expectation value of A against the ground space of H. Formally:

▶ **Definition 1** (APX-SIM$(H, A, k, \ell, a, b, \delta)$ [3]). *Given a k-local Hamiltonian H, an ℓ-local observable A, and real numbers a, b, and δ such that $b - a \geq n^{-c}$ and $\delta \geq n^{-c'}$, for n the number of qubits*[1] *H acts on and $c, c' > 0$ some constants, decide:*

- *If H has a ground state $|\psi\rangle$ satisfying $\langle\psi| A |\psi\rangle \leq a$, output YES.*
- *If for all $|\psi\rangle$ satisfying $\langle\psi| H |\psi\rangle \leq \lambda(H) + \delta$, it holds that $\langle\psi| A |\psi\rangle \geq b$, output NO.*

For clarity, any Hermitian matrix A is a valid observable representing some projective measurement; the eigenvalues of A denote the labels of the possible outcomes of the measurement, and the eigenvectors the corresponding quantum state onto which the system is projected post-measurement.

[1] We state Definition 1 using *qubits*, i.e. 2-dimensional local systems. One could also use higher dimensional qu*d*its, i.e. *d*-dimensional local systems, if desired. Indeed, in certain contexts, such as the containment result of Lemma 11, showing a result about qu*d*its is more general than just considering qubits (hence Lemma 11 explicitly uses qudits).

Motivation for APX-SIM. Given a naturally occurring quantum system with time evolution Hamiltonian H (which is typically k-local for $k \in O(1)$), we would like to learn something about the quantum state $|\psi\rangle$ the system settles into when cooled to near absolute zero. This low-temperature setting is particularly important, as it is where phenomena such as superconductivity and superfluidity manifest themselves. Thus, learning something about $|\psi\rangle$ potentially allows one to harness such phenomena for, say, materials design. The most "basic" experimental approach to learning something about $|\psi\rangle$ is to attempt to prepare a physical copy of $|\psi\rangle$, and then apply a local measurement to extract information from $|\psi\rangle$. However, given that preparing the ground state $|\psi\rangle$ of an arbitrary Hamiltonian is hard – it would allow one to solve the QMA-complete k-LH problem – we must wonder whether there is an easier approach. Formally, how hard is APX-SIM?

Perhaps surprisingly, it turns out that simulating a measurement on the ground state $|\psi\rangle$ is strictly *harder than* QMA. To show this, [3] proved that APX-SIM is $\mathrm{P}^{\mathrm{QMA[log]}}$-complete, for $\mathrm{P}^{\mathrm{QMA[log]}}$ the class of languages decidable by a P machine making a logarithmic number of adaptive queries to a QMA oracle. (See Section 2 and Appendix A for formal details on promise oracle classes P^{C}.) Why $\mathrm{P}^{\mathrm{QMA[log]}}$ instead of QMA? Intuitively, this is because APX-SIM does *not* include thresholds for the ground state energy as part of the input (in contrast to k-LH). This specification of APX-SIM is moreover well-motivated; typically one does not have an estimate of the ground state energy of H, since such an estimate is QMA-hard to compute to begin with.

Brief background on $\mathrm{P}^{\mathrm{QMA[log]}}$. The class $\mathrm{P}^{\mathrm{QMA[log]}}$ is likely strictly harder than QMA, since both QMA and co-QMA are contained in $\mathrm{P}^{\mathrm{QMA[log]}}$ (to put co-QMA in $\mathrm{P}^{\mathrm{QMA[log]}}$, use the QMA oracle once and flip its answer using the P machine). Thus, QMA $\neq \mathrm{P}^{\mathrm{QMA[log]}}$ unless co-QMA $\subseteq$ QMA (which appears unlikely). Just how much more difficult than QMA is $\mathrm{P}^{\mathrm{QMA[log]}}$? Intuitively, the answer is "slightly more difficult". Formally, QMA $\subseteq \mathrm{P}^{\mathrm{QMA[log]}} \subseteq$ PP [22] (where QMA $\subseteq \mathrm{A_0PP} \subseteq$ PP was known [32, 45, 34] prior to [22]; note the latter containment is strict unless the Polynomial-Time Hierarchy collapses [45]).

From a computer science perspective, there is an interesting relationship between APX-SIM and classical constraint satisfaction. Recall that k-LH is the QMA-complete generalization of MAX-k-SAT, in that the energy of a state is minimized by simultaneously satisfying as many k-local constraints as possible. Classically, one might be asked whether the solution to a MAX-k-SAT instance satisfies some easily verifiable property, such as whether the solution has even Hamming weight; such a problem is $\mathrm{P}^{\mathrm{NP[log]}}$-complete (see [46] for a survey). APX-SIM is a quantum analogue to these problems, in which we ask whether an optimal solution (the ground state) satisfies some property (expectation bounds for a specified measurement), and APX-SIM is analogously $\mathrm{P}^{\mathrm{QMA[log]}}$-complete. Beyond this connection, of course, the strong appeal of APX-SIM lies additionally in its physical motivation.

High level direction in this work. That APX-SIM is such a natural problem arguably demands that we study its hardness given natural settings. In this regard, the original $\mathrm{P}^{\mathrm{QMA[log]}}$-completeness result [3] was for simulating $O(\log n)$-local observables and $O(\log n)$-local Hamiltonians. From a physical perspective, one wishes to reduce the necessary locality, e.g. to $O(1)$-local observables and Hamiltonians. Hardness under this restriction was subsequently achieved [22], for 1-qubit observables and 5-local Hamiltonians, by combining the "query Hamiltonian" construction of Ambainis [3] with the circuit-to-Hamiltonian construction of Kitaev [31]. However, even arbitrary $O(1)$-local Hamiltonians may be rather artificial in contrast to naturally occurring systems. Ideally, one wishes to make statements

along the lines of "simulating measurements on a physical model such as the quantum Heisenberg model on a 2D lattice is harder than QMA", or "simulating measurements on a 1D local Hamiltonian is harder than QMA". This is what we achieve in the current paper. Interestingly, to attain this goal, we first take a turn into the world of parallel versus adaptive oracle queries.

Parallel versus adaptive queries

A natural question for oracle complexity classes is how the power of the class changes as access to the oracle is varied. In the early 1990's, it was shown [10, 27, 5] that a polynomial number of *parallel* or *non-adaptive* queries to an NP oracle are equivalent in power to a logarithmic number of *adaptive* queries. Formally, letting $\mathrm{P}^{||\mathrm{NP}}$ be the class of languages decidable by a P machine with access to polynomially many parallel queries to an NP oracle, it holds that $\mathrm{P}^{||\mathrm{NP}} = \mathrm{P}^{\mathrm{NP[log]}}$ [5].

We begin by considering the analogous question for $\mathrm{P}^{\mathrm{QMA[log]}}$ versus $\mathrm{P}^{||\mathrm{QMA}}$ (defined as $\mathrm{P}^{||\mathrm{NP}}$ but with a QMA oracle). For this, the direction $\mathrm{P}^{\mathrm{C[log]}} \subseteq \mathrm{P}^{||\mathrm{C}}$ was shown by [5] for all classes C. For the reverse, to show $\mathrm{P}^{||\mathrm{NP}} \subseteq \mathrm{P}^{\mathrm{NP[log]}}$, [5] first conducts binary search to determine the number of YES queries. Unfortunately, it is not clear how to carry out an analogous binary search in the setting of *promise problems*, such as for QMA. The problem is that, as explored in [22], oracles for promise classes like QMA may receive queries which violate their promise (such as an instance of k-LH with the ground state energy within the promise gap). By definition [23], in such cases the oracle can respond arbitrarily, even changing its answer given repeated queries. As a result, the number of YES queries, by which we mean queries on which the QMA oracle outputs 1, is not even well-defined. Thus, the technique of binary search fails for QMA. (Note that for a language version of QMA – i.e. the quantum analogue of MA as opposed to PromiseMA – this technique would still work.) To hence show $\mathrm{P}^{||\mathrm{QMA}} \subseteq \mathrm{P}^{\mathrm{QMA[log]}}$, we take a different approach: we show a *hardness* result. Specifically, we use a modification of the $\mathrm{P}^{\mathrm{QMA[log]}}$-hardness construction of [3], for which we require the locality improvements of [22], to show that APX-SIM is $\mathrm{P}^{||\mathrm{QMA}}$-hard. Combining with the fact that APX-SIM $\in \mathrm{P}^{\mathrm{QMA[log]}}$ [3] then yields the desired containment. This approach has two added benefits:

- First, the use of parallel, rather than adaptive, queries simplifies the "query Hamiltonian" construction of [3] significantly, which we later exploit to prove hardness results about physical Hamiltonians (Theorem 6) and 1D Hamiltonians (Theorem 10). Moreover, we give a simpler proof of Ambainis's original claim that APX-SIM is $\mathrm{P}^{\mathrm{QMA[log]}}$-hard; indeed, we generalize it to classes C beyond QMA to show:

  ▶ **Theorem 2.** *Let* C *be a class of languages or promise problems. Let $\mathcal{F}$ be a family of Hamiltonians for which k-LH is* C*-complete under poly-time many-one reductions for all $k \geq 2$. Suppose $\mathcal{F}$ is closed under positive linear combination of Hamiltonians, and that if $\{H_i\}_{i=1}^m \subset \mathcal{F}$, then $H_{cl} + \sum_{i=1}^m |1\rangle\langle 1|_i \otimes H_i \in \mathcal{F}$, where H_{cl} is any classical Hamiltonian (i.e. diagonal in the standard basis)*[2]*. Then, $\mathrm{P}^{\mathrm{C[log]}} = \mathrm{P}^{||\mathrm{C}}$, and APX-SIM is $\mathrm{P}^{\mathrm{C[log]}}$-complete when restricted to k-local Hamiltonians and observables from $\mathcal{F}$.*

  Recalling that k-LH is NP-complete, StoqMA-complete, and QMA-complete when restricted to the families of classical, stoquastic, and arbitrary k-local Hamiltonians, respectively [12], Theorem 2 thus gives the sequence of results:

[2] Briefly, the reason for the form of the expression $H_{\mathrm{cl}} + \sum_{i=1}^m |1\rangle\langle 1|_i \otimes H_i$ is that it suffices to encode our construction, while still belonging to several interesting families $\mathcal{F}$.

▶ **Corollary 3.** $\mathrm{P}^{\mathrm{NP}[\log]} = \mathrm{P}^{||\mathrm{NP}}, \mathrm{P}^{\mathrm{StoqMA}[\log]} = \mathrm{P}^{||\mathrm{StoqMA}}$, *and* $\mathrm{P}^{\mathrm{QMA}[\log]} = \mathrm{P}^{||\mathrm{QMA}}$.

- Second, we use the Cook-Levin theorem [11, 33], as opposed to Kitaev's circuit-to-Hamiltonian construction [31] as in [22]. This allows us to obtain a *constant* promise gap for the observable[3] A's threshold values (i.e. $b - a \geq \Omega(1)$, as opposed to $b - a \geq 1/\mathrm{poly}$), even when $\|A\| = O(1)$. Further, because the core of this construction is already spatially sparse, it eases proving hardness results about physical Hamiltonians (Theorem 6).

The complexity of APX-SIM for physically motivated Hamiltonians

With the simplifications that moving to parallel queries affords us (i.e. working with $\mathrm{P}^{||\mathrm{QMA}}$ versus $\mathrm{P}^{\mathrm{QMA}[\log]}$), we next study $\mathrm{P}^{||\mathrm{QMA}}$-hardness for physically motivated Hamiltonians. This requires a shift of focus to *simulations*, in the sense of [13], i.e. analog Hamiltonian simulations.

Recall that Kitaev originally proved QMA-hardness of k-LH for 5-local Hamiltonians [31]; this was brought down to 2-local Hamiltonians via perturbation theory techniques [29, 28]. Since then, there has been a large body of work (e.g. [36, 8, 12, 7, 39, 40]) showing complexity theoretic hardness results for ever simpler systems, much of which uses perturbative gadgets[4] to construct Hamiltonians which have approximately the same ground state energy as a Hamiltonian of an apparently more complicated form. Here, we wish to enable a similarly large number of results for the problem APX-SIM by using the same perturbative gadget constructions and ideas of analogue simulation.

To this end, Ref. [13] defines a strong notion of simulation which approximately preserves essentially all low-temperature properties of a Hamiltonian (including the ground state energy). It then observes that the perturbative gadget constructions used in the k-LH literature are examples of this definition of simulation. Ref. [13] then shows there exist simple families of Hamiltonians (such as the 2-qubit Heisenberg interaction) which are *universal* Hamiltonians, in the sense that they can simulate all $O(1)$-local Hamiltonians efficiently. Here "efficiently" means that the important parameters of the simulator Hamiltonian, are polynomially related to the corresponding parameters of the original Hamiltonian (see Definition 14).

How do simulations affect the complexity of APX-SIM? Ideally, we would like to show that efficient simulations similarly lead to reductions between classes of Hamiltonians for the problem APX-SIM. However, this is apparently difficult, as the definition of APX-SIM is not robust to small perturbations in the eigenvalues of the system. We instead consider a closely related, seemingly easier problem which we call ∀-APX-SIM.

▶ **Definition 4** (∀-APX-SIM$(H, A, k, \ell, a, b, \delta)$)**.** *Given a k-local Hamiltonian H, an ℓ-local observable A, and real numbers a, b, and δ such that satisfy $b - a \geq n^{-c}$ and $\delta \geq n^{-c'}$, for n the number of qubits H acts on and $c, c' > 0$ some constants, decide:*

- *If for all $|\psi\rangle$ s.t. $\langle\psi| H |\psi\rangle \leq \lambda(H) + \delta$, it holds that $\langle\psi| A |\psi\rangle \leq a$, then output YES.*
- *If for all $|\psi\rangle$ s.t. $\langle\psi| H |\psi\rangle \leq \lambda(H) + \delta$, it holds that $\langle\psi| A |\psi\rangle \geq b$, then output NO.*

[3] The constant gap is only for the input thresholds a, b for the expectation value of the observable A. The required "low-energy gap" defined by δ may be inverse polynomial, i.e. $\delta \geq 1/\mathrm{poly}$, and we note that the spectral gap of the Hamiltonian H may be arbitrarily small in our constructions unless otherwise noted. Thus, it is unclear how to apply this result to resolve questions concerning Hamiltonians with improved promise gaps, e.g. the Quantum PCP Conjecture.

[4] Very roughly, perturbative gadgets allow one to "craft" a set of desired low-lying eigenvalues/eigenspaces for a local Hamiltonian by carefully penalizing certain subspaces with non-constant weights.

Above, we have a stronger promise in the YES case than in APX-SIM: namely, *all* low-energy states $|\psi\rangle$ are promised to satisfy $\langle\psi| A |\psi\rangle \leq a$, as opposed to just a single ground state. Thus, ∀-APX-SIM is easier than APX-SIM, in that ∀-APX-SIM reduces to APX-SIM. (The reduction is trivial: a valid instance of ∀-APX-SIM is already a valid instance of APX-SIM.) We conclude that ∀-APX-SIM is contained in $\mathrm{P}^{\mathrm{QMA[log]}}$. Furthermore, the proof of Theorem 2 is actually sufficient to show that ∀-APX-SIM is $\mathrm{P}^{||\mathrm{C}}$-complete (when restricted to the corresponding family of Hamiltonians for arbitrary class C).

Our second result, Lemma 15, proves that efficient simulations correspond to reductions between instances of ∀-APX-SIM. As a byproduct, we combine this result with Theorem 2 and the universality classifications from [13] (cf. Corollary 3) to obtain complexity classifications for the original APX-SIM problem restricted to several families of Hamiltonians:

▶ **Theorem 5.** *Let $\mathcal{S}$ be an arbitrary fixed subset of Hermitian matrices on at most 2 qubits. Then the APX-SIM problem, restricted to Hamiltonians H and measurements A given as a linear combination of terms from $\mathcal{S}$, is*

1. P*-complete, if every matrix in $\mathcal{S}$ is 1-local;*
2. $\mathrm{P}^{\mathrm{NP[log]}}$*-complete, if $\mathcal{S}$ does not satisfy the previous condition and there exists $U \in SU(2)$ such that U diagonalizes all 1-qubit matrices in $\mathcal{S}$ and $U^{\otimes 2}$ diagonalizes all 2-qubit matrices in $\mathcal{S}$;*
3. $\mathrm{P}^{\mathrm{StoqMA[log]}}$*-complete, if $\mathcal{S}$ does not satisfy the previous condition and there exists $U \in SU(2)$ such that, for each 2-qubit matrix $H_i \in \mathcal{S}$, $U^{\otimes 2} H_i (U^\dagger)^{\otimes 2} = \alpha_i Z^{\otimes 2} + A_i I + I B_i$, where $\alpha_i \in \mathbb{R}$ and A_i, B_i are arbitrary single-qubit Hermitian matrices;*
4. $\mathrm{P}^{\mathrm{QMA[log]}}$*-complete, otherwise.*

Hardness of simulating local measurements on lattices and spatially sparse systems. With the previous two main results in hand, we are in a position to show that ∀-APX-SIM is $\mathrm{P}^{\mathrm{QMA[log]}}$-hard even when the Hamiltonian is restricted to a spatially sparse interaction graph (in the sense of [36]). This is analogous to the equivalent result for k-LH shown in [36], which was crucial in showing that the Local Hamiltonian problem is QMA-complete for Hamiltonians on a 2D square lattice. Formally, by exploiting the previously discussed results about parallel queries (Theorem 2) and simulations (Lemma 15) and by developing a variant of the hardness construction from Theorem 2, we are able to show the following:

▶ **Theorem 6.** *Let $\mathcal{F}$ be a family of Hamiltonians which can efficiently simulate any spatially sparse Hamiltonian. Then, APX-SIM is $\mathrm{P}^{\mathrm{QMA[log]}}$-complete even when restricted to a single-qubit observable and a Hamiltonian from the family $\mathcal{F}$.*

Via Theorem 6, we now obtain many corollaries via the long line of research using perturbative gadgets to prove QMA-completeness of restricted physical Hamiltonians; for brevity, here we list a select few such corollaries. We note that the locality of the observable input to APX-SIM may increase after simulation, but only by a constant factor which can be easily calculated based on the simulation used. For example, using the perturbative gadgets constructed in [39], the following is an immediate corollary of Theorem 6:

▶ **Corollary 7.** *APX-SIM is $\mathrm{P}^{\mathrm{QMA[log]}}$-complete even when the observable A is 4-local and the Hamiltonian H is restricted to be of the form $H = \sum_{(j,k)\in E} a_{(j,k)} h_{(j,k)}$ where $h_{(j,k)} = \alpha X_j X_k + \beta Y_j Y_k + \gamma Z_j Z_k$, E is the set of edges of a 2D square lattice, $a_{(j,k)} \in \mathbb{R}$, and at least two of α, β, γ are non-zero. The case $\alpha = \beta = \gamma$ corresponds to $XX + YY + ZZ$, which is the physically motivated Heisenberg interaction.*

But, there is not always a blow-up in the locality of A, as is shown by this corollary which follows from Theorem 6 and [42]:

▶ **Corollary 8.** *APX-SIM is* $\mathrm{P}^{\mathrm{QMA[log]}}$*-complete even when the observable A is 1-local and the Hamiltonian H is restricted to be of the form: $H = \sum_{(j,k)\in E} h_{(j,k)} + \sum_j B_j$, where $h_{(j,k)} = X_jX_k + Y_jY_k + Z_jZ_k$, E is the set of edges of a 2D square lattice, and B_j is a single qubit operator (that may depend on j).*

Finally, we remark that recent work on the simulation power of families of qudit Hamiltonians [40] can be used to show the following corollary:

▶ **Corollary 9.** *Let $|\psi\rangle$ be an entangled two qudit state. Then, APX-SIM is* $\mathrm{P}^{\mathrm{QMA[log]}}$*-complete even when the Hamiltonian H is restricted to be of the form $H = \sum_{j,k} \alpha_{j,k}|\psi\rangle\langle\psi|_{j,k}$, where $\alpha_{j,k} \in \mathbb{R}$ and $|\psi\rangle\langle\psi|_{j,k}$ denotes the projector onto $|\psi\rangle$ on qudits j and k.*

Each of these corollaries follows as the corresponding references show that the described families of Hamiltonians can efficiently simulate all spatially sparse Hamiltonians.

The complexity of APX-SIM on the line

We finally move to our last result, which characterizes the complexity of APX-SIM on the line. Historically, it was known that the NP-complete problem MAX-2-SAT on a line is efficiently solvable via dynamic programming or divide-and-conquer (even for large, but constant, dimension). It hence came as a surprise when [1] showed that 2-LH on a line is still QMA-complete. This result was for local dimension 13 ([1] claimed a result for 12-dimensional qudits; [26] identified and fixed an error in [1] by adding an extra dimension). [35] improved this to hardness for 12-dimensional qudits by leveraging the parity of the position of qudits. Most recently, [26] showed QMA-completeness for qudits of dimension 8 by allowing some of the clock transitions to be ambiguous. The complexity of k-LH on a 1D line remains open for local dimension $2 \leq d \leq 7$.

Returning to the setting of APX-SIM, the classical analogue of APX-SIM on a 1D line of bits is also in P; given a 2-local Boolean formula $\phi : \{0,1\}^n \mapsto \{0,1\}$, compute an optimal solution x to ϕ (which recall can be done in 1D as referenced above), and evaluate the desired efficiently computable local function on x (i.e. a "measurement" on a subset of the bits). This raises the question: is APX-SIM on a line still $\mathrm{P}^{\mathrm{QMA[log]}}$-complete? Or does its complexity in the 1D setting drop to, say, QMA? Our final result shows the former.

▶ **Theorem 10.** *APX-SIM is* $\mathrm{P}^{\mathrm{QMA[log]}}$*-complete even when restricted to Hamiltonians on a 1D line of 8-dimensional qudits and single-qudit observables.*

Thus, even in severely restricted geometries like the 1D line, simulating a measurement on a single qudit of the ground space remains harder than QMA.

Proof techniques for Theorem 10. We employ a combination of new and known ideas. We wish to simulate the idea from [22] that instead of having the P machine make m queries to a QMA oracle, it receives the answers to the queries as a "proof" $y \in \{0,1\}^m$ which it accesses whenever it needs a particular query answer. In [22], Ambainis's query Hamiltonian [3] was then used to ensure y was correctly initialized. However, it is not clear how to use Ambainis' query Hamiltonian (or variants of it) while maintaining a 1D layout.

We hence take a different approach. Instead of receiving the query *answers*, the P machine now has access to m QMA verifiers $\{V_i\}_{i=1}^m$ corresponding to the m queries, and for each of them receives a *quantum* proof $|\psi_i\rangle$ in some proof register R_i. The P machine then treats

the (probabilistic) outputs of each V_i as the "correct" answer to the query i. If a query i is a NO instance of a QMA problem, this works well – no proof can cause V_i to accept with high probability. However, if query i is a YES instance, a cheating prover may nevertheless submit a "bad" proof to verifier V_i, since flipping the output bit of V_i may cause the P machine to flip its final output bit. To prevent this, and thus ensure the P machine receives all correct answers with high probability, we use a delicate application of 1-local energy penalties, which we call "sifters", to the outputs of the V_i; just enough to penalize bad proofs for YES cases, but not enough to cause genuine NO cases to incur large energy penalties. Here, we again utilize our result that $\mathrm{P}^{\mathrm{QMA[log]}} = \mathrm{P}^{||\mathrm{QMA}}$ (Corollary 3), and choose to begin with a $\mathrm{P}^{||\mathrm{QMA}}$ instance; this allows us to apply identical, independent sifters to the output of each verifier V_i, significantly easing the subsequent analysis and transition to 1D.

We next plug this construction, where the P circuit has many sub-circuits V_i, into the 1D circuit-to-Hamiltonian construction of [26]. Similar to [22], we apply a corollary of the Projection Lemma of [28, 22] to argue that any low energy state must be close to a history state $|\psi\rangle$. Combining with our sifter Hamiltonian terms, we show in Lemma 23 that for $|\psi\rangle$ to remain in the low-energy space, it must encode V_i outputting approximately the right query answer for any query i. To then conclude that all query responses are jointly correct with high probability, and thus that the low-energy space encodes the correct final output to the $\mathrm{P}^{||\mathrm{QMA}}$ computation, we apply a known quantum non-commutative union bound. In fact, our argument immediately shows hardness for both APX-SIM and $\forall$-APX-SIM.

Open questions

Our results bring previous $\mathrm{P}^{\mathrm{QMA[log]}}$-hardness results for a remarkably natural problem, Approximate Simulation (APX-SIM), closer to the types of problems studied in the physics literature, where typically observables are $O(1)$-local, allowed interactions physically motivated, and the geometry of the interaction graph is constrained. There are many questions which remain open, of which we list a few here: (1) The coupling strengths for local Hamiltonian terms in Corollary 7,8,9 are typically non-constant, as these corollaries follow from the use of existing perturbation theory gadgets; can these coupling constants be made $O(1)$? Note this question is also open for the complexity classification of k-LH itself [12, 39]. (2) What is the complexity of $\mathrm{P}^{\mathrm{QMA[log]}}$? It is known that $\mathrm{P}^{\mathrm{QMA[log]}} \subseteq \mathrm{PP}$ [22]; can a tighter characterization be obtained? (3) Can similar hardness results for APX-SIM be shown for *translationally invariant* 1D systems? For reference, it is known that k-LH is $\mathrm{QMA}_{\mathrm{exp}}$-complete for 1D translationally invariant systems when the local dimension is roughly 40 [25, 4]. ($\mathrm{QMA}_{\mathrm{exp}}$ is roughly the quantum analogue of NEXP, in which the proof and verification circuit are exponentially large in the input size. The use of this class is necessary in [25, 4], as the only input parameter for 1D translationally invariant systems is the *length* of the chain.) If a similar hardness result holds for APX-SIM, presumably it would show $\mathrm{P}^{\mathrm{QMA}_{\mathrm{exp}}\mathrm{[log]}}$-hardness for 1D translationally invariant systems.

Organization. We prove Theorems 2, 5, 6, and 10 in Sections 3, 4, 5, and 6, respectively. Proofs omitted due to space constraints are deferred to Appendices B, D, C, and E.

2 Preliminaries

Notation. $\lambda(H)$ denotes the smallest eigenvalue of Hermitian operator H. For matrix A, define spectral norm $\|A\|_\infty := \max\{\|A|v\rangle\|_2 : \||v\rangle\|_2 = 1\}$ and trace norm $\|A\|_{\mathrm{tr}} := \mathrm{Tr}\sqrt{A^\dagger A}$. Throughout, we assume both $H = \sum_{i=1}^m H_i$ and observable $A = \sum_{i=1}^m A_i$ satisfy $m, \|H_i\|_\infty, \|A_i\|_\infty \in O(\mathrm{poly}(n))$, for n the number of qubits in the system.

Definitions. $\mathrm{P}^{\mathrm{QMA[log]}}$ [3] is the set of problems decidable by a polynomial-time deterministic Turing machine with the ability to query an oracle for a QMA-complete problem $O(\log n)$ times, where n is the size of the input. For a class C of languages or promise problems, the class $\mathrm{P}^{\mathrm{C[log]}}$ is similarly defined, except with an oracle for a C-complete problem. (See Appendix A for further formal details and discussion on promise oracle classes.) $\mathrm{P}^{||\mathrm{C}}$ is the set of problems decidable by a polynomial-time deterministic Turing machine given access to an oracle for a C-complete problem, with the restriction that all (up to $O(n^c)$ for $c \in \Theta(1)$) queries to the oracle be made in parallel. Such queries are hence *non-adaptive*, as opposed to the *adaptive* queries allowed to a $\mathrm{P}^{\mathrm{C[log]}}$ machine.

For $\mathrm{P}^{\mathrm{QMA[log]}}$ we assume the P machine makes queries to an oracle for the QMA-complete [31] k-local Hamiltonian problem (k-LH), defined as follows: Given a k-local Hamiltonian H and inverse polynomial-separated thresholds $a, b \in \mathbb{R}$, decide whether $\lambda(H) \leq a$ (YES-instance) or $\lambda(H) \geq b$ (NO-instance) [28]. We say an oracle query is *valid* (*invalid*) if it satisfies (violates) the promise gap of the QMA-complete problem the oracle answers. (An invalid query hence satisfies $\lambda(H) \in (a, b)$.) For any invalid query, the oracle can accept or reject arbitrarily. A *correct* query string $y \in \{0,1\}^m$ encodes a sequence of correct answers to all of the m queries made by the P machine, and an *incorrect* query string is one which contains at least one incorrect query answer. Note that for an invalid query, any answer is considered "correct", yielding the possible existence of multiple correct query strings. Nevertheless, the P machine is required to output the same final answer (accept or reject) regardless of how such invalid queries are answered [23].

3 Parallel versus adaptive queries

This section shows Theorem 2, i.e. that $\mathrm{P}^{\mathrm{C[log]}} = \mathrm{P}^{||\mathrm{C}}$ for appropriate complexity classes C, which will follow from Lemmas 11 and 12 below.

▶ **Lemma 11.** *Let H be a k-local Hamiltonian acting on n qudits, and let A be an observable on the same system of n qudits. If k-LH for $\alpha H + \beta A$ is contained in complexity class* C *for any $0 \leq \alpha, \beta \leq \mathrm{poly}(n)$ and for all $k \geq 1$, then APX-SIM$(H, A, k, \ell, a, b, \delta) \in \mathrm{P}^{\mathrm{C[log]}}$ for all $\ell \leq O(\log n)$ and $b - a, \delta \geq O(1/\mathrm{poly}\ n)$.*

The proof of Lemma 11 generalizes the known [3] proof that APX-SIM $\in \mathrm{P}^{\mathrm{QMA[log]}}$; the basic idea is to use binary search in conjunction with the oracle for C to estimate the ground state energy λ of H. One additional oracle query is then made to to determine whether H has a ground state with energy approximately λ and satisfying the promise thresholds for observable A. This last query, in particular, is where we must be careful in our generalization to arbitrary classes C. The formal proof is in Appendix B.

▶ **Lemma 12.** *Let $\mathcal{F}$ be a family of Hamiltonians for which k-LH is* C*-hard for all $k \geq 2$. Then $\forall$-APX-SIM is $\mathrm{P}^{||\mathrm{C}}$-hard even when $b - a = \Omega(1)$, the observable A is a single Pauli Z measurement, and when restricted to Hamiltonians of the form $H = H_{cl} + \sum_i |1\rangle\langle 1|_i \otimes H_i$, where H_{cl} is a classical Hamiltonian, and the H_i are Hamiltonians from $\mathcal{F}$.*

Before discussing the proof of Lemma 12, let us remark how Lemmas 11 and 12 combine to yield Theorem 2, i.e. that $\mathrm{P}^{\mathrm{C[log]}} = \mathrm{P}^{||\mathrm{C}}$. The interesting containment here is $\mathrm{P}^{||\mathrm{C}} \subseteq \mathrm{P}^{\mathrm{C[log]}}$. To show this, Lemma 12 yields that $\forall$-APX-SIM is $\mathrm{P}^{||\mathrm{C}}$-hard. But $\forall$-APX-SIM trivially reduces to APX-SIM, which Lemma 11 says is in $\mathrm{P}^{\mathrm{C[log]}}$. Hence, we have used a hardness result to show containment. The formal argument is in Appendix B.

We develop two tools needed to show Lemma 12: How to simplify [3]'s query Hamiltonian in the context of parallel queries (which is used to enforce correct query answers), and how to employ the Cook-Levin reduction (which enforces a correct simulation of the circuit given those query answers).

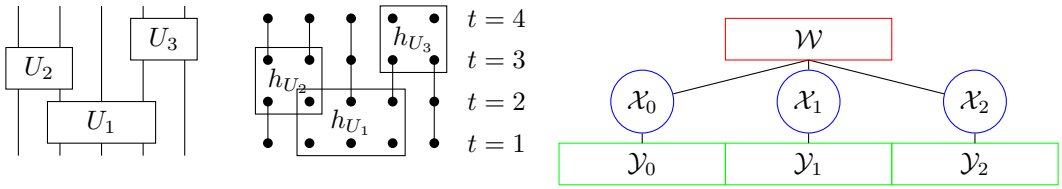


Figure 1 Left: Gates U_i in the circuit of the P machine. Middle: Hamiltonian terms h_{U_t} encoding each gate. Each straight line edge on the right represents the interaction $|01\rangle\langle 01| + |10\rangle\langle 10|$. The initialization terms H_{in} on qubits in time step $t = 0$ are omitted in the diagram. Right: The structure of the Hamiltonian $H = H_1 + H_2$ used in Lemma 12, for the case of 3 queries. H_1 acts on the space $\mathcal{W} \otimes \mathcal{X}$ and H_2 acts on $\mathcal{X} \otimes \mathcal{Y}$, where $\mathcal{X} = \bigotimes_i \mathcal{X}_i$ and $\mathcal{Y} = \bigotimes_i \mathcal{Y}_i$.

Tool 1: Simplifying Ambainis' query Hamiltonian. First, we give a simplified version of the "query Hamiltonian" of Ambainis [3], which will be useful in subsequent lemmas. Namely, given a $\mathrm{P}^{||\mathrm{C}}$ computation U for an appropriate class C, let $(H_{\mathcal{Y}_i}, a_i, b_i)$ be the instance of 2-LH corresponding to the i-th query made by U. Our "query Hamiltonian" is:

$$H = \sum_{i=1}^{m} M_i := \sum_{i=1}^{m} \left(\frac{a_i + b_i}{2} |0\rangle\langle 0|_{\mathcal{X}_i} \otimes I_{\mathcal{Y}_i} + |1\rangle\langle 1|_{\mathcal{X}_i} \otimes H_{\mathcal{Y}_i} \right), \tag{1}$$

where single qubit register $\mathcal{X}_i$ is intended to encode the answer to query i and $\mathcal{Y}_i$ encodes the ground state of $H_{\mathcal{Y}_i}$. Since each query is 2-local, H is 3-local. Notably, because U makes all of its queries in *parallel*, we are able to weight each of the m terms equally, unlike in [3, 22] which studied adaptive queries. This significantly eases our later analysis.

The key property of the query Hamiltonian H is given by the following lemma, which roughly says H encodes correct query answers in registers $\mathcal{X}_i$. This lemma is analogous to Lemma 3.1 of [22], but with an improved spectral gap; its proof (Appendix B) is similar to [22], but significantly simplified due to our use of parallel queries.

▶ **Lemma 13.** *Define for any $x \in \{0,1\}^m$ the space $\mathcal{H}_{x_1 \cdots x_m} := \bigotimes_{i=1}^m |x_i\rangle\langle x_i| \otimes \mathcal{Y}_i$. Then, there exists a correct query string $x \in \{0,1\}^m$ such that the ground state of H lies in $\mathcal{H}_{x_1 \cdots x_m}$. Moreover, if λ is the minimum eigenvalue of H restricted to this space, then for any incorrect query string $y_1 \cdots y_m$, any state in $\mathcal{H}_{y_1 \cdots y_m}$ has energy at least $\lambda + \epsilon$, where $\epsilon = \min_i(b_i - a_i)/2$.*

Tool 2: Adapting the Cook-Levin construction. We next model the Cook-Levin construction as a Hamiltonian for our setting. We may view the P machine as a circuit of classical reversible gates $U = U_m \ldots U_1$, in which time step i performs gate U_i. The evolution of the circuit is encoded into a 2D grid of qubits, where the t-th row of qubits corresponds to the state of the system at time step t; the output of the circuit is copied to a dedicated output bit in the final timestep. The overall Hamiltonian is diagonal in the computational basis with a groundspace of states corresponding to the correct evolution of the P machine.

Let I_t be the set of qubits which U_t acts non-trivially on. If a qubit $i \notin I_t$ (i.e. it is not acted on by the circuit at time step t), then there is an interaction $|01\rangle\langle 01| + |10\rangle\langle 10|$ on qubits (i,t) and $(i,t+1)$, to penalize states which encode a change on qubit i. To encode a classical reversible gate $U_t : x \mapsto U_t(x)$ acting at time t, we define an interaction $h_{U_t} = I - \sum_x |x\rangle\langle x|_t \otimes |U_t(x)\rangle\langle U_t(x)|_{t+1}$ acting non-trivially only on qubits (i,t') for $i \in I_t$ and t' equal to t or $t+1$. Figure 1 (middle) gives an illustration of this Hamiltonian. This

yields (positive semi-definite) propagation Hamiltonian

$$H_{\text{prop}} = \sum_{t=1}^{m} \left(h_{U_t} + \sum_{i \notin I_t} |0\rangle\langle 0|_{(i,t)} |1\rangle\langle 1|_{(i,t+1)} + |1\rangle\langle 1|_{(i,t)} |0\rangle\langle 0|_{(i,t+1)} \right), \tag{2}$$

whose ground space is spanned by states of the form: $|w(x)\rangle = |x\rangle_{t=1} \otimes |U_1 x\rangle_{t=2} \otimes \cdots \otimes |U_m \dots U_1 x\rangle_{t=m+1}$. We also add an H_{in} term consisting of 1-local $|1\rangle\langle 1|$ terms acting on the first ($t = 1$) row to ensure the start configuration is correct. One can show that the resulting Hamiltonian $H_{\text{prop}} + H_{\text{in}}$ has (1) unique ground state $|w(0^n)\rangle$ encoding the action of the circuit on the 0^n string, (2) ground state energy 0, and (3) spectral gap at least 1.

Proof sketch of Lemma 12. With our two tools in hand, we can finally sketch the proof of Lemma 12. Split the Hilbert space into three parts $\mathcal{W}$, $\mathcal{X} = \bigotimes_i \mathcal{X}_i$, $\mathcal{Y} = \bigotimes_i \mathcal{Y}_i$ and consider Hamiltonian $H = H_1 + H_2$, where H_1 acts on $\mathcal{W}$ and $\mathcal{X}$, and H_2 acts on $\mathcal{X}$ and $\mathcal{Y}$ (Figure 1). H_2 is the query Hamiltonian of Equation (1) (Tool 1); by Lemma 13, the low energy space of H_2 encodes in register in $\mathcal{X}$ a correct string of query answers for oracle C. $H_1 = H_{\text{prop}} + H_{\text{in}}$ is the classical Hamiltonian encoding the evolution of a classical P circuit, using the Cook-Levin construction (Tool 2), where H_{prop} is defined in Equation (2). Intuitively, the low energy space of H_1 simulates "reading" the correct query answers from $\mathcal{X}$ and using these to simulate the underlying P circuit in register $\mathcal{W}$. (Thus, $\mathcal{X}$ plays the role of a "message" register passing information between H_1 and H_2.) Details are in Appendix B.

4 Simulations and APX-SIM for physical classes of Hamiltonians

To study the complexity of APX-SIM for physically motivated Hamiltonians in Section 5, we require two tools: (1) hardness results for parallel query classes $\text{P}^{||\text{C}}$ (Section 3), and (2) an understanding of how *simulations* affect the hardness of the problem APX-SIM, which we now focus on. Roughly speaking, a simulation allows us to "reproduce" the low-energy physics of a desired physical model H, by instead using Hamiltonians H' of a different form. Formally, below we consider a simplified notion of simulation (a special case of the full definition given in [13]). This simpler case includes all of the important details necessary for the general case. Proofs with regard to the general definition of simulation are in Appendix F.

▶ **Definition 14** (Special case of definition in [13]; variant of definition in [7]). *We say that H' is a (Δ, η, ϵ)-simulation of H if there exists a local isometry $V = \bigotimes_i V_i$ such that*

1. *There exists an isometry $\widetilde{V}$ such that $\widetilde{V}\widetilde{V}^\dagger = P_{\leq \Delta(H')}$, where $P_{\leq \Delta(H')}$ is the projector onto the space of eigenvectors of H' with eigenvalues less than Δ, and $\|\widetilde{V} - V\| \leq \eta$;*
2. *$\|H'_{\leq \Delta} - \widetilde{V} H \widetilde{V}^\dagger\| \leq \epsilon$, where $H'_{\leq \Delta} = P_{\leq \Delta(H')} H' P_{\leq \Delta(H')}$.*

We say that a family $\mathcal{F}'$ of Hamiltonians can simulate a family $\mathcal{F}$ of Hamiltonians if, for any $H \in \mathcal{F}$ and any $\eta, \epsilon > 0$, and $\Delta \geq \Delta_0$ for some $\Delta_0 > 0$, there exists $H' \in \mathcal{F}'$ such that H' is a (Δ, η, ϵ)-simulation of H. We say that the simulation is efficient if, for H acting on n qudits, $\|H'\| = \text{poly}(n, 1/\eta, 1/\epsilon, \Delta)$; H' and $\{V_i\}$ are computable in polynomial-time given H, Δ, η and ϵ and provided that $\Delta, 1/\eta, 1/\epsilon$ are $O(\text{poly } n)$; and each isometry V_i maps from at most one qudit to $O(1)$ qudits.

Unlike in [13], here we have the additional requirement that the local isometry V is efficiently computable. This ensures that given some input Hamiltonian H and local observable A, we can use the notion of simulation to efficiently produce a simulating Hamiltonian H' *and* a simulating observable A'. As far as we are aware, all known constructions satisfying the notion of efficient simulation from [13] fulfill this additional requirement.

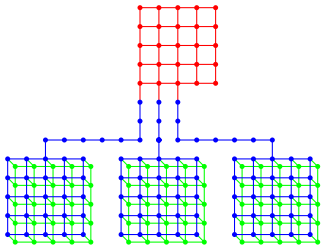


Figure 2 (Color figure) Geometric structure of Hamiltonian $H = H_1 + H_2 + H_3$ for the case of 3 queries. In words, H_1 is the top square, H_3 is the set of connecting wires and bottom three squares to which they are connected. H_2 is the remaining set of three squares at the bottom of the diagram.

One of the key properties of simulation is that it preserves eigenvalues of the target Hamiltonian up to a small additive factor ϵ. Unfortunately, even such small perturbations in eigenvalues can change the answer to APX-SIM if the Hamiltonian H has a small spectral gap. We hence instead work with the "more robust" problem $\forall$-APX-SIM, which recall trivially reduces to APX-SIM. Let $\mathcal{F}$-$\forall$-APXSIM denote the problem $\forall$-APX-SIM restricted to Hamiltonians taken from the family $\mathcal{F}$. The main result of this section is:

▶ **Lemma 15** (Simulations preserve hardness of $\forall$-APX-SIM)**.** *Let $\mathcal{F}$ be a family of Hamiltonians which can be efficiently simulated by another family $\mathcal{F}'$. Then, $\mathcal{F}$-$\forall$-APXSIM reduces to $\mathcal{F}'$-$\forall$-APXSIM via polynomial-time many-one reductions.*

The proof is rather technical, and hence deferred to Appendix C. As a corollary of Lemma 15, we also show Theorem 5, which recall gives a classification of the complexity of APX-SIM when restricted to families of Hamiltonians and measurements built up from a set of interactions $\mathcal{S}$: APX-SIM is either in P, $\mathrm{P}^{\mathrm{NP[log]}}$-complete, $\mathrm{P}^{\mathrm{StoqMA[log]}}$-complete, or $\mathrm{P}^{\mathrm{QMA[log]}}$-complete. Intuitively, this follows by combining the complexity characterizations and simulation results for k-LH of [12, 7, 13] with Lemma 15 and Theorem 2. However, some work is required to satisfy the preconditions of Theorem 2; details are in Appendix C.

5 Spatially sparse construction

We now combine the tools developed in the previous sections to study the complexity of APX-SIM for physical Hamiltonians. Our approach is to show that $\forall$-APX-SIM is $\mathrm{P}^{||\mathrm{QMA}}$-hard even for Hamiltonians on a *spatially sparse interaction graph*, defined below:

▶ **Definition 16** (Spatial sparsity [36])**.** *A spatially sparse interaction (hyper)graph G on n vertices is defined as a (hyper)graph in which*

1. *every vertex participates in $O(1)$ hyper-edges, and*
2. *there is a straight-line drawing in the plane such that every hyper-edge overlaps with $O(1)$ other hyper-edges and the surface covered by every hyper-edge is $O(1)$.*

▶ **Lemma 17.** *$\forall$-APX-SIM is $\mathrm{P}^{||\mathrm{QMA}}$-hard even for $b - a = \Omega(1)$, 1-local (single-qubit) observable A, and 4-local Hamiltonian H with a spatially sparse interaction graph.*

By combining Lemma 15, Lemma 17 and Corollary 3, we obtain Theorem 6, which recall shows APX-SIM is hard not only for families of Hamiltonians which are universal (i.e. families that can efficiently simulate any k-local Hamiltonian), but also for restricted families of Hamiltonians which can only efficiently simulate spatially sparse Hamiltonians. As stated in Section 1, this then yields the desired hardness results for APX-SIM on physical Hamiltonians such as the Heisenberg interaction on a 2D lattice (see, e.g., Corollary 7).

Proof sketch of Lemma 17. We adapt the proof of Lemma 12. Recall that the Hamiltonian H in Lemma 12 is $H = H_1 + H_2$, where H_2 uses a simplification of Ambainis's query Hamiltonian on each of the registers $\mathcal{X}_i \otimes \mathcal{Y}_i$ to encode the answer to that query into the state of $\mathcal{X}_i$, and H_1 encodes the evolution of the P circuit using the Cook-Levin construction on the $\mathcal{W}$ register (controlling on the states of the $\mathcal{X}_i$ registers). We arrange the qubits of the $\mathcal{W}$ register on a square lattice and note H_1 is already spatially sparse – this is an advantage of using the Cook-Levin construction over the Kitaev [31] history state construction. Furthermore, the Hamiltonian $H_{\mathcal{Y}_i}$, corresponding to the i-th QMA query, without loss of generality acts on a 2D square lattice [36], and hence is also spatially sparse.

The problem is that our version of Ambainis's query Hamiltonian H_2 is far from spatially sparse, since every qubit of $\mathcal{Y}_i$ interacts with $\mathcal{X}_i$. We resolve this by replacing each 1-qubit $\mathcal{X}_i$ register with a multi-qubit register of n_i qubits labeled $\{\mathcal{X}_i(j)\}_{j=1}^{n_i}$ (n_i the number of qubits of $\mathcal{Y}_i$). We spread out the qubits of the $\mathcal{X}_i$ register in space around the $\mathcal{Y}_i$ register, and modify H_2 so that each term is controlled only on a nearby qubit in $\mathcal{X}_i$. We also need to introduce a third term H_3 to ensure that all qubits in each $\mathcal{X}_i$ register are either all $|0\rangle$ or all $|1\rangle$. The construction is illustrated in Figure 2; the full proof is in Appendix D.

Finally, a brief comment about the claim that $b - a = \Omega(1)$ – as in Lemma 17, this is the promise gap for $\langle\psi| A |\psi\rangle$, and is $\Omega(1)$ due to our use of the Cook-Levin construction, which does not utilize history states (cf. [31]) to encode the action of the P machine.

6 Simulating measurements on a 1D line

We now show Theorem 10, i.e. that APX-SIM remains $\mathrm{P}^{\mathrm{QMA[log]}}$-complete even on a 1D line of 8-dimensional qudits with single-qudit observables. As the construction is rather involved, here we provide a sketch. Full details and a correctness proof are in Appendix E.

Sketch of 1D hardness construction. We give a reduction from $\mathrm{P}^{||\mathrm{QMA}}$ to $\forall$-APX-SIM, which by Theorem 2 yields the claim. Let Π be a $\mathrm{P}^{||\mathrm{QMA}}$ computation which takes in an input of size n and which consists of a uniformly generated polynomial-size classical circuit C making $m = O(\log n)$ 2-LH queries $\pi_i := (H_i, a_i, b_i)$ to a QMA oracle. As in Lemma 12, we treat the "answer register" in which C receives answers to its m queries as a proof register.

Our high-level approach consists of three steps: (1) construct a "master" circuit V composed of the verification circuits V_i corresponding to each query π_i and of the circuit C; (2) run V through the 1D circuit-to-Hamiltonian construction of [26] to obtain a 1D Hamiltonian G with local dimension 8, such that the low-energy space $\mathcal{S}$ of G consists of history states (of the form described in [26]); and (3) carefully add additional 1-local "sifter" penalty terms acting on the output qubits corresponding to each verification circuit V_i. Together, this yields a Hamiltonian H, whose low-energy space we show encodes satisfying proofs to each V_i (when possible). Specifically, the final step of "fine-grained splitting" of $\mathcal{S}$ (Step (3)) forces the output qubits of the circuits V_i to encode correct answers to query π_i, and thus the final circuit C receives a correct proof, hence leading the history states of step (2) to encode a correct simulation of Π. The answer to the computation Π can then be read off the ground state of H via an appropriate single qudit measurement. Note that Step (3) allows us to bypass the use of Ambainis' query Hamiltonian, which is a first for the study of $\mathrm{P}^{\mathrm{QMA[log]}}$.

1. Construction of V. Suppose each query π_i has corresponding QMA verification circuit V_i. We view the "master circuit" V as consisting of two phases:

1. (Verification phase) Given supposed proofs for each query, V runs all verification circuits V_i in parallel, where V_i acts on space $\mathcal{Y}_i \otimes \mathcal{W}_i \otimes \mathcal{X}_i$, for proof register $\mathcal{Y}_i$, ancilla register $\mathcal{W}_i$, and single-qubit output register $\mathcal{X}_i$.
2. (Simulated classical phase) The simulated P circuit C now receives the query answers $\mathcal{X} := \mathcal{X}_1 \otimes \cdots \otimes \mathcal{X}_m$ as its proof register as well as an ancilla register $\mathcal{W}_0$. It outputs a single qubit to an output register $\mathcal{X}_0$.

Crucially, note that given a set of proofs in register $\mathcal{Y} = \bigotimes_{i=1} \mathcal{Y}_i$, V does *not* necessarily yield the same answer as Π, since a malicious prover could intentionally send a "bad" proof to a YES query, flipping the final answer of V.

2. Construction of G. We now plug V into the circuit-to-Hamiltonian construction of Hallgren, Nagaj, and Narayanaswami [26] and make a small modification (see Appendix E) to obtain a nearest-neighbor 1D Hamiltonian G on 8-dimensional qudits.

3. Adding 1-local "sifters". We now add 1-local Hamiltonian terms which serve to "sift" through bad proofs, or more accurately split the ground space of G, so as to force low-energy states to encode correct query answers. Namely, even a correct simulation of circuit V may not output the correct answer for Π if a malicious prover gives a "bad" proof to query register $\mathcal{Y}_i$, even though π_i is a YES-query. We hence wish to penalize proofs $|\psi_i\rangle$ which lead verifier V_i to reject, whenever there exists a proof $|\phi_i\rangle$ V_i would have accepted (i.e. π_i is a YES query). To do so, we (roughly) add a "sifter" penalty term $\epsilon\,|0\rangle\langle 0|_{\mathcal{X}_i}$ to each answer register $\mathcal{X}_i$, for ϵ a carefully chosen inverse polynomial. In particular, ϵ must simultaneously (1) penalize NO answers enough to ensure the ground space encodes YES answers for YES-queries, but (2) be small enough that genuine NO query proofs are not accidentally rejected (i.e. when π_i is a NO-query). We collectively give the sifter terms the label H_{out}.

Final construction. The final Hamiltonian is $H := G + H_{\text{out}}$, with 1-qudit observable A penalizing output 0 on the designated output qudit of G. Proving correctness requires a number of additional lemmas (Appendix E); as a sample, two tools used are a corollary of the Projection Lemma of [28, 22] (low energy states must be close to a history state $|\psi\rangle$), and a (known) quantum union bound (all query answers are jointly correct with high probability).

References

1 D. Aharonov, D. Gottesman, S. Irani, and J. Kempe. The power of quantum systems on a line. *Communications in Mathematical Physics*, 287(1):41–65, 2009. doi:10.1007/s00220-008-0710-3.

2 D. Aharonov and T. Naveh. Quantum NP — A survey. Available at arXiv:quant-ph/0210077v1, 2002.

3 A. Ambainis. On physical problems that are slightly more difficult than QMA. In *Proceedings of 29th IEEE Conference on Computational Complexity (CCC)*, pages 32–43, 2014. doi:10.1109/ccc.2014.12.

4 J. Bausch, T. Cubitt, and M. Ozols. The complexity of translationally invariant spin chains with low local dimension. *Annales Henri Poincaré*, 18(11):3449–3513, 2017. doi:10.1007/s00023-017-0609-7.

5 R. Beigel. Bounded queries to SAT and the Boolean hierarchy. *Theoretical computer science*, 84(2):199–223, 1991. doi:10.1016/0304-3975(91)90160-4.

6 A. D. Bookatz. QMA-complete problems. *Quantum Information & Computation*, 14(5–6):361–383, 2014. doi:10.26421/QIC14.5-6.

7 S. Bravyi and M. Hastings. On complexity of the quantum Ising model. *Communications in Mathematical Physics*, 349(1):1–45, 2017. doi:10.1007/s00220-016-2787-4.

8 Sergey Bravyi, David P. DiVincenzo, and Daniel Loss. Schrieffer–Wolff transformation for quantum many-body systems. *Annals of Physics*, 326(10):2793–2826, 2011. doi:10.1016/j.aop.2011.06.004.

9 B. Brown, S. Flammia, and N. Schuch. Computational difficulty of computing the density of states. *Physical Review Letters*, 107(4):040501, 2011. doi:10.1103/physrevlett.107.040501.

10 S. Buss and L. Hay. On truth-table reducibility to SAT. *Information and Computation*, 91(1):86–102, 1991. doi:10.1016/0890-5401(91)90075-D.

11 S. Cook. The complexity of theorem proving procedures. In *Proceedings of the 3rd ACM Symposium on Theory of Computing (STOC)*, pages 151–158, 1972. doi:10.1145/800157.805047.

12 T. Cubitt and A. Montanaro. Complexity classification of local Hamiltonian problems. *SIAM J. Comput.*, 45(2):268–316, 2016. doi:10.1137/140998287.

13 T. Cubitt, A. Montanaro, and S. Piddock. Universal quantum Hamiltonians. *Proceedings of the National Academy of Sciences*, 115(38):9497–9502, 2018. doi:10.1073/pnas.1804949115.

14 T. Cubitt, D. Perez-Garcia, and M. M. Wolf. Undecidability of the spectral gap. *Nature*, 528:207–211, 2015. doi:10.1038/nature16059.

15 G. De las Cuevas and T. Cubitt. Simple universal models capture all classical spin physics. *Science*, 351(6278):1180–1183, 2016. doi:10.1126/science.aab3326.

16 C. A. Fuchs and J. van de Graaf. Cryptographic distinguishability measures for quantum-mechanical states. *IEEE Transactions on Information Theory*, 45(4):1216–1227, 1999. doi:10.1109/18.761271.

17 J. Gao. Quantum union bounds for sequential projective measurements. *Phys. Rev. A*, 92(5):052331, 2015. doi:10.1103/PhysRevA.92.052331.

18 S. Gharibian, Y. Huang, Z. Landau, and S. Woo Shin. Quantum Hamiltonian complexity. *Foundations and Trends in Theoretical Computer Science*, 10(3):159–282, 2015. doi:10.1561/0400000066.

19 S. Gharibian and J. Kempe. Hardness of approximation for quantum problems. In *Automata, Languages and Programming*, volume 7319 of *Lecture Notes in Computer Science*, pages 387–398, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-31594-7_33.

20 S. Gharibian, Z. Landau, S. W. Shin, and G. Wang. Tensor network non-zero testing. *Quantum Information & Computation*, 15(9–10):885–899, 2015. doi:10.26421/QIC15.9-10.

21 S. Gharibian and J. Sikora. Ground state connectivity of local Hamiltonians. *ACM Transactions on Computation Theory*, 10(2), 2018. doi:10.1145/3186587.

22 S. Gharibian and J. Yirka. The complexity of simulating local measurements on quantum systems. In M. M. Wilde, editor, *12th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2017)*, volume 73 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl. doi:10.4230/LIPIcs.TQC.2017.2.

23 O. Goldreich. On promise problems: A survey. In O. Goldreich, A.L. Rosenberg, and A.L. Selman, editors, *Theoretical Computer Science*, volume 3895 of *Lecture Notes in Computer Science*, pages 254–290. Springer, Berlin, Heidelberg, 2006. doi:10.1007/11685654_12.

24 D. Gosset, J. C. Mehta, and T. Vidick. QCMA hardness of ground space connectivity for commuting Hamiltonians. *Quantum*, 1:16, 2017. doi:10.22331/q-2017-07-14-16.

25 D. Gottesman and S. Irani. The quantum and classical complexity of translationally invariant tiling and Hamiltonian problems. *Theory of Computing*, 9(2):31–116, 2013. doi:10.4086/toc.2013.v009a002.

26 S. Hallgren, D. Nagaj, and S. Narayanaswami. The local Hamiltonian problem on a line with eight states is QMA-complete. *Quantum Information & Computation*, 13(9–10):721–750, 2013. doi:10.26421/QIC13.9-10.

27 L. Hemachandra. The strong exponential hierarchy collapses. *Journal of Computer and System Sciences*, 39(3):299–322, 1989. doi:10.1016/0022-0000(89)90025-1.

28 J. Kempe, A. Kitaev, and O. Regev. The complexity of the local Hamiltonian problem. *SIAM Journal on Computing*, 35(5):1070–1097, 2006. doi:10.1137/s0097539704445226.

29 J. Kempe and O. Regev. 3-local Hamiltonian is QMA-complete. *Quantum Information & Computation*, 3(3):258–264, 2003. doi:10.26421/QIC3.3.

30 I. H. Kim. Markovian matrix product density operators : Efficient computation of global entropy. Available at arXiv:1709.07828v2 [quant-ph], 2017.

31 A. Kitaev, A. Shen, and M. Vyalyi. *Classical and Quantum Computation*. American Mathematical Society, 2002.

32 A. Kitaev and J. Watrous. Parallelization, amplification, and exponential time simulation of quantum interactive proof systems. In *Proceedings of the 32nd ACM Symposium on Theory of Computing (STOC)*, pages 608–617, 2000. doi:10.1145/335305.335387.

33 L. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.

34 C. Marriott and J. Watrous. Quantum Arthur-Merlin games. *Computational Complexity*, 14(2):122–152, 2005. doi:10.1007/s00037-005-0194-x.

35 D. Nagaj. *Local Hamiltonians in quantum computation*. PhD thesis, Massachusetts Institute of Technology, 2008. Available at arXiv:0808.2117v1 [quant-ph].

36 R. Oliveira and B. M. Terhal. The complexity of quantum spin systems on a two-dimensional square lattice. *Quantum Information & Computation*, 8(10):900–924, 2008. doi:10.26421/QIC8.10.

37 T. J. Osborne. Hamiltonian complexity. *Reports on Progress in Physics*, 75(2):022001, 2012. doi:10.1088/0034-4885/75/2/022001.

38 S. K. Oskouei, S. Mancini, and M. M. Wilde. Union bound for quantum information processing. *Proceedings of the Royal Society A*, 475(2221), 2019. doi:10.1098/rspa.2018.0612.

39 S. Piddock and A. Montanaro. The complexity of antiferromagnetic interactions and 2D lattices. *Quantum Information & Computation*, 17(7&8):636–672, 2017. doi:10.26421/QIC17.7-8.

40 S. Piddock and A. Montanaro. Universal qudit Hamiltonians. Available at arXiv:1802.07130v1 [quant-ph], 2018.

41 G. Scarpa, A. Molnar, Y. Ge, J. J. Garcia-Ripoll, N. Schuch, D. Perez-Garcia, and S. Iblisdir. Computational complexity of PEPS zero testing. Available at arXiv1802.08214 [quant-ph], 2018.

42 N. Schuch and F. Verstraete. Computational complexity of interacting electrons and fundamental limitations of Density Functional Theory. *Nature Physics*, 5:732–735, 2009. doi:10.1038/nphys1370.

43 P. Sen. Achieving the Han-Kobayashi inner bound for the quantum interference channel. In *Proceedings of the 2012 IEEE International Symposium on Information Theory*, pages 736–740, 2012. doi:10.1109/ISIT.2012.6284656.

44 Y. Shi and S. Zhang. Note on quantum counting classes. Available at http:://www.cse.cuhk.edu.hk/syzhang/papers/SharpBQP.pdf.

45 M. Vyalyi. QMA=PP implies that PP contains PH. Available at Electronic Colloquium on Computational Complexity (ECCC), 2003.

46 K.W. Wagner. Bounded query computations. In *Proceedings of Structure in Complexity Theory Third Annual Conference*, pages 260–277, 1988. doi:10.1109/SCT.1988.5286.

A Additional notation and definitions

Notation. For a subspace S, $S^{\perp}$ denotes the orthogonal complement of S. We denote the restriction of an operator H to subspace S as $H|_S$. The null space of H is denoted $\mathrm{Null}(H)$.

Definitions. We define the notion of containment of a promise problem $\Pi = (\Pi_{\mathrm{yes}}, \Pi_{\mathrm{no}})$ in complexity class P^{C} for promise class C following Definition 1.3 of Goldreich [23].

▶ **Definition 18** (Cook reduction among promise problems [23]). *A promise problem $\Pi = (\Pi_{\mathrm{yes}}, \Pi_{\mathrm{no}})$ is Cook-reducible to promise problem $\Pi' = (\Pi'_{\mathrm{yes}}, \Pi'_{\mathrm{no}})$ if there exists a polynomial-time oracle Turing machine M such that:*

- *For every $x \in \Pi_{\mathrm{yes}}$, $M^{\Pi'}(x) = 1$,*
- *for every $x \in \Pi_{\mathrm{no}}$, $M^{\Pi'}(x) = 0$,*

and any query q to Π' is answered by 1 if $q \in \Pi'_{\mathrm{yes}}$, by 0 if $q \in \Pi'_{\mathrm{no}}$, and arbitrarily otherwise.

Remarks: (1) Quantum complexity classes, such as QMA, are typically promise classes, despite the lack of the prefix "Promise" in their name. (In contrast, in the classical complexity theory community, one typically distinguishes between, say, MA and PromiseMA.) Thus, as stated in Section 2, a QMA oracle can (in line with Definition 18) answer an invalid QMA query (i.e. violating the promise of the oracle) arbitrarily. Regardless of how such invalid queries are answered, the P machine must output the same final answer [23]. (2) If according to Definition 18, Π Cook-reduces to Π' for Π' in promise class C, then we say $\Pi \in \mathrm{P}^{\mathrm{C}}$. However, it is crucial to note that this does *not necessarily* imply that $\Pi \in C$. A notorious example [23] of this is the promise problem xSAT, for which Π_{yes} is the set of two-tuples (ϕ_1, ϕ_2) where ϕ_1 and ϕ_2 are satisfiable and unsatisfiable Boolean formulae, respectively, and Π_{no} is analogous but with ϕ_2 and ϕ_1 being satisfiable and unsatisfiable, respectively. Then, one can show (see Theorem 5.1 of [23]) that NP is Cook-reducible to xSAT, and that xSAT is in NP $\cap$ co-NP, and yet this does not necessarily imply that NP $\in$ NP $\cap$ co-NP.

B Proofs for Section 3

Proof of Lemma 11. We need to show the existence of a poly(n) time classical algorithm to decide APX-SIM while making at most $O(\log n)$ queries to an oracle for C. As with the proof in [3], the idea is to use $O(\log n)$ oracle queries to determine the ground space energy $\lambda(H)$ of H by binary search, and then use one final query to determine the answer. In [3] the final query is a QMA query; here we show how this final query can be performed differently so that only an oracle for C is required.

First calculate a lower bound μ for $\lambda(A)$, the lowest eigenvalue of A. If A acts only on $O(1)$ qudits, then $\lambda(A)$ can be calculated via brute force (up to, say, inverse exponential additive error) in $O(1)$ time. If A acts on many qudits, then $\lambda(A)$ can alternatively be approximated to within inverse polynomial additive error by binary search (as in [3]) by querying the C oracle $O(\log \|A\|) = O(\log n)$ times. Note that without loss of generality, we may assume $0 \leq b - \mu \leq q(n)$ for some efficiently computable polynomial q. The lower bound holds since if $b < \mu \leq \lambda(A)$, we conclude our APX-SIM instance is a NO instance, and we reject. For the upper bound, it holds that $\mu \leq \|A\|_{\infty}$, and we may assume $b \leq \|A\|_{\infty}$, as otherwise our APX-SIM instance is either a YES or invalid instance, and in both cases we can accept. By assumption, $\|A\|_{\infty} \leq q(n)$ for appropriate polynomial q which can be computed efficiently by applying the triangle inequality to the local terms of A; note $\|A\|_{\infty}$ may hence be replaced by q in the bounds above.

Perform binary search with the oracle for C (an example of how to perform binary search with an oracle for a promise problem is given in [3]) to find λ^* such that $\lambda(H) \in [\lambda^*, \lambda^* + \epsilon]$ where

$$\epsilon = \frac{\delta(b-a)}{2(b-\mu)} \geq 1/\mathrm{poly}(n)$$

since $0 \leq b - \mu \leq \mathrm{poly}(n)$. This requires $O(\log 1/\epsilon) = O(\log n)$ queries to the oracle for C. Next perform one final query to the C oracle to solve k-LH with Hamiltonian H' with thresholds a' and b', where

$$H' = (b-\mu)H + \delta A \quad \text{and} \quad \begin{array}{l} a' = (\lambda^* + \epsilon)(b-\mu) + \delta a \\ b' = \lambda^*(b-\mu) + \delta b \end{array}$$

and accept if and only if this final query accepts. Observe this is an allowed query for the C oracle because H' is of the form required in the statement of the lemma (recall $b - \mu \geq 0$), and also

$$b' - a' = \delta(b-a) - \epsilon(b-\mu) = \delta(b-a)/2 \geq 1/\mathrm{poly}(n).$$

Now, if APX-SIM$(H, A, k, l, a, b, \delta)$ is a YES instance, then there exists $|\psi\rangle$ such that $\langle\psi| H |\psi\rangle = \lambda(H)$ and $\langle\psi| A |\psi\rangle \leq a$. Then

$$\langle\psi| (b-\mu)H + \delta A |\psi\rangle \leq \lambda(H)(b-\mu) + \delta a \leq (\lambda^* + \epsilon)(b-\mu) + \delta a = a'$$

and the algorithm accepts as required.

Now suppose the input is a NO instance. We will show that $\langle\psi| H' |\psi\rangle \geq b'$ for any $|\psi\rangle$ and so the algorithm rejects as required. First, if $|\psi\rangle$ is low-energy with $\langle\psi| H |\psi\rangle \leq \lambda(H) + \delta$, then it also satisfies $\langle\psi| A |\psi\rangle \geq b$, and so

$$\langle\psi| (b-\mu)H + \delta A |\psi\rangle \geq \lambda(H)(b-\mu) + \delta b \geq \lambda^*(b-\mu) + \delta b = b'$$

where we have used $\langle\psi| H |\psi\rangle \geq \lambda(H) \geq \lambda^*$ and $b - \mu \geq 0$. Otherwise, if $|\psi\rangle$ is high energy with $\langle\psi| H |\psi\rangle \geq \lambda(H) + \delta$, then

$$\begin{aligned} \langle\psi| (b-\mu)H + \delta A |\psi\rangle &\geq (\lambda(H) + \delta)(b-\mu) + \delta\lambda(A) \\ &= \lambda(H)(b-\mu) + \delta b + \delta(\lambda(A) - \mu) \geq \lambda^*(b-\mu) + \delta b = b' \end{aligned}$$

where we have used $\langle\psi| A |\psi\rangle \geq \lambda(A)$ and $\lambda(A) - \mu \geq 0$. Thus, we reject. ◀

Proof of Lemma 13. We proceed by contradiction. Let $x \in \{0,1\}^m$ ($y \in \{0,1\}^m$) denote a correct (incorrect) query string which has lowest energy among all *correct* (*incorrect*) query strings against H. (Note that x and y are well-defined, though they may not be unique; in this latter case, any such x and y will suffice for our proof.) For any $z \in \{0,1\}^m$, define λ_z as the smallest eigenvalue in $\mathcal{H}_z$.

Since y is an incorrect query string, there exists at least one $i \in \{1, \ldots, m\}$ such that y_i is the wrong answer to a valid query $H_{\mathcal{Y}_i}$. If query i is a YES-instance, the smallest eigenvalue of M_i corresponds to setting $\mathcal{X}_i$ to (the correct query answer) $|1\rangle$, and is at most a_i. On the other hand, the space with $\mathcal{X}_i$ set to $|0\rangle$ has all eigenvalues equaling $(a_i + b_i)/2$. A similar argument shows that in the NO-case, the $|0\rangle$-space has eigenvalues equaling $(a_i + b_i)/2$, and the $|1\rangle$-space has eigenvalues at least b_i. We conclude that flipping query bit i to the correct query answer $\overline{y}_i$ allows us to "save" an energy penalty of $(b_i - a_i)/2$ against M_i, and since all other terms act invariantly on $\mathcal{X}_i \otimes \mathcal{Y}_i$, we save $(b_i - a_i)/2$ against H as well.

Let y' denote y with bit i flipped. If y' is also an incorrect query string, we have $\lambda_{y'} < \lambda_y$, a contradiction due to the minimality of y. Conversely, if y' is a correct query string, then we must have $\lambda_{y'} \geq \lambda_x + (b_i - a_i)/2 \geq \lambda + \epsilon$, as otherwise we contradict the minimality of x. ◀

The following technical lemma will be used in the proofs of Lemmas 12 and 15.

▶ **Lemma 19.** *Let H be a Hamiltonian and ρ a density matrix satisfying* $\mathrm{Tr}(H\rho) \leq \lambda(H)+\delta$. *Let P be the projector onto the space of eigenvectors of H with energy less than $\lambda(H)+\delta'$. Then,*

$$\frac{1}{2}\|\rho-\rho'\|_1 \leq \sqrt{\frac{\delta}{\delta'}}, \qquad \text{where } \rho' = P\rho P/\,\mathrm{Tr}(P\rho).$$

Proof of Lemma 19. First, bound the trace distance by the fidelity in the usual way (using one of the Fuchs-van de Graf inequalities [16]):

$$\frac{1}{2}\|\rho-\rho'\|_1 \leq \sqrt{1-F(\rho,\rho')^2} \tag{3}$$

where

$$F(\rho,\rho') = \mathrm{Tr}\left(\sqrt{\sqrt{\rho}\rho'\sqrt{\rho}}\right) = \mathrm{Tr}\left(\sqrt{\frac{\sqrt{\rho}P\rho P\sqrt{\rho}}{\mathrm{Tr}(P\rho)}}\right) = \frac{1}{\sqrt{\mathrm{Tr}(P\rho)}}\,\mathrm{Tr}(\sqrt{\rho}P\sqrt{\rho}) = \sqrt{\mathrm{Tr}(P\rho)},$$

where the third equality follows since $(\sqrt{\rho}P\sqrt{\rho})^2 = \sqrt{\rho}P\rho P\sqrt{\rho}$ and since the latter is positive semi-definite. Now, it remains to bound $\mathrm{Tr}(P\rho)$. We note that H has eigenvalues at least $\lambda(H)+\delta'$ on the space annihilated by P and eigenvalues at least $\lambda(H)$ everywhere else, and so $H \succeq (\lambda(H)+\delta')(I-P)+\lambda(H)P = (\lambda(H)+\delta')I-\delta' P$. Therefore, using the bound on $\mathrm{Tr}(H\rho)$, we have

$$\lambda(H)+\delta \geq \mathrm{Tr}(H\rho) \geq (\lambda(H)+\delta')\,\mathrm{Tr}(\rho) - \delta'\,\mathrm{Tr}(P\rho) \quad \Leftrightarrow \quad 1-\mathrm{Tr}(P\rho) \leq \frac{\delta}{\delta'}.$$

Substituting this back into Equation (3) proves the result. ◀

Proof of Lemma 12. We split the Hilbert space into three parts $\mathcal{W}$, $\mathcal{X} = \bigotimes_i \mathcal{X}_i$, $\mathcal{Y} = \bigotimes_i \mathcal{Y}_i$ and have a Hamiltonian of the form $H = H_1 + H_2$, where H_1 acts on $\mathcal{W}$ and $\mathcal{X}$, and H_2 acts on $\mathcal{X}$ and $\mathcal{Y}$. H_2 is the query Hamiltonian of Equation (1), and therefore by Lemma 13 the space of eigenvectors of H_2 with eigenvalues less than $\lambda(H_2)+\epsilon$ is spanned by states of the form: $|x\rangle_{\mathcal{X}} \otimes |\phi\rangle_{\mathcal{Y}}$, where x is a correct string of answers for the queries to the C oracle.

$H_1 = H_{\mathrm{prop}} + H_{\mathrm{in}}$ is the classical Hamiltonian encoding the evolution of a classical P circuit, using the Cook-Levin construction of Section 3, where H_{prop} is as defined in Equation (2). For clarity, H_{prop} and H_{in} act on $\mathcal{W}$ and $\mathcal{W}\otimes\mathcal{X}$, respectively. We think of $\mathcal{W}$ as "laid out in a 2D grid" as in Figure 1, and of $\mathcal{X}$ as playing the role of a "message" register passing information between H_1 and H_2. We modify the Hamiltonian H_{in} which initializes the qubits at the start of the classical circuit. For each qubit $\mathcal{X}_i$ in $\mathcal{X}$, we initialize a corresponding qubit of the first ($t=0$) row of $\mathcal{W}$ into the same state with a penalty term $|1\rangle\langle 1|_{\mathcal{X}_i}\otimes|0\rangle\langle 0|_{\mathcal{W}_i} + |0\rangle\langle 0|_{\mathcal{X}_i}\otimes|1\rangle\langle 1|_{\mathcal{W}_i}$. All other qubits in the first ($t=0$) row of $\mathcal{W}$ are initialized to $|0\rangle$ with a penalty $|1\rangle\langle 1|$. The full construction is depicted diagrammatically in Figure 1. Note that as stated in the claim, H is of the form $H = H_{\mathrm{cl}} + \sum_i^m |1\rangle\langle 1|_i \otimes H_i$, where H_{cl} contains H_1 and the local terms of H_2 which are tagged with $|0\rangle\langle 0|$ in registers $\mathcal{X}_i$.

We can argue about the low-energy eigenspace of H as follows. Since the ground spaces of H_1 and H_2 have non-trivial intersection, $\lambda(H) = \lambda(H_1)+\lambda(H_2) = \lambda(H_2)$. Moreover, since $[H_1,H_2]=0$ (they overlap only on the $\mathcal{X}$ register, on which they are both diagonal in the standard basis), and since we may assume without loss of generality that $\lambda(H_2)+\epsilon$ is inverse polynomially bounded below 1 (otherwise, we can scale H_1 by an appropriate fixed polynomial), we conclude the space of eigenstates of H with eigenvalue less than $\lambda(H)+\epsilon$,

henceforth denoted $\mathcal{H}_{\text{low}}$, is spanned by states of the form $|\Phi\rangle = |w\rangle_{\mathcal{W}} \otimes |x\rangle_{\mathcal{X}} \otimes |\phi\rangle_{\mathcal{Y}}$, where x is a string of correct answers to the oracle queries and w is the classical string encoding the correct computation of the P circuit acting on x. The qubit corresponding to the output bit of the P circuit will be in the state $|1\rangle$ (resp. $|0\rangle$) in a YES (resp. NO) instance of $\forall$-APX-SIM.

To complete the proof let the observable $A = Z_{\text{out}}$, a Pauli Z measurement on the qubit corresponding to the output bit of the P circuit, and let $\delta = \epsilon/16$ and $\delta' = \epsilon$. Consider any state $|\psi\rangle$ with $\langle\psi| H |\psi\rangle \leq \lambda(H) + \delta$. Then by Lemma 19, there exists a state $|\psi'\rangle \in \mathcal{H}_{\text{low}}$ such that $\langle\psi'| H |\psi'\rangle \leq \lambda(H) + \delta' = \lambda(H) + \epsilon$ which satisfies $\| |\psi\rangle\langle\psi| - |\psi'\rangle\langle\psi'| \|_1 \leq 1/2$. So,

$$\langle\psi'| Z_{\text{out}} |\psi'\rangle = \begin{cases} -1 & \text{in a YES instance} \\ 1 & \text{in a NO instance} \end{cases}$$

which implies by Hölder's inequality that $\langle\psi| A |\psi\rangle$ is $\leq -1/2$ in a YES instance and $\geq 1/2$ in a NO instance, as required. ◀

Proof of Theorem 2. The containment $\mathrm{P}^{\mathrm{C[log]}} \subseteq \mathrm{P}^{||\mathrm{C}}$ follows directly from the same argument that $\mathrm{P}^{\mathrm{NP[log]}} \subseteq \mathrm{P}^{||\mathrm{NP}}$ of [5], which we summarized in Section 1. By Lemma 11, APX-SIM is contained in $\mathrm{P}^{\mathrm{C[log]}}$ for Hamiltonians and observables from $\mathcal{F}$. And by Lemma 12 $\forall$-APX-SIM is $\mathrm{P}^{||\mathrm{C}}$-hard for Hamiltonians from $\mathcal{F}$, even when the observable is a single Pauli Z measurement, which is contained in $\mathcal{F}$ by the assumption that $\mathcal{F}$ contains any classical Hamiltonian H_{cl}. Since $\forall$-APX-SIM trivially reduces to APX-SIM, we thus have that APX-SIM is similarly $\mathrm{P}^{||\mathrm{C}}$-hard, and the result follows. ◀

C Proofs for Section 4

Proof of Lemma 15. Let $\Pi = (H, A, k, \ell, a, b, \delta)$ be an instance of $\mathcal{F}$-$\forall$-APXSIM. We will demonstrate that one can efficiently compute $H' \in \mathcal{F}'$ and A', k', ℓ', a', b', and δ' such that $\Pi' = (H', A', k', \ell', a', b', \delta')$ is a YES (respectively NO) instance of $\forall$-APX-SIM if Π is a YES (resp. NO) instance of $\forall$-APX-SIM; further, we will have that $\ell' \in O(\ell), a' = a + (b-a)/3, b' = b - (b-a)/3$ and $\delta - \delta' \geq 1/\text{poly}(n)$. To do so, we shall pick parameters Δ, η, ϵ so that $\Delta, 1/\eta, 1/\epsilon$ are $O(\text{poly } n)$, upon which the definition of efficient simulation (Definition 14) guarantees we can efficiently compute a Hamiltonian H' being a (Δ, η, ϵ)-simulation of H, which we claim will preserve YES and NO instances H.

Let us leave Δ, η, ϵ arbitrary for now, and assume we have a simulation of the form given in Definition 14. Then, there exists an isometry $\widetilde{V} : \mathcal{H} \to \mathcal{H}'$ ($\mathcal{H}$ and $\mathcal{H}'$ are the spaces H and H' act on, respectively) which maps onto the space of eigenvectors of H' with eigenvalues less than Δ, i.e. onto $S_{\leq\Delta} := \text{Span}\{|\psi\rangle : H' |\psi\rangle = \lambda |\psi\rangle, \lambda \leq \Delta\}$. In addition, $\widetilde{V}$ satisfies $\|\widetilde{V} - \bigotimes_i V_i\| \leqslant \eta$ and $\|H_{\leqslant\Delta} - \widetilde{V} H \widetilde{V}^\dagger\| \leqslant \epsilon$.

Let $|\psi'\rangle$ be a low-energy state of H' satisfying $\langle\psi'| H' |\psi'\rangle \leq \lambda(H') + \delta'$ for δ' to be set later. First, we show that $|\psi'\rangle$ is close to a state $\widetilde{V} |\psi\rangle$ where $|\psi\rangle$ is a low-energy state of H; then, we will show that there exists an observable A', depending only on A and the isometries V_i, such that $\langle\psi'| A' |\psi'\rangle$ approximates $\langle\psi| A |\psi\rangle$ for any choice of $|\psi\rangle$. Since by Definition 14 A is efficiently computable, our choice of A' will be as well.

Let $|\phi\rangle = P_{\leq\Delta(H')} |\psi'\rangle / \|P_{\leq\Delta(H')} |\psi'\rangle\|$ be the (normalized) component of $|\psi'\rangle$ in $S_{\leq\Delta}$. By Lemma 19, we have

$$\frac{1}{2} \| |\psi'\rangle\langle\psi'| - |\phi\rangle\langle\phi| \|_1 \leq \sqrt{\frac{\delta'}{\Delta - \lambda(H')}}.$$

Since $S_{\leq\Delta} = \mathrm{Im}(\widetilde{V})$, there must exist a state $|\psi\rangle$ in $\mathcal{H}$ such that $\widetilde{V}|\psi\rangle = |\phi\rangle$; next, we will show that $|\psi\rangle$ has low-energy with respect to H. Note that $|\psi'\rangle = \sqrt{p}|\phi\rangle + \sqrt{1-p}|\phi^\perp\rangle$ for some $p \in [0,1]$ and a state $|\phi^\perp\rangle$ in $S^\perp_{\leq\Delta}$ which has higher energy: $\langle\phi^\perp|H'|\phi^\perp\rangle \geq \Delta \geq \langle\phi|H'|\phi\rangle$. Therefore,

$$\langle\psi'|H'|\psi'\rangle = p\langle\phi|H'|\phi\rangle + (1-p)\langle\phi^\perp|H'|\phi^\perp\rangle \geqslant \langle\phi|H'|\phi\rangle,$$

which implies that

$$\begin{aligned}
\langle\psi|H|\psi\rangle - \langle\psi'|H'|\psi'\rangle &\leqslant \langle\psi|H|\psi\rangle - \langle\phi|H'|\phi\rangle && (4)\\
&= \langle\phi|\widetilde{V}H\widetilde{V}^\dagger|\phi\rangle - \langle\phi|H'|\phi\rangle && (5)\\
&\leqslant \|H'_{\leq\Delta} - \widetilde{V}H\widetilde{V}^\dagger\| \leqslant \epsilon. && (6)
\end{aligned}$$

So, $\langle\psi|H|\psi\rangle \leqslant \lambda(H') + \delta' + \epsilon \leqslant \lambda(H) + \delta' + 2\epsilon$, where the final inequality follows from Lemma 27 of [13], which roughly states that eigenvalues are preserved up to error ϵ in a simulation (in particular, the minimum eigenvalues satisfy $|\lambda(H') - \lambda(H)| \leq \epsilon$).

For any local measurement A_S acting on subset of S qubits $\mathcal{H}_S$ (here $\mathcal{H}_S$ is the Hilbert space for qudits in set $S \subseteq [n]$), we can define the local measurement $A'_S = V_S A_S V_S^\dagger$ on $\mathcal{H}'_S$ where $V = \bigotimes V_i$ is the local isometry in the definition of simulation and $V_S := \bigotimes_{i\in S} V_i$. Note that A'_S acts only on the $O(|S|)$ qudits which V_S maps to. Furthermore, $V^\dagger(A'_S \otimes I)V = A_S \otimes I$ and so

$$\begin{aligned}
|\langle\psi'|A'_S\otimes I|\psi'\rangle - \langle\psi|A_S\otimes I|\psi\rangle| &= |\langle\psi'|A'_S\otimes I|\psi'\rangle - \langle\psi|V^\dagger(A'_S\otimes I)V|\psi\rangle| && (7)\\
&\leqslant \|A'_S\|\||\psi'\rangle\langle\psi'| - V|\psi\rangle\langle\psi|V^\dagger\|_1 && (8)\\
&\leqslant \|A_S\|\left(\||\psi'\rangle\langle\psi'| - |\phi\rangle\langle\phi|\|_1 + \|\widetilde{V}|\psi\rangle\langle\psi|\widetilde{V}^\dagger - V|\psi\rangle\langle\psi|V^\dagger\|_1\right) && (9)\\
&\leqslant \|A_S\|\left(\||\psi'\rangle\langle\psi'| - |\phi\rangle\langle\phi|\|_1 + 2\|\widetilde{V} - V\|\right) && (10)\\
&\leqslant \|A_S\|\left(2\sqrt{\frac{\delta'}{\Delta - \lambda(H')}} + 2\eta\right) && (11)
\end{aligned}$$

where to get to (10), we have used the triangle inequality to bound:

$$\begin{aligned}
&\|\widetilde{V}|\psi\rangle\langle\psi|\widetilde{V}^\dagger - V|\psi\rangle\langle\psi|V^\dagger\|_1 && (12)\\
&\leq \|\widetilde{V}|\psi\rangle\langle\psi|\widetilde{V}^\dagger - V|\psi\rangle\langle\psi|\widetilde{V}^\dagger\|_1 + \|V|\psi\rangle\langle\psi|\widetilde{V}^\dagger - V|\psi\rangle\langle\psi|V^\dagger\|_1 && (13)\\
&= \|\widetilde{V} - V\|\left(\||\psi\rangle\langle\psi|\widetilde{V}^\dagger\|_1 + \|V|\psi\rangle\langle\psi|\|_1\right) && (14)\\
&= 2\|\widetilde{V} - V\| && (15)
\end{aligned}$$

Therefore, to ensure that Π' is a YES (resp. NO) instance if Π is a YES (resp. NO) instance, we will choose $a' = a + (b-a)/3$ and $b' = b - (b-a)/3$. Choosing $\delta', \Delta, \epsilon, \eta$ such that

$$0 < \delta' + 2\epsilon < \delta \qquad \text{and} \qquad 0 < \|A\|\left(2\sqrt{\frac{\delta'}{\Delta - \lambda(H')}} + 2\eta\right) < \frac{b-a}{3}$$

completes the proof. ◀

Proof of Theorem 5. We first discuss containment in the claimed complexity classes, and then hardness.

Containment. In the first case it is trivial to simulate the outcome of 1-local measurements on the ground state of a 1-local Hamiltonian, as the ground state is an easily calculated product state. For the other three cases, it was shown in [12] and [7], that k-LH for these three families of Hamiltonians is complete for the classes NP, StoqMA, QMA, respectively. Therefore, by Lemma 11, APX-SIM is contained in $\mathrm{P}^{\mathrm{NP[log]}}, \mathrm{P}^{\mathrm{StoqMA[log]}}$ and $\mathrm{P}^{\mathrm{QMA[log]}}$, respectively. (Note that the precondition of Lemma 11 is met, i.e. for H and A given as a linear combination of terms from $\mathcal{S}$ and I, we have that k-LH for $\alpha H + \beta A$ is contained in the respective complexity class of NP, StoqMA, or QMA, for any $0 \leq \alpha, \beta \leq \mathrm{poly}(n)$, and for all $k \geq 1$.)

Hardness. Starting with the referenced completeness results of [12, 7] above, we now wish to show APX-SIM is hard for $\mathrm{P}^{\mathrm{NP[log]}}, \mathrm{P}^{\mathrm{StoqMA[log]}}$ and $\mathrm{P}^{\mathrm{QMA[log]}}$ for cases 2–4 of our claim. At first glance, it may seem that Theorem 2 already yields this result, since that theorem says that APX-SIM is $\mathrm{P}^{\mathrm{C[log]}}$-complete when restricted to k-local Hamiltonians and observables from a family $\mathcal{F}$. Unfortunately, however, a precondition of Theorem 2 is that $\mathcal{F}$ must contain all classical (i.e. diagonal in standard basis) Hamiltonians, which is *not* necessarily true for cases 2–4 of our claim here. Thus, some work is required get the hardness claims of cases 2–4 here.

To achieve this, we first apply Lemma 12 to conclude that $\forall$-APX-SIM is hard for classes $\mathrm{P}^{||\mathrm{NP}}, \mathrm{P}^{||\mathrm{StoqMA}}$ and $\mathrm{P}^{||\mathrm{QMA}}$ for the families of classical, stoquastic and arbitrary local Hamiltonians, respectively. (In contrast to the Hamiltonians of cases 2–4 of our claim here, the sets of classical, stoquastic and arbitrary local Hamiltonians *do* contain all diagonal Hamiltonians, and thus satisfy the preconditions of Lemma 12.) We then use simulations, in combination with Lemma 15, to reduce the sets of classical, stoquastic, and arbitrary local Hamiltonians to the Hamiltonians in cases 2,3,4 of our claim here, respectively.

Specifically, it was shown in [13] that the three families of Hamiltonians in cases 2–4 of our claim can efficiently simulate all classical, stoquastic and arbitrary local Hamiltonians, respectively, via some local isometry V (see Definition 14). It follows by Lemma 15 (which states that simulations act like hardness reductions) that $\forall$-APX-SIM is hard for $\mathrm{P}^{||\mathrm{NP}}, \mathrm{P}^{||\mathrm{StoqMA}}$ and $\mathrm{P}^{||\mathrm{QMA}}$ respectively, with respect to (using the notation of Lemma 15) a local observable A' (in the larger, simulating, space) such that $A' = VAV^\dagger$ (where in our case A will equal Pauli Z due to the proof of Lemma 12). The only obstacle to achieving our current claim is that we also require A' to be chosen as a linear combination of terms from $\mathcal{S}$ and I. This is what the remainder of the proof shall show.

Observation (*). To begin, note the proof of Lemma 12 used single qubit observable Z, since we encoded the P machine's output in a single bit, which we assumed was set to $|0\rangle$ for "reject" and $|1\rangle$ for "accept". However, without loss of generality, we may alter the starting P machine to encode its output in some more general function on two bits, such as the parity function. (For example, the P machine can be assumed to output a 2-bit string q, such that q has odd parity if and only if the P machine wishes to accept.) We use this observation as follows. Consider any classical observable A with two distinct eigenvalues $\lambda_x < \lambda_y$ corresponding to eigenstates $|x\rangle$ and $|y\rangle$, respectively, for distinct strings $x, y \in \{0,1\}^2$. Then, assuming the specification of A is independent of the number of qubits in the system (thus, A is specified to within constant bits of precision, and so $\lambda_y - \lambda_x \in \Theta(1)$), if we set the P machine to output x when it wishes to accept and y when it wishes to reject, a measurement with observable A suffices to distinguish these two cases. With this observation in hand, we consider cases 2–4 of our claim, in particular with respect to the action of isometry V.

Case 2: $\mathrm{P}^{||\mathrm{NP}}$*-completeness.* First note that in this case we can assume without loss of generality that all interactions in $\mathcal{S}$ are diagonal (by performing a global basis change of $U^{\otimes n}$ if necessary) . Since we are not in the first case we know also that there is a 2-local interaction in $\mathcal{S}$ with at least two distinct eigenvalues. By Observation (*), it will suffice to simulate such an observable on a particular pair of qubits in the original system; call this operator A. For the $\mathrm{P}^{\mathrm{NP[log]}}$ case, the isometry V appends some ancilla qubits in a computational basis state (in the $U^{\otimes n}$ basis) [15]. We can therefore choose A' to be the same 2-local observable A, but acting on the corresponding qubits in the larger, simulating system; that is, if we let $A' = A \otimes I$ (where the identity term acts on the ancilla qubits), then $V^\dagger A' V = A$ as desired.

Case 3: $\mathrm{P}^{||\mathrm{StoqMA}}$*-completeness.* For the third case, one can check that the reductions in [7] correspond to a simulation with an isometry V which maps each qubit $|0\rangle \mapsto |0011\rangle$ and $|1\rangle \mapsto |1100\rangle$ and appends some additional ancilla qubits in a computational basis state (see discussion in Section 9.4 of [13]). Thus, a classical 2-local observable $Z \otimes Z + \mathrm{diag}(A) \otimes I + I \otimes \mathrm{diag}(B)$ (which we may use by Observation (*)) can be simulated in the larger, simulating space on physical qubits $1, 2, 3, 4$ (logical qubit 1) and $5, 6, 7, 8$ (logical qubit 2) via:

$$V^\dagger (Z_1 Z_5 + A_1 + B_5) V = Z \otimes Z + \mathrm{diag}(A) \otimes I + I \otimes \mathrm{diag}(B),$$

where $\mathrm{diag}(A)$ denotes the diagonal part of A, i.e. $\mathrm{diag}(A) = \sum_{i=0}^{1} |i\rangle\langle i|A|i\rangle\langle i|$. Thus, measuring observable $(Z_1 Z_5 + A_1 + B_5)$ on the larger, simulating Hamiltonian H' (which has the desired form of Case 3 here) is equivalent to measuring $Z \otimes Z + \mathrm{diag}(A) \otimes I + I \otimes \mathrm{diag}(B)$ on the starting Hamiltonian H in the simulation (again, using notation of Lemma 15).

Case 4: $\mathrm{P}^{||\mathrm{QMA}}$*-completeness.* The final case is slightly more complicated. When showing that these Hamiltonians are universal, the one step with a non-trivial isometry is simulating $\{X, Z, XX, ZZ\}$-Hamiltonians with $\{XX + YY\}$-Hamiltonians or $\{XX + YY + ZZ\}$-Hamiltonians in Theorem 41 of [13]. In both of these cases, the isometry V maps each qubit via action

$$|0\rangle \mapsto |\Psi^-\rangle_{13} |\Psi^-\rangle_{24} \qquad |1\rangle \mapsto \tfrac{2}{\sqrt{3}} |\Psi^-\rangle_{12} |\Psi^-\rangle_{34} - \tfrac{1}{\sqrt{3}} |\Psi^-\rangle_{13} |\Psi^-\rangle_{24}.$$

In the proof of Theorem 41 of [13], it is shown that a single Z observable can be reproduced by choosing $A = h_{13}$ (where either $h = XX + YY$ or $h = XX + YY + ZZ$), that is $V^\dagger h_{13} \otimes I_{24} V$ is proportional to Z.

The proof is completed by Corollary 3 (i.e. logarithmic adaptive queries are equivalent to polynomially many parallel queries). ◀

D Proofs for Section 5

Proof of Lemma 17. We will construct a Hamiltonian on the registers $\mathcal{W}$, $\mathcal{X}_i$ and $\mathcal{Y}_i$ for $i \in \{1, \ldots m\}$, for which the problem $\forall$-APX-SIM encodes the output of a $\mathrm{P}^{||\mathrm{QMA}}$ circuit, where m is the number of parallel queries to the QMA oracle.

Let the qubits of $\mathcal{W}$ and $\mathcal{Y}_i$ be arranged on distinct parts of a square lattice. For each qubit of $\mathcal{Y}_i$, there is a corresponding qubit in $\mathcal{X}_i$, and $\mathcal{X}_i$ contains a path of qubits leading from $\mathcal{Y}_i$ to $\mathcal{W}$. See Figure 2 for an example layout in the case $m = 3$.

Let E_i be the set of edges of the square lattice of qubits of $\mathcal{Y}_i$ (i.e. not including the edges connecting $\mathcal{Y}_i$ to $\mathcal{X}_i$ in Figure 2) and let $H_{\mathcal{Y}_i} = \sum_{(j,k) \in E_i} h^i_{\mathcal{Y}_i(j,k)}$ be a 2D nearest neighbor Hamiltonian on $\mathcal{Y}_i$ corresponding to the i-th query. We have used the subscript notation $\mathcal{Y}_i(j,k)$ to denote the action of an operator on the j-th and k-th qubits of the $\mathcal{Y}_i$ register.

$H_{\mathcal{Y}_i}$ has ground state energy less than a_i if query i is a YES instance and energy greater than b_i in a NO instance. Then, let $H_2 = \sum_i H_2^{(i)}$ where

$$H_2^{(i)} = \frac{a_i + b_i}{2}|0\rangle\langle 0|_{\mathcal{X}_i(1)} \otimes I_{\mathcal{Y}_i} + \sum_{(j,k)\in E_i} \left(|1\rangle\langle 1|_{\mathcal{X}_i(g(j,k))} \otimes h^i_{\mathcal{Y}_i(j,k)}\right),$$

where $g(j,k)$ is the location of the "nearest" qubit in $\mathcal{X}_i$ to edge (j,k) in $\mathcal{Y}_i$. Here, the choice "nearest" is somewhat arbitrary; for concreteness, one can set $g(j,k) = j$, i.e. pick the vertex in $\mathcal{X}_i$ which aligns with the first coordinate of the edge (j,k). (In this sense, Figure 2 is not entirely accurate, since it depicts the 3-local constraint $|1\rangle\langle 1|_{\mathcal{X}_i(g(j,k))} \otimes h^i_{\mathcal{Y}_i(j,k)}$ as a pair of 2-local constraints. This is done solely for the purpose of simplifying the illustration, as otherwise one would need to draw hyperedges of size 3.)

Let $H_1 = H_{\text{prop}} + H_{\text{in}}$ be the Cook-Levin Hamiltonian where H_{prop} is exactly as in Lemma 12. Let H_{in} initialize the qubits of the first ($t = 1$) row of the qubits in $\mathcal{W}$. For each query i, we have a penalty term $|1\rangle\langle 1|_{\mathcal{X}_i(1)}|0\rangle\langle 0| + |0\rangle\langle 0|_{\mathcal{X}_i(1)}|1\rangle\langle 1|$ which effectively copies the state of $\mathcal{X}_i(1)$, the qubit in $\mathcal{X}_i$ nearest to $\mathcal{W}$, onto the i-th qubit of the first row of $\mathcal{W}$. For all the remaining qubits in the first ($t = 1$) row of $\mathcal{W}$, we have a penalty term $|1\rangle\langle 1|$, effectively initializing the qubit into the $|0\rangle$ state.

Restricted to the subspace $\mathcal{H}$ where each $\mathcal{X}_i$ register is either all $|0\rangle$ or all $|1\rangle$, $H_1 + H_2$ is exactly the same Hamiltonian as in Lemma 12. It remains to give a high energy penalty to all other states not in this subspace. We do this with $H_3 = \sum_{i=1}^m H_3^{(i)}$ where each term $H_3^{(i)}$ acts on $\mathcal{X}_i$:

$$H_3^{(i)} = \Delta_i \sum_{(j,k)\in G_i} \left(|0\rangle\langle 0|_{\mathcal{X}_i(j)}|1\rangle\langle 1|_{\mathcal{X}_i(k)} + |1\rangle\langle 1|_{\mathcal{X}_i(j)}|0\rangle\langle 0|_{\mathcal{X}_i(k)}\right)$$

where G_i is the set of edges between the qubits of the $\mathcal{X}_i$ register. G_i consists of edges between nearest neighbors on the square lattice E_i and on the path of qubits from $\mathcal{Y}_i$ to $\mathcal{W}$. The overall Hamiltonian $H = H_1 + H_2 + H_3$ is therefore spatially sparse.

$H_3^{(i)}$ is a classical Hamiltonian, so all of its eigenstates can be taken to be of form $|x\rangle$ for some $x \in \{0,1\}^{n_i}$. Its ground space $\mathcal{G}_i$ contains $|0\rangle^{\otimes n_i}$ and $|1\rangle^{\otimes n_i}$; and all states in $\mathcal{G}_i^\perp$ have energy at least Δ_i. Choosing $\Delta_i > \delta + \sum_{(j,k)\in E_i} \|h^i_{\mathcal{Y}_i(j,k)}\|$ ensures that all states in $\mathcal{G}_i^\perp$ have energy greater than $\lambda(H) + \delta$.

Then $H = H_1 + H_2 + H_3$ is block diagonal with respect to the split of each subspace $\mathcal{G}_i \oplus \mathcal{G}_i^\perp$; restricted to the spaces $\mathcal{G}_i$, H is exactly the Hamiltonian from Lemma 12, and all states in spaces $\mathcal{G}_i^\perp$ have energy greater than $\lambda(H) + \delta$. The result then follows just as in the proof of Lemma 12. ◀

E Proofs for Section 6

We now give all details of our 1D hardness construction from Section 6, and prove correctness thereof in Section E.1.

Our 1D hardness construction. We give a reduction from $\text{P}^{||\text{QMA}}$ to $\forall$-APX-SIM, which by Theorem 2 yields the claim. Let Π be a $\text{P}^{||\text{QMA}}$ computation which takes in an input of size n and which consists of a uniformly generated polynomial-size classical circuit C making $m = O(\log n)$ 2-LH queries $\pi_i := (H_i, a_i, b_i)$ to a QMA oracle. As in Lemma 12, we treat the "answer register" in which C receives answers to its m queries as a proof register.

Our high-level approach consists of three steps: (1) construct a "master" circuit V composed of the verification circuits V_i corresponding to each query π_i and of the circuit C; (2) run V through the 1D circuit-to-Hamiltonian construction of [26] to obtain a 1D Hamiltonian G with local dimension 8 constructed such that the low-energy space $\mathcal{S}$ of G must consist of history states (of the form described in [26]); and (3) carefully add additional 1-local penalty terms acting on the output qubits corresponding to each verification circuit V_i to obtain final Hamiltonian H such that the low-energy space must encode satisfying proofs to each V_i whenever possible. This final step of "fine-grained splitting" of S forces the output qubits of the circuits V_i to encode correct answers to query π_i, and thus the final circuit C receives a correct proof, hence leading the history states of step (2) to encode a correct simulation of Π. The answer to the computation Π can then be read off the ground state of H via an appropriate single qudit measurement.

1. Construction of V. Suppose each query π_i has corresponding QMA verification circuit V_i. Without loss of generality, we may henceforth assume that the completeness/soundness error of V_i is at most $p \leq 2^{-n}$, for p to be set later, by standard error reduction [2, 34]; thus, if a particular query (H_i, a_i, b_i) is valid (i.e. $\lambda(H) \notin (a_i, b_i)$), then either there exists a proof such that V_i outputs YES with probability at least $1-p$ or no proof causes V_i to output YES with probability greater than p. Next, since Π is a $\mathrm{P}^{||\mathrm{QMA}}$ computation, all queries and corresponding V_i can be precomputed in polynomial-time. We view the "master circuit" V as consisting of two phases:

1. (Verification phase) Given supposed proofs for each query, V runs all verification circuits V_i in parallel, where V_i acts on space $\mathcal{Y}_i \otimes \mathcal{W}_i \otimes \mathcal{X}_i$, for proof register $\mathcal{Y}_i$, ancilla register $\mathcal{W}_i$, and single-qubit output register $\mathcal{X}_i$.
2. (Simulated classical phase) The simulated P circuit C now receives the query answers $\mathcal{X} := \mathcal{X}_1 \otimes \cdots \otimes \mathcal{X}_m$ as its proof register as well as an ancilla register $\mathcal{W}_0$. It outputs a single qubit to an output register $\mathcal{X}_0$.

This completes the construction of V, which acts on $\mathcal{Y} \otimes \mathcal{W} \otimes \mathcal{X}$, where $\mathcal{Y} = \bigotimes_{i=1} \mathcal{Y}_i$,$\mathcal{W} = \bigotimes_{i=1} \mathcal{W}_i$, and $\mathcal{X} = \bigotimes_{i=1} \mathcal{X}_i$. Crucially, note that given a set of proofs in register $\mathcal{Y}$, V does *not* necessarily yield the same answer as Π, since a malicious prover could intentionally send a "bad" proof to a YES query, flipping the final answer of V.

2. Construction of G. We now plug V into the circuit-to-Hamiltonian construction of Hallgren, Nagaj, and Narayanaswami [26] to obtain a nearest-neighbor 1D Hamiltonian $G' = \Delta_{\mathrm{in}} H_{\mathrm{in}} + \Delta_{\mathrm{prop}} H_{\mathrm{prop}} + \Delta_{\mathrm{pen}} H_{\mathrm{pen}} + H_{\mathrm{out}}$, where $\Delta_{\mathrm{in}}, \Delta_{\mathrm{prop}}$, and Δ_{pen} are at most polynomials in n which we will set as needed; we review this construction more closely below. Set $G = G' - H_{\mathrm{out}}$, since in our setting the task of "checking the output" will be delegated to the observable A. Note that as an intermediate step, [26] maps V to a circuit V' which it then maps to G'; we describe the role of V' in the following review. Our construction will make two trivial assumptions about the behavior of V', including how it arranges its query answers between the verification phase and the simulated classical phase and how it stores its output in the final timestep; we defer details about these assumptions until we define our "fine-grained splitting" in step 3 and when we define our observable.

Review of 1D QMA construction [26]. Suppose an arbitrary circuit U acts on n qubits. Begin by arbitrarily arranging these qubits along a line. The circuit U is then "linearized", meaning it is mapped to a new circuit U' which consists of R rounds in which each round applies a sequence of $n-1$ two-qubit gates acting on nearest neighbors. The i-th gate in

a round acts on qubits $(i, i+1)$. This "linearization" is achieved in polynomial time by inserting swap and identity gates as needed, and U' is at most polynomially larger than U.

To reduce U' to an instance of k-LH, we wish to design a mapping similar to Kitaev's circuit-to-Hamiltonian construction for showing QMA-hardness of 5-LH on general geometry [31]. In both settings, the goal is to design an H which enforces a structure on any state in its low-energy space. In the construction of [31], $H = H_{\text{in}} + H_{\text{prop}} + H_{\text{stab}} + H_{\text{out}}$, and the minimizing state of H has the form of a *history state*:

$$|\eta\rangle = \frac{1}{\sqrt{L+1}} \sum_{t=0}^{L} U_t \cdots U_1 |\psi\rangle_{\mathcal{Y}} |0 \cdots 0\rangle_W |t\rangle_C .$$

Intuitively, H_{stab} forces a structure on the clock register C of basis states $|0\rangle, |1\rangle, \ldots,$ such that each will correspond to a timestep of U. Then, H_{in} ensures the ancilla register W is set to the all $|0\rangle$ state when $|t\rangle = |0\rangle$. The term H_{prop} ensures that the workspaces entangled with timesteps $|t\rangle$ and $|t+1\rangle$ are related by the 2-qubit gate U_{t+1}. Together, these terms ensure that a minimizing state $|\psi_{\text{hist}}\rangle$ encodes a correct simulation of the circuit U, and that all low-energy states are close to $|\psi_{\text{hist}}\rangle$. In fact, a valid $|\psi_{\text{hist}}\rangle$ lies in the nullspace of $H_{\text{in}} + H_{\text{prop}} + H_{\text{stab}}$. Finally, H_{out} penalizes the low-energy space if the output qubit has overlap with $|0\rangle$.

Now in the 1D setting, the goal remains the same: design H such that the structure of its low-energy state is a superposition over a sequence of states corresponding to timesteps in the computation of U'. But, we now appear unable to entangle the workspace with a separate clock register using nearest neighbor interactions. Instead, the constructions of [1, 26] employ qudits of higher dimension as a means to label the qubits, with each labeling encoding a particular timestep. [26] then doubles the number of qudits in order to lower the necessary number of labels. The construction of [26] thus maps U' to a Hamiltonian $H = H_{\text{in}} + H_{\text{prop}} + H_{\text{out}} + H_{\text{pen}}$ acting on $2nR$ qudits of dimension 8, where the qudits are arranged on a 1D line in R blocks of $2n$ qudits (i.e. one block per round in U').

Let us further describe the idea of labeling, or "marking", of qudits. For example, a qubit $\alpha|0\rangle + \beta|1\rangle$ may be encoded as $\alpha|A\rangle + \beta|B\rangle$ if that qubit is ready for a gate to be applied or as $\alpha|C\rangle + \beta|D\rangle$ if that round's gate has already been applied, where $|A\rangle, |B\rangle, |C\rangle, |D\rangle$ are some basis states. The possible configurations, or arrangements, of labels along the line form a set of orthogonal spaces. [26] thus introduces a Hamiltonian term H_{pen} which enforces a set of "legal configurations" of the workspace, penalizing all other configurations. We then map each of the configurations which remain in the low-energy space of H to timesteps in the computation of U', effectively assigning the job of encoding the workspace in a particular timestep to a particular configuration of qudits. We note that the crucial feature of the set of legal configurations developed by [26] is that they are sufficiently identifiable solely by 2-local nearest neighbor checks[5] such that penalties can be correctly assigned when constructing 1D analogs of the terms $H_{\text{in}}, H_{\text{prop}}, H_{\text{out}}$. Similar to the general geometry case of [31], the construction of [26] enforces that the nullspace of $H_{\text{in}} + H_{\text{prop}} + H_{\text{pen}}$ consists of history states

$$|\psi_{\text{hist}}\rangle = \frac{1}{\sqrt{L+1}} \sum_{t=0}^{L} |\psi_t\rangle , \tag{16}$$

[5] For clarity, in [26] not all illegal configurations are immediately detectable by H_{pen}. Any such undetectable illegal configurations are instead shown to eventually evolve under H_{prop} into detectable illegal configurations.

such that $|\psi_{\text{hist}}\rangle$ is a superposition over states in each legal configuration, $|\psi_0\rangle$ encodes a properly initialized workspace, and each pair $|\psi_t\rangle$ and $|\psi_{t+1}\rangle$ are related according to the corresponding timestep of U'. Finally, again similar to the general geometry case, all low-energy states must be close to $|\psi_{\text{hist}}\rangle$ (we make these two claims explicit and give proofs in Lemma 22).

The full description of the labeling, the legal configurations, and their mapping to timesteps by [26] is rather involved. Here, we introduce sufficient details for our later analysis. We begin with a single block of $2n$ qudits, where recall each block is used to encode a single round (taken from [26]):

$$\|\blacktriangleright\circledcirc\,|\square\circledcirc\,|\cdots|\square\circledcirc\,|\square\bigcirc\| \tag{17}$$

Recall the design of U' began by arranging the qubits of U arbitrarily on the line; the i-th qubit on that line corresponds to qudits $2i-1$ and $2i$ in (17). Thus, each qubit of U', henceforth denoted a *logical qubit*, is encoded into two consecutive qudits. Each pair of qudits representing a logical qubit is depicted as separated by a $|$ for clarity. The standard basis for each 8-dimensional qudit is labeled by

$$\{\,|\bigcirc\rangle,\ |\ominus\rangle,\ |\circledcirc\rangle,\ |\otimes\rangle,\ |\blacktriangleright_0\rangle,\ |\blacktriangleright_1\rangle,\ |\square_0\rangle,\ |\square_1\rangle\,\},$$

where, as described earlier, the current state of a qudit can be used to encode a logical qubit and to label the qudit. The first four states should be thought of as 1-dimensional labels; they are used to ensure the correct propagation of the circuit and do not encode a logical qubit. The final four states are used to either label a qudit with $\blacktriangleright$, in which case a logical qubit is encoded as a superposition of $|\blacktriangleright_0\rangle$ and $|\blacktriangleright_1\rangle$, or with $\square$, in which case a logical qubit is encoded as a superposition of $|\square_0\rangle$ and $|\square_1\rangle$. To make this example more concrete, a product state of $(\alpha|0\rangle+\beta|1\rangle)^{\otimes n}$ on n logical qubits could be encoded as

$$(\alpha\,|\blacktriangleright_0\rangle+\beta\,|\blacktriangleright_1\rangle)\otimes|\circledcirc\,\rangle\otimes(\alpha\,|\square_0\rangle+\beta\,|\square_1\rangle)\otimes|\circledcirc\,\rangle\otimes\cdots \tag{18}$$

$$\otimes\,(\alpha\,|\square_0\rangle+\beta\,|\square_1\rangle)\otimes|\bigcirc\,\rangle\,. \tag{19}$$

Next, here is an example depicting multiple blocks (from Table 2 of [26]):

$$\cdots\otimes\otimes\|\blacktriangleright\circledcirc\,|\square\circledcirc\,|\square\bigcirc\|\bigcirc\,\bigcirc\,|\bigcirc\,\bigcirc\,|\bigcirc\,\bigcirc\|\bigcirc\,\bigcirc\cdots, \tag{20}$$

where the blocks are delineated by $\|$. The labels $\otimes$ to the left depict "dead" qudits, while the labels $\bigcirc$ to the right depict "unborn" qudits. By construction, all logical qubits are encoded in a block between the dead and unborn labels. In this example, the logical qubits line up with the beginning of a new block, beginning with $\|\blacktriangleright$ and ending with the first $\bigcirc\|$.

At a high level, the set of legal configurations is mapped to a sequence of timesteps as follows. The first timestep corresponds to a configuration similar to (17), with n logical qubits encoded in the leftmost block of $2n$ qudits, with no $\otimes$ labels anywhere, and with the "gate" label $\blacktriangleright$ on the first qudit. The second configuration has the $\blacktriangleright$ label shifted to the right, on the second qudit. Next, the third configuration has the second qudit labeled $\square$ and the third qudit labeled $\blacktriangleright$. This propagation of the $\blacktriangleright$ label rightwards continues, with each step corresponding to another legal configuration, until it reaches the end of the block. As the $\blacktriangleright$ passes between logical qubits $(i, i+1)$, the corresponding configurations map to timesteps i and $i+1$ of round 1, and H_{prop} enforces that configurations are related by the application of gate U'_i. Thus, when we reach a configuration with $\blacktriangleright$ at the end of the block, i.e. $\blacktriangleright\|$, all gates in the current round will have been applied. Next, before encoding the

next round of gates, our goal becomes to shift all of the logical qubits encoded in the current block rightwards $2n$ spots into the second block. To do this, the $\blacktriangleright$ label becomes a special $\ominus$ label and moves to the left one spot at a time until it reaches the end of the logical qubits (here, the leftwards $\|$). As the label $\ominus$ moves left, it shifts each logical qubit to the right one spot, i.e. $|\square\ominus\rangle \to |\ominus\square\rangle$. This process repeats, with a label propagating rightwards to the end of the logical qubits (now past the rightwards $\|$), then the label $\ominus$ propagating to the left, shifting logical qubits to the right, and so on, until the logical qubits have shifted entirely into the second block. Then, the gate label $\blacktriangleright$ once again transitions down the line, with successive configurations encoding the second round of gates of U'. Throughout this sequence, $\bigcirc$ labels to the right are consumed, while all qudits to the left are labeled $\otimes$. This procedure continues until the entire circuit has been simulated.

Lastly, we observe that the final timestep of U' is encoded by [26] in the following configuration:

$$\cdots \otimes \otimes \| \otimes \otimes \| \otimes \square | \circledcirc \square | \cdots | \circledcirc \square | \circledcirc \blacktriangleright \| \tag{21}$$

3. Adding 1-local "sifters". We now add 1-local Hamiltonian terms which serve to "sift" through bad proofs, or more accurately to split the ground space of G, so as to force low-energy states to encode correct query answers. As previously described, even a correct simulation of the circuit V may not output the correct answer for instance Π if a malicious prover supplies incorrect proofs to the query registers $\mathcal{Y}_i$; in particular, a prover might send a proof which accepts with low probability even though π_i is a YES-instance. Intuitively, we wish to penalize states encoding a proof $|\psi_i\rangle$ which leads verifier V_i to reject with high probability when there exists a proof $|\phi_i\rangle$ such that V_i would have accepted with high probability (here, query π_i is a YES instance). For answer register $\mathcal{X}_i$, we add a "sifter" penalty term $\epsilon\,|0\rangle\langle 0|_{\mathcal{X}_i}$, for ϵ some inverse polynomial to be set later. These terms are similar to the H_{out} term from other Hamiltonian constructions; but, here we are not only concerned about the ground space but also about the low-energy space. As in other constructions, we must penalize NO answers enough to ensure the ground space encodes YES answers when possible. But, given a correct NO answer, the penalty must be small enough that the energy is gapped lower than any state which encodes an incorrect YES, such as those which by encode an invalid computation leading to YES.

However, because the encoding enforced by G shifts the block of logical qubits rightwards along the line as the computation progresses, the location of a particular logical qubit's encoding depends on the current timestep. Thus, in order to properly act on logical qubit $\mathcal{X}_i$, we must be careful to specify the configuration which the penalty term acts on.

We may assume that once V' finishes simulating all of the circuits V_i, it arranges each of the outputs in the first m logical qubits on the line, finishing by the end of some round $r^* - 1$, such that the i-th logical qubit on the line is the qubit which V stored in $\mathcal{X}_i$. (The value of r^* can be determined during the construction of V'.) We may also assume that V' then "pauses" by applying only identity gates in round r^*. This round is encoded in block r^*, and since each block is comprised of $2n$ qudits, the answers to queries 1 to m are thus simultaneously stored in qudits

$$q_i := (2n)(r^* - 1) + (2i - 1). \tag{22}$$

The m sifter terms are given by

$$H_{\text{out},i} = \epsilon\,|\blacktriangleright_0\rangle\langle\blacktriangleright_0|_{q_i},$$

where the subscript denotes the qudit which the term acts on and ϵ is to be set later. Note that there is a unique legal configuration in which any given qudit is labeled $\blacktriangleright$, so $H_{\text{out},i}$ will apply to at most one state $|\psi_t\rangle$ in the history state of Equation (16). Finally, we define $H_{\text{out}} = \sum_{i=1}^{m} H_{\text{out},i}$.

The final Hamiltonian. Our final Hamiltonian is $H := G + H_{\text{out}} = \Delta_{\text{in}} H_{\text{in}} + \Delta_{\text{prop}} H_{\text{prop}} + \Delta_{\text{pen}} H_{\text{pen}} + H_{\text{out}}$, with $\Delta_{\text{in}}, \Delta_{\text{prop}}, \Delta_{\text{pen}}$ polynomials to be set later.

The observable. Recall the configuration from (21), which corresponds to the final timestep in the computation of a circuit passed to the construction of [26]. Note that this is the unique timestep in which the final qudit is labeled $\blacktriangleright$. We assume, without loss of generality, that V' places its final output in the rightmost logical qubit on the line. Thus, we choose single-qudit observable $A = |\blacktriangleright_0\rangle\langle\blacktriangleright_0|_{2nR}$, where the subscript denotes that A acts on the rightmost qudit on the line, where R is the number of rounds in V'.

Setting parameters. Let L denote the number of legal configurations which the history state in (16) is summed over, which is at most polynomial in n. We have that H is k-local and A is ℓ-local for $k := 2$ and $\ell := 1$. Set $\epsilon = 1/(8m)$, where recall m is the (polynomial) number of queries. Then, set p, the completeness/soundness error of each V_i, to some inverse-exponential in n such that $p < \epsilon$ for all n. Set $a = 1/(4L)$ and $b = 3/(4L)$. We will set δ to a sufficiently small fixed inverse polynomial in n in the proof of Lemma 23, which will then set $\Delta_{\text{in}}, \Delta_{\text{prop}}, \Delta_{\text{pen}}$ to sufficiently large fixed polynomials in n via the proof of Lemma 22.

This concludes our deterministic polynomial-time mapping of the input $\text{P}^{||\text{QMA}}$ computation Π to the 1D instance $\tilde{\Pi} := (H, A, k, \ell, a, b, \delta)$ of $\forall$-APX-SIM.

E.1 Correctness

We now prove Theorem 10 by showing correctness of our construction from Section 6. A number of lemmas required in the proof are deferred to Section E.1.1 to ease the exposition; in particular, we require Lemma 22, which explicitly proves two facts about the low-energy space of the construction of [26], Lemma 23, which shows that a history state in our construction must simultaneously encode nearly correct answers for all valid queries π_i, and Lemma 24, which states a Commutative Quantum Union Bound.

Proof of Theorem 10. Containment in $\text{P}^{\text{QMA}[\log]}$ was already shown for up to $O(\log n)$-local H by [3], with no restriction on the geometry. Our goal is now to show $\text{P}^{||\text{QMA}}$-hardness, which by Theorem 2 yields $\text{P}^{\text{QMA}[\log]}$-hardness. We show hardness for the problem $\forall$-APX-SIM, which recall from Section 1 trivially reduces to APX-SIM, thus yielding hardness for APX-SIM. Let Π be a $\text{P}^{||\text{QMA}}$ computation and map it to the $\forall$-APX-SIM instance $\tilde{\Pi} = (H, A, k, l, a, b, \delta)$ as described in Section 6. The proof proceeds in two parts: We first show that low energy states must necessarily encode correct query answers, and subsequently apply this to show correctness in YES and NO cases for Π.

Low energy states approximately encode correct query answers. Recall that $H = G + H_{\text{out}}$. Let δ, γ denote arbitrary inverse polynomials in n which will be set later in Lemma 23. Consider any state $|\psi\rangle$ such that $\langle\psi| H |\psi\rangle \leq \lambda(H) + \delta$. Since $H_{\text{out}} \succeq 0$, $\langle\psi| G |\psi\rangle \leq \lambda(H) + \delta$ as well. By Lemma 22, for sufficiently large fixed polynomials $\Delta_{\text{in}}, \Delta_{\text{prop}}, \Delta_{\text{pen}}$, two statements thus hold: First, the nullspace $\mathcal{S}$ of Hamiltonian $G = \Delta_{\text{in}} H_{\text{in}} + \Delta_{\text{prop}} H_{\text{prop}} + \Delta_{\text{pen}} H_{\text{pen}}$ is

the span of all correctly encoded history states, as defined in Equation (16); Second, there exists a correctly encoded history state $|\psi_{\text{hist}}\rangle$ such that

$$\| |\psi\rangle\langle\psi| - |\psi_{\text{hist}}\rangle\langle\psi_{\text{hist}}| \|_{\text{tr}} \leq \gamma. \tag{23}$$

Combining Equation (23) with the Hölder Inequality and the fact that $\| H_{\text{out}} \|_\infty = m\epsilon$ yields that

$$|\operatorname{Tr}[H_{\text{out}} |\psi\rangle\langle\psi|] - \operatorname{Tr}[H_{\text{out}} |\psi_{\text{hist}}\rangle\langle\psi_{\text{hist}}|]| \leq \gamma \| H_{\text{out}} \|_\infty = m\epsilon\gamma.$$

Since $|\psi_{\text{hist}}\rangle$ is a nullstate of G and $\langle\psi| H_{\text{out}} |\psi\rangle \leq \langle\psi| H |\psi\rangle \leq \lambda(H) + \delta$, we conclude

$$\langle\psi_{\text{hist}}| H |\psi_{\text{hist}}\rangle \leq \lambda(H) + \delta + m\epsilon\gamma. \tag{24}$$

Next, let $I \subseteq \{1, \ldots, m\}$ be the set of indices corresponding to valid queries π_i, and for all $i \in I$ define $x_i = 1$ if π_i is a YES-instance and $x_i = 0$ if π_i is a NO-instance.[6] Recall now from Section 6 that at the beginning of round r^*, V' has encoded the answer to the i-th QMA query in qudit q_i (defined in Equation (22)). Let $|\psi_{t^*}\rangle$ denote the unique (normalized) state in the superposition comprising $|\psi_{\text{hist}}\rangle$ in which q_1 is labeled ▶ (i.e. the first timestep corresponding to round r^*). Since during round r^*, V' only applies identity gates, the qubits encoded in qudits q_i during timestep t^*, in which q_1 is labeled ▶ and all other q_i are labeled □, are exactly the same as in successive timesteps in which other q_i are labeled by ▶. More formally, $|\langle\psi_{t^*}|\square_{x_i}\rangle_{q_i}|^2 = L|\langle\psi_{\text{hist}}|\blacktriangleright_{x_i}\rangle_{q_i}|^2$ for any $i \in I$, and so by Lemma 23,

$$\left|\langle\psi_{t^*}|\square_{x_i}\rangle_{q_i}\right|^2 \geq 1 - \epsilon, \tag{25}$$

where[7] we substitute the label ▶ for □ when $i = 1$, and where the factor of L^{-1} is removed due to the normalization of $|\psi_{t^*}\rangle$.

This is for any single query $\pi_i, i \in I$; from this, we can obtain that $|\psi_{t^*}\rangle$ simultaneously encodes nearly correct query answers to *all* valid queries. To do so, define $\Gamma := \Pi_{i\in I} |\square_{x_i}\rangle\langle\square_{x_i}|_{q_i}$ (where again, we replace label ▶ for □ when $i = 1$). Then, by the Commutative Quantum Union Bound (Lemma 24),

$$\langle\psi_{t^*}| \Gamma |\psi_{t^*}\rangle \geq 1 - |I|\,\epsilon \geq 1 - m\epsilon. \tag{26}$$

It follows that we may write $|\psi_{t^*}\rangle = \alpha |\phi_1\rangle + \beta |\phi_2\rangle$ for unit vectors $|\phi_1\rangle, |\phi_2\rangle$ such that $\Gamma |\phi_1\rangle = |\phi_1\rangle$ and $\Gamma |\phi_2\rangle = 0$, and where $\alpha, \beta \in \mathbb{C}, |\alpha|^2 + |\beta|^2 = 1$, and $|\alpha|^2 \geq 1 - m\epsilon$. Intuitively, $|\phi_1\rangle$ is the part of $|\psi_{t^*}\rangle$ that encodes correct strings of query answers on I, while $|\phi_2\rangle$ encodes strings with at least one incorrect query answer in I – for clarity, $|\phi_1\rangle$ may encode a superposition of multiple *distinct* correct strings of query answers, since queries with indices not in I may be answered arbitrarily.

---

[6] Without loss of generality, we may assume at least one query is valid ($I \neq \emptyset$). This is because if all queries are invalid, then all simulations of the P circuit C must output the same answer no matter the sequence of query answers C receives. Thus, all history states will encode the same final answer, and α (defined after (26)) equals 1, satisfying the lower bound found of $\alpha \geq 1 - m\epsilon$.

[7] We implicitly apply identity on all qudits other than q_i, i.e. $\left|\langle\psi_{\text{hist}}|\square_{x_i}\rangle_{q_i}\right|^2 := \operatorname{Tr}\left[|\psi_{\text{hist}}\rangle\langle\psi_{\text{hist}}| \left(I \otimes |\square_{x_i}\rangle\langle\square_{x_i}|_{q_i} \otimes I\right)\right]$.

Application to YES versus NO cases for Π. We have shown that for any low energy state $|\psi\rangle$, there exists a history state $|\psi_{\text{hist}}\rangle$ close to $|\psi\rangle$ which has large amplitude on all the correct query answers for set I in round r^*. We can now analyze the YES and NO cases for our $\text{P}^{\text{QMA[log]}}$ problem Π.

Recall that $|\phi_1\rangle$ may be a superposition over *multiple* correct query strings (due to invalid queries π_i for $i \notin I$). Nevertheless, since the classical circuit C for the $\text{P}^{\text{QMA[log]}}$ machine is required to output the *same* answer regardless of how invalid queries are answered (i.e. for any given correct string of query answers), all query strings which $|\phi_1\rangle$ is a superposition over lead C to output the same, correct final answer. Thus, setting $y = 0$ if Π is a YES-instance and $y = 1$ if Π is a NO-instance, we have

$$\left|\langle\psi_{\text{hist}}| A |\psi_{\text{hist}}\rangle - \frac{y}{L}\right| \leq \frac{m\epsilon}{L},$$

where the factor of L^{-1} is due to the fact A applies only to the final configuration/time step. Combining Equation (23) with the Hölder inequality yields that

$$|\operatorname{Tr}[A|\psi\rangle\langle\psi|] - \operatorname{Tr}[A|\psi_{\text{hist}}\rangle\langle\psi_{\text{hist}}|]| \leq \gamma,$$

since $\|A\|_\infty = 1$, and so

$$\left|\langle\psi| A |\psi\rangle - \frac{y}{L}\right| \leq \frac{m\epsilon}{L} + \gamma,$$

Given that we set $\delta = \gamma = 1/(256m^2L) < 1/(8L)$ in Lemma 23 and $\epsilon = 1/(8m)$, we have that $\gamma + m\epsilon/L < 1/(4L)$. We conclude that for all low-energy states $|\psi\rangle$ (i.e. states satisfying $\langle\psi| H |\psi\rangle \leq \lambda(H) + \delta$), if Π is a YES-instance then $\langle\psi| A |\psi\rangle \leq 1/(4L)$ (i.e. we have a YES instance of ∀-APX-SIM), and if Π is a NO-instance then $\langle\psi| A |\psi\rangle \geq 3/(4L)$ (i.e. we have a NO instance of ∀-APX-SIM), as desired. ◀

E.1.1 Required lemmas for proof of Theorem 10

We begin by restating a known lemma and corollary.

▶ **Lemma 20** (Kempe, Kitaev, Regev [28]). *Let $H = H_1 + H_2$ be the sum of two Hamiltonians operating on some Hilbert space $\mathcal{H} = \mathcal{S} + \mathcal{S}^\perp$. The Hamiltonian H_1 is such that $\mathcal{S}$ is a zero eigenspace and the eigenvectors in $\mathcal{S}^\perp$ have eigenvalue at least $J > 2\|H_2\|_\infty$. Then,*

$$\lambda(H_2|_{\mathcal{S}}) - \frac{\|H_2\|_\infty^2}{J - 2\|H_2\|_\infty} \leq \lambda(H) \leq \lambda(H_2|_{\mathcal{S}}),$$

where recall $\lambda(H_2|_{\mathcal{S}})$ denotes the smallest eigenvalue of H_2 restricted to space $\mathcal{S}$.

▶ **Corollary 21** ([22]). *Let $H = H_1 + H_2$ be the sum of two Hamiltonians operating on some Hilbert space $\mathcal{H} = \mathcal{S} + \mathcal{S}^\perp$. The Hamiltonian H_1 is such that $\mathcal{S}$ is a zero eigenspace and the eigenvectors in $\mathcal{S}^\perp$ have eigenvalue at least $J > 2\|H_2\|_\infty$. Let $K := \|H_2\|_\infty$. Then, for any $\delta \geq 0$ and vector $|\psi\rangle$ satisfying $\langle\psi| H |\psi\rangle \leq \lambda(H) + \delta$, there exists a $|\psi'\rangle \in \mathcal{S}$ such that*

$$\||\psi\rangle\langle\psi| - |\psi'\rangle\langle\psi'|\|_{\text{tr}} \leq 2\left(\frac{K + \sqrt{K^2 + \delta(J - 2K)}}{J - 2K}\right).$$

We now prove the lemmas required for Theorem 10.

▶ **Lemma 22.** *Assume the notation of Section 6. For $G = \Delta_{\text{in}}H_{\text{in}} + \Delta_{\text{prop}}H_{\text{prop}} + \Delta_{\text{pen}}H_{\text{pen}}$, the following hold:*

1. *For sufficiently large (efficiently computable) polynomials* $\Delta_{\text{in}}, \Delta_{\text{prop}}, \Delta_{\text{pen}}$*, the null space of* G *is the span of all correctly encoded history states, i.e. of the form in Equation (16).*
2. *For any fixed inverse polynomials* δ *and* γ*, there exist efficiently computable polynomials* $\Delta_{\text{in}}, \Delta_{\text{prop}}, \Delta_{\text{pen}}$ *such that for any* $|\psi\rangle$ *attaining* $\langle\psi| G |\psi\rangle \leq \lambda(G) + \delta$*, there exists a correctly encoded history state* $|\psi_{\text{hist}}\rangle$ *such that*

$$\| |\psi\rangle\langle\psi| - |\psi_{\text{hist}}\rangle\langle\psi_{\text{hist}}| \|_{\text{tr}} \leq \gamma.$$

Proof. The analysis of G is more subtle than that of, say, the 5-local Kitaev circuit-to-Hamiltonian construction [31]. The latter required the analysis of two orthogonal subspaces acted on invariantly by the Hamiltonian in question; the span of all correctly encoded history states, and the span of all states with an incorrectly encoded clock register (i.e. illegal configurations). In [26], however, due to the restrictions of encoding in 1D, there are *two* types of illegal configurations which can arise – those which are detectable by local checks, and those which are not – and G does not act invariantly on the spaces of legal and illegal configurations. The soundness analysis of the QMA-hardness construction of [26] (see Section 6 therein, which we follow below) hence independently analyzes *three* types of subspaces which are acted on invariantly by H_{prop}: (1) The span of legal configurations and certain locally detectable illegal configurations, (2) the span of certain other locally detectable illegal configurations, and (3) the span of illegal configurations which are not locally detectable. We shall henceforth refer to these subspaces as S_1, S_2, and S_3, respectively.

Proof of claim 1. This claim is implicit in [26]; we sketch a proof to make it explicit here. Claim 2 of [26] and the subsequent discussion explicitly show that any valid history state is a null state of G. For the reverse containment, Section 6.2 of [26] shows that for sufficiently large polynomials $\Delta_{\text{in}}, \Delta_{\text{prop}}, \Delta_{\text{pen}}$, $\lambda((\Delta_{\text{prop}} H_{\text{prop}} + \Delta_{\text{pen}} H_{\text{pen}})|_{S_3}) \in \Omega(1)$. That $\lambda(G|_{S_2}) \geq \Delta_{\text{pen}}$ follows since H_{pen} is a sum of pairwise commuting projectors. Thus, $\text{Null}(G)$ resides in S_1. Section 6.1 of [26] shows that $\text{Null}(H_{\text{prop}}|_{S_1 \cap \text{Null}(H_{\text{pen}})})$ is spanned by valid history states. We conclude that the span of all valid history states contains $\text{Null}(G)$.

Proof of claim 2. We know from claim 1 that $\text{Null}(G)$ is precisely the span of all correctly encoded history states. Let $\mathcal{C}$ denote the orthogonal complement of $\text{Null}(G)$. Then, we know from the proof of claim 1 that $\lambda(G|_{\mathcal{C} \cap S_2}) \geq \Delta_{\text{pen}} \in \Omega(1)$, and that $\lambda((\Delta_{\text{prop}} H_{\text{prop}} + \Delta_{\text{pen}} H_{\text{pen}})|_{\mathcal{C} \cap S_3}) \in \Omega(1)$. (Here we have used the fact that $S_2 \cup S_3 \subseteq \mathcal{C}$.) Since δ is assumed to be inverse polynomial in n, and since we know from claim 1 that $\lambda(H) \leq 0$, it follows that no vector $|\psi\rangle$ from S_2 or S_3 can attain $\langle\psi| G |\psi\rangle \leq \lambda(G) + \delta$.

We are thus reduced to the case $|\psi\rangle \in S_1$, which we prove using three applications of Corollary 21. (To reduce notation, in the remainder of this proof all operators are implicitly restricted to S_1.) In the first application, let $H_1 = \Delta_{\text{pen}} H_{\text{pen}}$ and $H_2 = \Delta_{\text{in}} H_{\text{in}} + \Delta_{\text{prop}} H_{\text{prop}}$. Suppose $\langle\psi| H_1 + H_2 |\psi\rangle \leq \lambda(H) + \delta$. Then by Lemma 21, there exists a vector $|\psi'\rangle \in \text{Null}(H_{\text{pen}})$ such that

$$\| |\psi\rangle\langle\psi| - |\psi'\rangle\langle\psi'| \|_{\text{tr}} \leq 2 \left(\frac{K_1 + \sqrt{K_1^2 + \delta(J_1 - 2K_1)}}{J_1 - 2K_1} \right) =: 2\gamma_1,$$

for $K_1 := \|H_2\|_\infty$ and $J_1 > 2K_1$. (Note that since $\Delta_{\text{pen}} H_{\text{pen}}$ is a sum of commuting projectors, its smallest non-zero eigenvalue is at least Δ_{pen}, i.e. $J \geq \Delta_{\text{pen}}$.) By the Hölder inequality,

$$|\text{Tr}((H_1 + H_2) |\psi\rangle\langle\psi|) - \text{Tr}((H_1 + H_2) |\psi'\rangle\langle\psi'|)| \leq 2\gamma_1 \|H_1 + H_2\|_\infty =: \epsilon_1. \tag{27}$$

Combining these facts, we have

$$\begin{aligned}\langle\psi'|\,(H_1+H_2)|_{\mathrm{Null}(H_{\mathrm{pen}})}\,|\psi'\rangle &= \langle\psi'|\,(H_1+H_2)\,|\psi'\rangle \\ &\leq \lambda((H_1+H_2)) + \delta + \epsilon_1 \\ &\leq \lambda((H_1+H_2)|_{\mathrm{Null}(H_{\mathrm{pen}})}) + \delta + \epsilon_1 \\ &=: \lambda((H_1+H_2)|_{\mathrm{Null}(H_{\mathrm{pen}})}) + \delta_2, \end{aligned} \tag{28}$$

where the first statement holds since $|\psi'\rangle \in \mathrm{Null}(H_{\mathrm{pen}})$, the second by Equation (27), and the third by the Projection Lemma (this follows directly since projections can only increase the smallest eigenvalue).

We now repeat the process for $H_1 = \Delta_{\mathrm{prop}} H_{\mathrm{prop}}|_{\mathrm{Null}(H_{\mathrm{pen}})}$ and $H_2 = \Delta_{\mathrm{in}} H_{\mathrm{in}}|_{\mathrm{Null}(H_{\mathrm{pen}})}$. The key observation (used also in [26]) is that restricted to $S_1 \cap \mathrm{Null}(H_{\mathrm{pen}})$, H_{prop} is now positive semidefinite, has a 1-dimensional null space spanned by the correct history state (the action of H_{prop} ignores the initial setting of ancilla qubits, including the proof register, which in general leads to multiple correct history states), and its smallest non-zero eigenvalue is at least $1/(2(L+1)^2)$ (recall L is the number of time steps a valid history state sums over). Thus, by Lemma 21, there exists a vector $|\psi''\rangle \in \mathrm{Null}(H_{\mathrm{pen}}) \cap \mathrm{Null}(H_{\mathrm{prop}})$ such that

$$\| |\psi'\rangle\langle\psi'| - |\psi''\rangle\langle\psi''| \|_{\mathrm{tr}} \leq 2\left(\frac{K_2 + \sqrt{K_2^2 + \delta_2(J_2 - 2K_2)}}{J_2 - 2K_2}\right) =: 2\gamma_2,$$

for $K_2 := \|H_2\|_\infty$ and $J_2 > 2K_2$. Note that $J_2 \geq \Delta_{\mathrm{prop}}/(2(L+1)^2)$. By the Hölder inequality,

$$|\mathrm{Tr}((H_1+H_2)\,|\psi'\rangle\langle\psi'|) - \mathrm{Tr}((H_1+H_2)\,|\psi''\rangle\langle\psi''|)| \leq 2\gamma_2 \|H_1+H_2\|_\infty =: \epsilon_2,$$

which yields

$$\begin{aligned}\langle\psi''|\,(H_1+H_2)|_{\mathrm{Null}(H_{\mathrm{prop}})}\,|\psi''\rangle &= \langle\psi''|\,(H_1+H_2)\,|\psi''\rangle \\ &\leq \lambda((H_1+H_2)) + \delta_2 + \epsilon_2 \\ &\leq \lambda((H_1+H_2)|_{\mathrm{Null}(H_{\mathrm{prop}})}) + \delta_2 + \epsilon_2 \\ &=: \lambda((H_1+H_2)|_{\mathrm{Null}(H_{\mathrm{prop}})}) + \delta_3. \end{aligned}$$

Finally, we repeat the process for $H_1 = \Delta_{\mathrm{in}} H_{\mathrm{in}}|_{\mathrm{Null}(H_{\mathrm{pen}})\cap\mathrm{Null}(H_{\mathrm{prop}})}$ and $H_2 = 0$. Since by claim 1 we know the joint null space of $H_{\mathrm{in}}, H_{\mathrm{prop}}, H_{\mathrm{pen}}$ is non-empty, by Lemma 21, there exists a vector $|\psi'''\rangle \in \mathrm{Null}(H_{\mathrm{pen}}) \cap \mathrm{Null}(H_{\mathrm{prop}}) \cap \mathrm{Null}(H_{\mathrm{in}})$ such that

$$\| |\psi''\rangle\langle\psi''| - |\psi'''\rangle\langle\psi'''| \|_{\mathrm{tr}} \leq 2\sqrt{\frac{\delta_3}{J_3}} =: 2\gamma_3,$$

for $J_3 > 0$. Note that $J_3 \geq \Delta_{\mathrm{in}}$ since H_{in} is a sum of commuting projectors. By claim 1, since $|\psi'''\rangle$ is in the joint null space of $H_{\mathrm{in}}, H_{\mathrm{prop}}, H_{\mathrm{pen}}$, it is a correctly encoded history state; denote it $|\psi_{\mathrm{hist}}\rangle$. By the triangle inequality we have

$$\| |\psi\rangle\langle\psi| - |\psi_{\mathrm{hist}}\rangle\langle\psi_{\mathrm{hist}}| \|_{\mathrm{tr}} \leq 2(\gamma_1+\gamma_2+\gamma_3).$$

The claim now follows by observing that all variables involved, i.e. δ_2, δ_3, ϵ_1, ϵ_2, γ_1, γ_2, γ_3, J_1, J_2, J_3, decrease inverse polynomially in (a non-empty subset of) polynomials $\Delta_{\mathrm{in}}, \Delta_{\mathrm{prop}}, \Delta_{\mathrm{pen}}$. Thus, for any desired target accuracy q, we may attain the claim by setting $\Delta_{\mathrm{in}}, \Delta_{\mathrm{prop}}, \Delta_{\mathrm{pen}}$ as sufficiently large polynomials. (Note that this requires upper bounding terms of the form $K_2 := \|H_2\|_\infty$, which is easily done via triangle inequality of the spectral norm and the fact that projections can only decrease maximum eigenvalues.) ◀

▶ **Lemma 23.** *Assume the notation of Section E.1. For all $i \in I$, it holds that*

$$\left| \langle \psi_{\text{hist}} | \blacktriangleright_{x_i} \rangle_{q_i} \right|^2 \geq \frac{1-\epsilon}{L}, \tag{29}$$

where recall q_i is the index of the qudit which encodes the output corresponding to query π_i following the verification phase.

Proof. For clarity, the factor of L^{-1} comes from the L configurations which $|\psi_{\text{hist}}\rangle$ is a sum over. Recall there is a unique configuration in which any given qudit is labeled $\blacktriangleright$, implying all history states $|\psi_{\text{hist}}\rangle$ satisfy

$$\left| \langle \psi_{\text{hist}} | \blacktriangleright_0 \rangle_{q_i} \right|^2 + \left| \langle \psi_{\text{hist}} | \blacktriangleright_1 \rangle_{q_i} \right|^2 = \frac{1}{L}. \tag{30}$$

We prove our claim by contradiction via an exchange argument. Suppose there exists a valid query[8] π_j with correct answer x_j such that

$$\left| \langle \psi_{\text{hist}} | \blacktriangleright_{x_j} \rangle_{q_j} \right|^2 < \frac{1-\epsilon}{L}.$$

Since $|\psi_{\text{hist}}\rangle$ is a correctly encoded history state, we claim π_j must be a YES-instance. For if π_j were a NO-instance, then all simulations of V_j (on any possible proof) output NO with probability at least $1-p$. Thus, $|\psi_{\text{hist}}\rangle$ always encodes an output qubit such that

$$\left| \langle \psi_{\text{hist}} | \blacktriangleright_0 \rangle_{q_j} \right|^2 \geq \frac{1-p}{L} \geq \frac{1-\epsilon}{L},$$

which would contradict our supposition.

Given that π_j is a YES-instance, we have that $\left| \langle \psi_{\text{hist}} | \blacktriangleright_1 \rangle_{q_j} \right|^2 \leq (1-\epsilon)/L$, and so by Equation (30), $\langle \psi_{\text{hist}} | H_{\text{out},j} | \psi_{\text{hist}} \rangle \geq \epsilon^2/L$. Further, since π_j is a YES-instance, there exists a QMA proof $|\omega\rangle$ which causes V_j to output YES with probability at least $1-p$. By exchanging the QMA proof which $|\psi_{\text{hist}}\rangle$ encodes for circuit V_j with the proof $|\omega\rangle$, we obtain a new history state $|\psi'_{\text{hist}}\rangle$ which satisfies

$$\left| \langle \psi'_{\text{hist}} | \blacktriangleright_1 \rangle_{q_j} \right|^2 \geq \frac{1-p}{L},$$

and so $\langle \psi'_{\text{hist}} | H_{\text{out},j} | \psi'_{\text{hist}} \rangle \leq p\epsilon/L$. Hence,

$$\langle \psi_{\text{hist}} | H_{\text{out},j} | \psi_{\text{hist}} \rangle - \langle \psi'_{\text{hist}} | H_{\text{out},j} | \psi'_{\text{hist}} \rangle \geq \frac{(\epsilon-p)\epsilon}{L}, \tag{31}$$

i.e. flipping the incorrect query answer saves a non-trivial energy penalty on $H_{\text{out},j}$.

We now use this to obtain the desired contradiction. Recall that $H = G + H_{\text{out}}$. We make two observations: First, because all the QMA queries are made in parallel, flipping the answer to query π_j does not affect the other queries the P machine makes or the answers it receives. Thus, $|\psi_{\text{hist}}\rangle$ and $|\psi'_{\text{hist}}\rangle$ obtain the same energy on all terms of H_{out} other than $H_{\text{out},j}$, and Equation (31) holds for H_{out} in place of $H_{\text{out},j}$. (Analyzing adaptive queries, rather than parallel, would require that penalties for later queries be carefully weighted less than penalties for earlier queries [3], leading to a significantly more involved analysis.)

[8] If all queries are invalid, then Lemma 23 holds vacuously.

Second, both $|\psi_{\text{hist}}\rangle$ and $|\psi'_{\text{hist}}\rangle$ are null states of G, and so we may substitute H for H_{out}, yielding

$$\langle\psi_{\text{hist}}| H |\psi_{\text{hist}}\rangle - \langle\psi'_{\text{hist}}| H |\psi'_{\text{hist}}\rangle \geq \frac{(\epsilon - p)\epsilon}{L}. \tag{32}$$

Now, recall from Equation (24) that $\langle\psi_{\text{hist}}| H |\psi_{\text{hist}}\rangle \leq \lambda(H) + \delta + m\epsilon\gamma$. Since δ and γ are inverse polynomials which (by Lemma 22) we are free to choose as needed (the choice of δ and γ, in turn, will mandate the choices of $\Delta_{\text{in}}, \Delta_{\text{prop}}, \Delta_{\text{pen}}$ via Lemma 22), we set $\delta = \gamma = 1/(256m^2L)$ (where recall L and m are fixed polynomials in n). These choices of δ, γ satisfy $\delta + m\epsilon\gamma < (\epsilon - p)\epsilon/L$, which combined with Equation (32) gives that $\langle\psi_{\text{hist}}| H |\psi_{\text{hist}}\rangle > \lambda(H) + \delta + m\epsilon\gamma$, i.e. $|\psi_{\text{hist}}\rangle$ could not have been close to the ground state energy of H. Hence, we have a contradiction, completing the proof. ◀

Finally, we require a known quantum analogue of the union bound for commuting operators (see, e.g. [38]). Generalizations to *non-commuting* projectors are given in [43, 17, 38].

▶ **Lemma 24** (Commutative Quantum Union Bound). *Let $\{P_i\}_{i=1}^m$ be a set of pairwise commuting projectors, each satisfying $0 \preceq P_i \preceq I$. Then for any quantum state ρ,*

$$1 - \text{Tr}(\Pi_m \cdots P_1 \rho P_1 \cdots \Pi_m) \leq \sum_{i=1}^{m} \text{Tr}((I - P_i)\rho).$$

The simple proof of Lemma 24 is given below for completeness.

Proof of Lemma 24. We proceed by induction on m. The case of $m = 1$ is trivial. Consider $m > 1$. Since the P_i pairwise commute, $\text{Tr}(P_m \cdots P_1 \rho P_1 \cdots P_m) = \text{Tr}(P_m \cdots P_1 \rho) := \text{Tr}(P_m M \rho)$ for brevity, and M is a projector. Then,

$$\begin{aligned} 1 - \text{Tr}(P_m M\rho) = \quad & \text{Tr}((I - P_m)M\rho) + \text{Tr}(P_m(I - M)\rho) + \text{Tr}((I - P_m)(I - M)\rho) \\ = \quad & \text{Tr}((I - P_m)\rho) + \text{Tr}((I - M)\rho) - \text{Tr}((I - P_m)(I - M)\rho) \\ \leq \quad & \text{Tr}((I - P_m)\rho) + \text{Tr}((I - M)\rho), \end{aligned}$$

where the second equality holds since $\text{Tr}((I - P_m)(I - M)\rho)$ equals

$$\begin{aligned} &\text{Tr}((I - P_m)\rho) + \text{Tr}((I - M)\rho) - (\text{Tr}((I - P_m)M\rho) + \\ &\text{Tr}(P_m(I - M)\rho) + \text{Tr}((I - P_m)(I - M)\rho)). \end{aligned}$$

Applying the induction hypothesis completes the proof. ◀

F General simulations

In this section we will give a full proof of Lemma 15 and show that *any* efficient simulation will preserve hardness of ∀-APX-SIM, not just the special case considered in Definition 14. To state the full definition of simulation, we must first introduce the notion of an encoding.

▶ **Definition 25** ([13]). *We say a map $\mathcal{E} : \mathcal{B}(\mathcal{H}) \to \mathcal{B}(\mathcal{H}')$ is an encoding if it is of the form*

$$\mathcal{E}(M) = V(M \otimes P + \overline{M} \otimes Q)V^\dagger$$

where $\overline{M}$ denotes the complex conjugate of M, P and Q are orthogonal projectors (i.e. $PQ = 0$) on an ancilla space E; and V is an isometry $V : \mathcal{H} \otimes E \to \mathcal{H}'$.

When $\mathcal{H}$ is a many body system with a decomposition $\mathcal{H} = \bigotimes_{i=1}^n \mathcal{H}_i$, we say $\mathcal{E}$ is a local encoding if $E = \bigotimes_{i=1}^n E_i$ such that:

- $V = \bigotimes_{i=1}^{n} V_i$ *where each* V_i *acts on* $\mathcal{H}_i \otimes E_i$.
- *for each* i, *there exist orthogonal projectors* P_{E_i} *and* Q_{E_i} *on* E *which act non-trivially only on* E_i, *and satisfy* $PP_{E_i} = P$ *and* $QQ_{E_i} = Q$.

We are now ready to give the full definition of simulation.

▶ **Definition 26** ([13]). *We say that* H' *is a* (Δ, η, ϵ)*-simulation of* H *if there exists a local encoding* $\mathcal{E}(M) = V(M \otimes P + \overline{M} \otimes Q)V^\dagger$ *such that:*

1. *There exists an isometry* $\widetilde{V} : \mathcal{H} \otimes E \to \mathcal{H}'$ *such that* $\|\widetilde{V} - V\| \leq \eta$; *and that the encoding* $\widetilde{\mathcal{E}}(M) = \widetilde{V}(M \otimes P + \overline{M} \otimes Q)\widetilde{V}^\dagger$ *satisfies* $\widetilde{\mathcal{E}}(I) = P_{\leq \Delta(H')}$.
2. $\|H'_{\leq \Delta} - \widetilde{\mathcal{E}}(H)\| \leq \epsilon$.

We say that a family $\mathcal{F}'$ *of Hamiltonians can simulate a family* $\mathcal{F}$ *of Hamiltonians if, for any* $H \in \mathcal{F}$ *and any* $\eta, \epsilon > 0$ *and* $\Delta \geq \Delta_0$ *(for some* $\Delta_0 > 0$*), there exists* $H' \in \mathcal{F}'$ *such that* H' *is a* (Δ, η, ϵ)*-simulation of* H. *We say that the simulation is efficient if, in addition, for* H *acting on* n *qudits,* $\|H'\| = \mathrm{poly}(n, 1/\eta, 1/\epsilon, \Delta)$; H' *and* $\{ V_i \}$ *are efficiently computable given* H, Δ, η *and* ϵ; *and each local isometry* V_i *in the decomposition* $V = \bigotimes_i V_i$ *maps to* $O(1)$ *qudits.*

We note that Definition 14 is just the special case of Definition 26 where $\mathcal{E}(M) = VMV^\dagger$. We are now ready to restate and prove Lemma 15.

▶ **Lemma 15** (Simulations preserve hardness of $\forall$-APX-SIM). *Let* $\mathcal{F}$ *be a family of Hamiltonians which can be efficiently simulated by another family* $\mathcal{F}'$. *Then* $\mathcal{F}$-$\forall$-$APXSIM$ *reduces to* $\mathcal{F}'$-$\forall$-$APXSIM$.

Proof. For brevity, let $P_{\leq \Delta} := P_{\leq \Delta(H')}$. Let $\rho' = |\psi'\rangle\langle\psi'|$ be a state on $\mathcal{H}'$ such that $\langle\psi'| H' |\psi'\rangle \leq \delta'$ and let $\widetilde{\rho} = P_{\leq \Delta}\rho' P_{\leq \Delta} / \operatorname{Tr}(P_{\leq \Delta}\rho')$, so that by Lemma 19, we have $\|\rho' - \widetilde{\rho}\|_1 \leq 2\sqrt{\frac{\delta'}{\Delta - \lambda(H')}}$.

Since $P_{\leq \Delta}$ commutes with H', we have

$$\operatorname{Tr}(H'\rho') = \operatorname{Tr}(H' P_{\leq \Delta}\rho' P_{\leq \Delta}) + \operatorname{Tr}(H'(I - P_{\leq \Delta})\rho'(I - P_{\leq \Delta})) \tag{33}$$

$$= p \operatorname{Tr}(H'\widetilde{\rho}) + (1-p)\operatorname{Tr}(H'\widetilde{\rho}^\perp) \geq \operatorname{Tr}(H'\widetilde{\rho}), \tag{34}$$

where $p = \operatorname{Tr}(P_{\leq \Delta}\rho')$, $\widetilde{\rho}^\perp = (I - P_{\leq \Delta})\rho'(I - P_{\leq \Delta}) / \operatorname{Tr}((I - P_{\leq \Delta})\rho')$, and the final inequality follows because $\operatorname{Tr}(H'\widetilde{\rho}^\perp) \geq \Delta \geq \operatorname{Tr}(H'\widetilde{\rho})$.

Now let

$$\rho = \operatorname{Tr}_E\left(\widetilde{V}^\dagger \widetilde{\rho} \widetilde{V}(I \otimes P)\right) + \operatorname{Tr}_E\left(\overline{\widetilde{V}^\dagger \widetilde{\rho} \widetilde{V}(I \otimes Q)}\right)$$

and note that for any operator A on $\mathcal{H}$, we have $\operatorname{Tr}(\widetilde{\mathcal{E}}(A)\widetilde{\rho})$ equals

$$\operatorname{Tr}\left(\widetilde{V}(A \otimes P + \overline{A} \otimes Q)\widetilde{V}^\dagger \widetilde{\rho}\right) = \operatorname{Tr}\left(A \otimes P \widetilde{V}^\dagger \widetilde{\rho} \widetilde{V}\right) + \operatorname{Tr}\left(\overline{A} \otimes Q \widetilde{V}^\dagger \widetilde{\rho} \widetilde{V}\right) = \operatorname{Tr}(A\rho).$$

Therefore,

$$\begin{aligned}
\operatorname{Tr}(H\rho) &= \operatorname{Tr}(\widetilde{\mathcal{E}}(H)\widetilde{\rho}) \\
&\leq \operatorname{Tr}(H'\widetilde{\rho}) + \|H'_{\leq \Delta} - \widetilde{\mathcal{E}}(H)\| \\
&\leq \operatorname{Tr}(H'\rho') + \epsilon \\
&\leq \lambda(H') + \delta' + \epsilon \\
&\leq \lambda(H) + \delta' + 2\epsilon,
\end{aligned}$$

where the second inequality follows from Equation (34) and the last inequality from Lemma 27 of [13], which roughly states that eigenvalues are preserved up to additive error ϵ in a simulation.

At this point the proof diverges from the simpler case because ρ may be a mixed state, even when $\rho' = |\psi'\rangle\langle\psi'|$ is pure. Despite having a bound on $\mathrm{Tr}(H\rho)$, this bound may not hold for all pure states in the spectral decomposition of ρ. Let $\rho_\delta = P_\delta \rho P_\delta / \mathrm{Tr}(P_\delta)$, where P_δ is the projector onto eigenvectors of H with energy less than δ. By Lemma 19, $\|\rho - \rho_\delta\|_1 \leq 2\sqrt{\frac{\delta'+2\epsilon}{\delta}}$. We will use the spectral decomposition of $\rho_\delta = \sum_i \mu_i |\phi_i\rangle\langle\phi_i|$ where the $|\phi_i\rangle$ are orthogonal states with energy $\langle\phi_i| H |\phi_i\rangle \leq \lambda(H) + \delta$ and thus, for observable A given as part of of $\mathcal{F}$-$\forall$-APXSIM input,

$$\mathrm{Tr}(A\rho_\delta) = \sum_i \mu_i \langle\phi_i| A |\phi_i\rangle \quad \begin{cases} \leq a \text{ in a YES instance} \\ \geq b \text{ in a NO instance.} \end{cases}$$

Let $U = V\widetilde{V}^\dagger$, which satisfies $U\widetilde{\mathcal{E}}(A) = \mathcal{E}(A)U$ for any A, and so $\mathcal{E}(I)U\widetilde{\rho}U^\dagger = U\widetilde{\mathcal{E}}(I)\widetilde{\rho}U^\dagger = U\widetilde{\rho}U^\dagger$. Now we need to choose A' such that $A'\mathcal{E}(I) = \mathcal{E}(A)$. (Two notes: First, $\mathcal{E}(I) \neq I$ necessarily, as P and Q need not sum to identity. Second, setting $A' = \mathcal{E}(A)$ is not necessarily desirable, as P and Q may be non-local projectors.) For example if $A = B_i \otimes I$, let $A' = V_i(B_i \otimes P_{E_i} + \overline{B_i} \otimes Q_{E_i})V_i^\dagger \otimes I$. We note that the locality of A' depends on the number of qudits which V_i maps to, which is $O(1)$ by the definition of efficient simulation. Then

$$\mathrm{Tr}(A\rho) = \mathrm{Tr}\left(\widetilde{\mathcal{E}}(A)\widetilde{\rho}\right) = \mathrm{Tr}\left(\mathcal{E}(A)U\widetilde{\rho}U^\dagger\right) = \mathrm{Tr}(A'\mathcal{E}(I)U\widetilde{\rho}U^\dagger) = \mathrm{Tr}(A'U\widetilde{\rho}U^\dagger)$$

and therefore

$$\begin{aligned} |\mathrm{Tr}(A'\rho') - \mathrm{Tr}(A\rho_\delta)| &\leq |\mathrm{Tr}(A'\rho') - \mathrm{Tr}(A'U\widetilde{\rho}U^\dagger)| + |\mathrm{Tr}(A\rho) - \mathrm{Tr}(A\rho_\delta)| \\ &\leq \|A'\|\left(\|\rho' - \widetilde{\rho}\|_1 + \|\widetilde{\rho} - U\widetilde{\rho}U^\dagger\|_1\right) + \|A\|\|\rho - \rho_\delta\|_1 \\ &\leq \|A\|\left(2\sqrt{\frac{\delta'}{\Delta - \lambda(H')}} + 2\eta + 2\sqrt{\frac{\delta'+2\epsilon}{\delta}}\right), \end{aligned}$$

We note that $\|\widetilde{\rho} - U\widetilde{\rho}U^\dagger\|_1 \leq 2\eta$ follows from $\|U - \widetilde{V}\widetilde{V}^\dagger\| \leq \eta$, and that $\widetilde{V}\widetilde{V}^\dagger\widetilde{\rho} = P_{\leq\Delta}\widetilde{\rho} = \widetilde{\rho}$. Therefore we just need to choose $\Delta, \epsilon, \eta, \delta'$ such that this is less than $(b-a)/3$ and then set $a' = a + (b-a)/3$ and $b' = b - (b-a)/3$. ◀

Secret Key Agreement from Correlated Data, with No Prior Information

Marius Zimand
Towson University, MD, USA
http://orion.towson.edu/~mzimand/
mzimand@towson.edu

Abstract

A fundamental question that has been studied in cryptography and in information theory is whether two parties can communicate confidentially using exclusively an open channel. We consider the model in which the two parties hold inputs that are correlated in a certain sense. This model has been studied extensively in information theory, and communication protocols have been designed which exploit the correlation to extract from the inputs a shared secret key. However, all the existing protocols are not universal in the sense that they require that the two parties also know some attributes of the correlation. In other words, they require that each party knows something about the other party's input. We present a protocol that does not require any prior additional information. It uses space-bounded Kolmogorov complexity to measure correlation and it allows the two legal parties to obtain a common key that looks random to an eavesdropper that observes the communication and is restricted to use a bounded amount of space for the attack. Thus the protocol achieves complexity-theoretical security, but it does not use any unproven result from computational complexity. On the negative side, the protocol is not efficient in the sense that the computation of the two legal parties uses more space than the space allowed to the adversary.

2012 ACM Subject Classification Theory of computation → Models of computation; Mathematics of computing → Information theory; Security and privacy → Information-theoretic techniques

Keywords and phrases secret key agreement, Kolmogorov complexity, extractors

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.21

Funding *Marius Zimand*: The author has been supported in part by the National Science Foundation through grant CCF 1811729.

Acknowledgements I want to thank Andrei Romashchenko for useful discussions. I also thank the anonymous referees for their observations which have helped me correct some errors and improve the presentation.

1 Introduction

The goal of a secret key agreement protocol is to allow two parties that communicate through a public channel to obtain a shared string that is secret in some reasonable sense (e.g., information-theoretical, complexity-theoretical, or some other sense) to anyone that has observed the communication. There are some well-known such protocols, such as the Diffie-Hellman protocol, or various public-key cryptosystems, that are efficient and used in the real world. However, they have the disadvantage of relying on some unproven hardness conjectures in computational complexity. Another setting is to assume that the two parties hold at the beginning of the protocol pieces of information that have a certain degree of correlation. Then, in some circumstances, it is possible to compute the shared secret key without any unproven assumption. For a simple illustration, suppose that Alice holds a line L in the 2-dimensional affine space, and Bob holds a point P which lies on L. Then Alice sends Bob the slope of L, after which Bob, *knowing that his P is on L*, can compute the intercept of L. Now, both Alice and Bob have the intercept of L, which they can use as a secret key, because the adversary has only seen the slope, which is independent of the intercept.

In this paper, we consider the latter type of secret key agreement protocols. Thus, Alice starts with a string x, Bob starts with y, and, after several rounds of interacting via messages exchanged over a public channel, they obtain at the end of the protocol a common secret key, that is a string z which is random conditioned by the transcript of the protocol. The protocol is probabilistically computable, i.e., there exists a probabilistic algorithm so that Alice computes each of her messages by running the algorithm on her input and on the messages that she has received from Bob so far, and Bob computes his messages similarly. As in the above example, if x and y are *correlated* in some way, one can hope to use the information that is common to these strings to extract with high probability a secret key.

The study of this scenario has a long history in Information Theory and the common flavor of the results is that for many interpretations of "correlated," secret key agreement is possible. Leung [7], Bennett et al. [3], Maurer [8], Ahlswede and Csiszár [1] have started an extensive research line dedicated to the case when x and y are generated by a stochastic process, whose properties describe their correlation (see the survey of Narayan and Tyagi [11]). Recently, Romashchenko and Zimand [13] have studied this problem in the very general framework of Algorithmic Information Theory using Kolmogorov complexity to gauge correlation without using any generative model for the provenance of x and y. [1]

In all these works, Alice and Bob possess at the beginning of the protocol, in addition to x and y, some information about how these strings are correlated. For instance, in the above example, Bob knows that the point P is on the line L. In the scenarios based on generative models, Alice and Bob know various attributes of the joint distribution of the two random variables (X, Y) which describe the stochastic process that generates the pair (x, y), such as entropy, ergodic properties, etc. In the algorithmic information theory setting used in [13], Alice and Bob know the complexity profile of (x, y), which is the 3-tuple $(C(x), C(y), C(x, y))$, where $C(\cdot)$ denotes Kolmogorov complexity. (Throughout this paper, $C(x)$, called the Kolmogorov complexity of x or the minimal description length of x, is the length of a shortest program that when executed by a universal Turing machine prints x.)

Can Alice and Bob agree on a secret key without any additional prior information? A disclaimer: This is not really a problem relevant for cryptography, because the protocols are not efficient. We rather view it as a question about the fundamental limits of information processing and communication. The challenge is that Alice and Bob have to detect a type of correlation of their inputs through rounds of communication without leaking too much information to the eavesdropper, so that they can compute a shared secret key of reasonable length.

What is a reasonable length of the secret key? The relevant parameter that comes into play is the *mutual information* of x and y, denoted $I(x : y)$, which intuitively represents the amount of information that is shared by x and y. In case we use Kolmogorov complexity to measure the amount of information, $I(x : y)$ is defined as $C(x) + C(y) - C(x, y)$, and, up to logarithmic precision, is also equal to $C(x) - C(x \mid y)$ and to $C(y) - C(y \mid x)$. It is shown in [13] (extending a classical result from [1] which is valid for inputs generated by memoryless processes, and which is using Shannon entropy to measure information), that no computable protocol (even probabilistic) can obtain a shared secret key longer than the mutual information of the inputs x and y. On the other hand, a protocol is presented in [13] that with high probability produces a shared secret of length $I(x : y)$ (up to logarithmic

[1] We point out that unlike the protocols based on hardness assumption (e.g., Diffie-Hellman protocol) which achieve complexity-theoretic security and are efficient, the protocols in the works above achieve information-theoretic security but do not run in polynomial time.

precison), provided, as mentioned above, the two parties know the complexity profile of the inputs. Thus, the above discussion suggests that it is natural to aim for a shared secret key whose length is equal to the mutual information of the inputs, for some concept of information that measures the detectable correlation.

Our contribution. We identify space-bounded Kolmogorov complexity as a concept of information that allows secret key agreement without any prior information or special setup (e.g., shared randomness, special extra channel) between the two parties. The space-bounded Kolmogorov complexity with space bound S of a string x, denoted $C^S(x)$, is similar to standard Kolomogorov complexity except that the universal Turing machine is restricted to use at most S cells on the working tape (see Section 1.1 for the formal definition). We show that the correlation induced by space-bounded Kolmogorov complexity can be determined without revealing much about x and y, which, in turn, allows the parties to compute a common secret key.

The protocols that we design produce a key z that is random given the transcript in the sense of space-bounded Kolmogorov complexity, where the transcript is the set of messages sent by Alice and Bob. Formally, we require that $C^S(z \mid \text{transcript})$ is close to the length of z (denoted $|z|$), for some space bound S. In other words, an eavesdropper which is bounded to use space S and knows the transcript, needs essentially $|z|$ bits to find the secret key z, which is the same as if she did not know the transcript. If $C^S(z \mid \text{transcript}) \geq |z| - \Delta$, we say that Δ is the randomness deficiency of z with respect to the transcript, and thus we want to obtain z with small randomness deficiency. We also want the length of z to be close to the mutual information of x and y, which in the case of space-bounded Kolmogorov complexity is defined as $C^{S_1}(x) - C^{S_2}(x \mid y)$ for space bounds S_1 and S_2. We next present our results.

We recall that a function $S(n)$ is fully space constructible if there is a Turing machine M that uses exactly $S(n)$ cells for every natural number n and for every input of length n.

▶ **Theorem 1.** *Let S be any fully space constructible function.such that $S(n) \geq n$.*

There is a randomized protocol that allows Alice on input x (an n-bit string) and Bob on input y (of arbitrary length) to obtain with probability $(1-\epsilon)$ a common string z such that

(i) $|z| \geq^+ C^{\lambda_1 \cdot S(n)}(x) - C^{\lambda_2 \cdot S(n)}(x \mid y)$,

(ii) $C^{S(n)}(z \mid \textit{transcript}) \geq^+ |z| - \Delta$,

where $\Delta \leq C^{\lambda_3 \cdot S(n)}(x) - C^{\lambda_4 \cdot S(n)}(x)$, $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are constants that depend only on the universal Turing machine, and $\geq^+$ hides a loss of precision bounded by $O(\log(n/\epsilon))$.

Note: The notation $a \geq^+ b$ means that $a \geq b - \alpha$, where α is the specified loss of precision.

The communication complexity of the protocol in the above theorem is $n^2 + O(n \log(1/\epsilon)) + C^{S(n)}(x \mid y)$, which is very large. The protocol in [13] (in which the parties also hold the complexity profile of (x, y)) has communication complexity roughly $C(x \mid y)$, which is shown to be optimal. Thus, in our case, it would be desirable to have a protocol with communication complexity close to $C^{S(n)}(x \mid y)$. The protocol in the next theorem has *information complexity* $C^{S(n)}(x \mid y)$ plus a polylogarithmic term and communication complexity $2C^{S(n)}(x \mid y)$ plus a polylogarithmic term.

▶ **Theorem 2** (Main Result). *Let S be a fully space constructible function such that $S(n) \geq p_0(n)$, where $p_0(n)$ is a fixed polynomial that only depends on the universal Turing machine.*

There is a randomized protocol that allows Alice on input x (an n-bit string) and Bob on input y (of arbitrary length) to obtain with probability $(1-\epsilon)$ a common string z such that

(i) $|z| \geq^+ C^{S(n)}(x) - C^{S(n)}(x \mid y)$,

(ii) $C^{S(n)}(z \mid \textit{transcript}) \geq |z| - \Delta$,

where $\Delta \leq^+ C^{\lambda^{-1}S(n)}(x) - C^{\lambda \cdot S(n)}(x)$, λ *is a constant that depends only on the universal Turing machine, and* $\geq^+$ *hides a loss of precision bounded by* $O(\log^3(n/\epsilon))$.
Furthermore, the length of the transcript is bounded by $2C^{S(n)}(x \mid y) + O(\log^3(n/\epsilon))$.

In the above theorems, the key z has Δ, the randomness deficiency conditioned by the transcript, bounded by $C^{S'(n)}(x) - C^{S''(n)}(x)$, where S' and S'' differ by a multiplicative constant. Thus, intuitively, Δ is small. A particularly favorable case is when x is a shallow string. A string x is S-shallow if $C^{S(n)}(x) =^+ C(x)$, i.e., if $S(n)$ is enough space to allow the construction of x from a description which is close to a shortest description. For every space bound S, most strings are S-shallow and in case x is such a string then $\Delta =^+ 0$.

1.1 Prerequisites

The S-space bounded Kolmogorov complexity of x conditioned by y with respect to a Turing machine M, denoted $C^S_M(x \mid y)$, is defined by

$$C^S_M(x \mid y) = \min\{|p| \mid M(p, y) = x \text{ and } M \text{ uses at most } S \text{ cells.}\}$$

In the case of space-bounded Kolmogorov complexity, simulation by the universal machine incurs a constant blow-up in space usage. More precisely, there exists a universal Turing machine U and a constant $\gamma > 1$ such that for any space bound S, for any Turing machine M and for all strings x, y,

$$C^{\gamma S}_U(x \mid y) \leq C^S_M(x \mid y) + O(1).$$

As usual, we fix a universal machine U, and denote more simply $C^S(\cdot)$ instead of $C^S_U(\cdot)$. Also, in case the string y used in the condition is the empty string, we drop the condition in the notation.

The *chain rules* for space-bounded Kolmogorov complexity are as follows: There exists a constant $\gamma > 1$ such that for any space bound S, it holds that:

$$\begin{array}{lcl} C^{\gamma S}(x, y) & \leq & C^S(x) + C^S(y \mid x) + O(\log(|x| + |y|)), \\ C^S(x, y) & \geq & C^{\gamma S}(x) + C^{\gamma S}(y \mid x) + O(\log(|x| + |y|)). \end{array} \tag{1}$$

To simplify the writing of expressions, we sometimes use the notation $CS^{(i)}(\dots)$ instead of $C^{\gamma^i \cdot S}(\dots)$, where S is a space bound and γ (or sometimes λ) is a constant which is clearly defined in the context. For instance, the last inequality will be written as $CS^{(0)}(x, y) \geq CS^{(1)}(x) + CS^{(1)}(y \mid x) + O(\log(|x| + |y|)$.

2 Outline of the proofs

The proofs of both Theorem 1 and Theorem 2 have the same structure. We present an outline, in which, for simplicity, we skip some technical details and ignore small factors in the quantitative relations. Recall that initially Alice holds x and Bob holds y. The protocols in both proofs have two phases: (1) *Information reconciliation*, in which Alice communicates x to Bob by sending him just enough information that allows him to obtain x given his y, and (2) *Secret key construction*, in which, separately, Alice and Bob compute the secret key z. All the communication happens in the Information reconciliation phase.

Phase 1 (Information reconciliation): First, Alice and Bob agree on a space bound $S = S(n)$. Next, Alice sends Bob a randomized hash function h. The goal is for Alice to send Bob, as a fingerprint, some prefix of $h(x)$ that permits Bob to construct Alice's string x using

the fingerprint and his string y. To avoid sending more information than what Bob needs, Alice sends the bits of $h(x)$ sequentially *one bit per round*. At each round, Bob attempts to construct x by checking if the fingerprint of some string in a set of possible candidates matches the prefix of $h(x)$ that he has received so far. More precisely, at each round j, the candidates are those strings whose S-space-bounded complexity conditioned by y is at most j. If Bob finds a string among these candidates with a fingerprint that matches the prefix of $h(x)$ sent so far by Alice, he believes that he has found x, tells Alice to stop sending further bits by sending her "1", and Phase 1 stops. Otherwise, he tells Alice that he needs more bits by sending her "0" (in which case Alice sends in the next round the next bit of $h(x)$).

Let p be the prefix of $h(x)$ that Alice sends to Bob during the entire Phase 1. Then, with high probability, at the end of Phase 1,

1. Bob has x,
2. $|p| \leq C^S(x \mid y)$, because we show that Bob can reconstruct x by round $j = C^S(x \mid y)$. In the proof of Theorem 1, a random matrix H also appears in the condition (as we explain below), but this has little impact, because H is a random.

Phase 2 (Secret key construction): After Phase 1, both Alice and Bob have x (with high probability). They both compute the shared secret key z by exhaustive searching a minimal length program of x given p in space S. So, from p and z, it is possible to construct x. It follows that z and p are independent, because otherwise z would not be minimal. But then z and the transcript of the protocol are also almost independent, because the transcript consists of p and the sequence "$0 \ldots 01$" sent by Bob, and the complexity of $0 \ldots 01$ is low (at most $\log n + O(1)$). Thus, z is a secret key. Let us now estimate the length of z. Since x can be constructed from p and z in space S, it follows that $C^S(x) \leq |p| + |z|$, and thus the length of z is at least $C^S(x) - |p|$, which, by the above bound of $|p|$, is at least $C^S(x) - C^S(x \mid y)$, which is the mutual information of x and y in the framework of space-bounded Kolmogorov complexity.

The main technical issue is finding the hash function that is used in Phase 1. In the proof of Theorem 1, this is just a random linear function given by a random matrix H, chosen by Alice. H is roughly n^2 bits long, and Alice needs to also send H to Bob. This is the reason the communication complexity is large. Also, the information-theoretical considerations in Phase 2, are somewhat more delicate, because we need to take into account H. To reduce the communication complexity, one has to use a shorter hash function. One idea is to use Newman's theorem from communication complexity, in which H is chosen from a smaller sample space. But the sample space needs to be effectively constructed, and the obvious way to do this leads to a loss of precision that is logarithmic in both the length of x and of y, which can be very damaging in case y is much longer than x. In Theorem 2, we use for hashing an explicit extractor of Raz, Reingold, and Vadhan [12], which has the special property that if we take prefixes of the output, the extractor property is preserved. These type of extractors, called *prefix extractors*, allow much more communication-efficient hashing, in the sense that Alice does not need to send the hashing function to Bob, at the cost of making Bob's reconstruction of x more complicated.

In our technical approach, we were inspired by several papers. Muchnik [9] has introduced bipartite graphs similar to extractors and has used for a certain type of information reconciliation concepts similar to what we call *heavy nodes* and *poor nodes* in the proof of Theorem 2. Prefix extractors have been used for information reconciliation in [10] and [15], and the first paper analyzes the case of space-bounded Kolmogorov complexity. The application to secret-key agreement is a novel contribution of this paper. Some of the information-theoretical

estimations are similar to those in [13]. The idea of sending pieces of a fingerprint in several rounds for the problem of information reconciliation (similarly to our Phase 1) has been used before in [4, 6], and, the closest to our approach, in [5], where they study the communication complexity of this problem in terms of the Kolmogorov complexity of the two inputs. There is however a significant difference with the information reconcilation phase in our main result, because, as standard in communication complexity, the protocol in [5] is not computable, and therefore they can use random hash functions for fingerprinting.

3 Proof of Theorem 1

Phase 1: Information reconciliation. Before sending the first message, Alice takes a random matrix H with entries in the finite field GF[2], with $(n + \log(1/\delta))$ rows and n columns, where n is the length of x and $\delta = \epsilon/2n$. The random matrix H defines a random linear function h mapping n bit strings to $n + \log(1/\delta)$ bit strings (viewed as vectors over GF[2]), given by the expression $h(v) = H \cdot v$.

In Round 0, Alice sends to Bob n, H, and the first $1 + \log(1/\delta)$ bits of $h(x)$.

Then in each subsequent round, Alice sends to Bob the next bit of $h(x)$ till Bob announces that he does not need any additional bits. Thus, at round $j \geq 1$, Bob has received the first $(j + 1) + \log(1/\delta)$ bits of $h(x)$, a string which we denote p_j. Bob checks if there is a string u in $B_j = \{u \in \{0,1\}^n \mid C^{S(n)}(u \mid y, H) \leq j\}$ such that p_j is a prefix of $h(u)$. If there is such a string u, he believes that u is x, and announces that he does not need any extra bits and the information reconciliation stops here. If there is no such string u, Bob announces that he needs more bits and the protocol proceeds with the next round.

Bob may be wrong at round j, if there is a string u different from x in B_j such that the prefixes of length $(j + 1) + \log(1/\delta)$ of $h(u)$ and $h(x)$ coincide. For an arbitrary string $u \neq x$, the probability that $h(u)$ and $h(x)$ agree in the first $(j + 1) + \log(1/\delta)$ bits is $2^{-((j+1)+\log(1/\delta))} = \delta/2^{j+1}$. Since B_j has less than 2^{j+1} elements, by the union bound, the probability that Bob is wrong at round j is less than δ.

Let $k = C^{S(n)}(x \mid y, H)$. Let $\mathcal{E}$ be the event that Bob is wrong at one of the rounds $1, \ldots, k$. $\mathcal{E}$ has probability at most $k\delta \leq (n + c)\delta \leq 2n\delta = \epsilon$. Conditioned by $\mathcal{E}$ not being true, the protocol reaches round k, when Bob finds x. Thus, with probability $1 - \epsilon$, at the end of round k, Bob has obtained x, and the string $p := p_k$ is a program for x given y and H in space $\gamma' \cdot S(n)$, for some constant γ', and p has length $C^{S(n)}(x \mid y, H)$.

Phase 2: Secret key construction. By exhaustive search, Alice and (separately) Bob find z, the first program of x given p and H in space S. We show that z satisfies the conclusion of the theorem.

We denote $S := S(n)$ and we let $\geq^+$ hide a loss of precision of $O(\log(n/\epsilon))$. Recall that we use the notation $CS^{(i)}(\ldots)$ in lieu of $C^{\lambda^i \cdot S}(\ldots)$, where λ is here the maximum between the above γ' and γ (the constant from the chain rule (1)).

First, we notice that, with high probability, conditioning by a random H does not decrease complexities by too much.

▷ Claim 3. For every space bound $\mathcal{S}$, for every n-bit string u, for every string v, if H is chosen uniformly at random independent of u and v, we have

$$CS^{(0)}(u \mid v, H) \geq^+ CS^{(2)}(u \mid v) \text{ with probability } 1 - \epsilon.$$

Proof.

$$\begin{aligned} CS^{(0)}(u \mid v, H) &\geq^{+} CS^{(1)}(u, H \mid v) - CS^{(0)}(H \mid v) \\ &\geq^{+} CS^{(2)}(u \mid v) + CS^{(2)}(H \mid u, v) - CS^{(0)}(H \mid v) \\ &\geq^{+} CS^{(2)}(u \mid v) \text{ with probability } 1-\epsilon. \end{aligned} \tag{2}$$

In the first two lines, we use the chain rule, and in the last line, we use the fact that, for every i, $CS^{(i)}(H \mid u, v) \geq |H| - \log(1/\epsilon) - 1$, with probability $1-\epsilon$ (by a standard counting argument) and $CS^{(i)}(H \mid v) \leq |H| + O(1)$ for every H. ◁

Now we can show part (i) of Theorem 1.

$$\begin{aligned} |z| \quad = CS^{(0)}(x \mid p, H) \quad &\geq^{+} CS^{(1)}(x, p \mid H) - CS^{(0)}(p \mid H) \\ &\qquad \text{(chain rule)} \\ &\geq^{+} CS^{(2)}(x \mid H) - |p| \\ &\qquad \text{(because } |p| \geq^{+} CS^{(0)}(p \mid H)) \\ &\geq^{+} CS^{(2)}(x \mid H) - CS^{(0)}(x \mid y, H) \text{ with probability } 1-\epsilon \\ &\qquad \text{(because } |p| = CS^{(0)}(x \mid y, H)) \\ &\geq^{+} CS^{(4)}(x) - CS^{(0)}(x \mid y) \text{ with probability } 1-2\epsilon \\ &\qquad \text{(by Claim 3)} \end{aligned} \tag{3}$$

Next we move to part (ii), where we need to show that the complexity of the secret key z, conditioned by the transcript of the protocol, is close to the length of z. The transcript consists of p, H, n (all sent by Alice to Bob) and of Bob's sequence of responses $s = 000\ldots01$ of length $\ell = k + 1 + \log(1/\epsilon)$. Bob's sequence has complexity bounded by $\log \ell + O(1) = O(\log(n/\epsilon))$, and therefore, for every i we have $CS^{(i)}(z \mid s, p, H, n) =^{+} CS^{(i)}(z \mid p, H)$. Thus we can ignore s and n in the condition and it is enough to bound from below $CS^{(i)}(z \mid p, H)$. We show the following estimation, which ends the proof of the theorem.

▷ Claim 4. With probability $1-2\epsilon$, $CS^{(4)}(z \mid p, H) \geq^{+} |z| - \Delta$, where $\Delta = CS^{(-2)}(x) - CS^{(8)}(x)$.

Proof. We need an upper bound of $|z|$:

$$\begin{aligned} |z| \quad = CS^{(0)}(x \mid p, H) \quad &\leq^{+} CS^{(-1)}(x, p \mid H) - CS^{(0)}(p \mid H) \\ &\qquad \text{(chain rule)} \\ &\leq^{+} CS^{(-2)}(x \mid H) - CS^{(1)}(x \mid y, H) \text{ with probability } 1-\epsilon \\ &\qquad (p \text{ can be computed from } x,\ H \text{ and its length; and } x \text{ from } p, y, H) \\ &\leq^{+} CS^{(-2)}(x) - CS^{(3)}(x \mid y) \text{ with probability } 1-2\epsilon \\ &\qquad \text{(by Claim 3)} \end{aligned} \tag{4}$$

Next,

$$\begin{aligned} CS^{(5)}(p, z \mid H) &\geq^{+} CS^{(6)}(x \mid H) \text{ with probability } 1-\epsilon \\ &\qquad (x \text{ can be computed from } p, z, H) \\ &\geq^{+} CS^{(8)}(x) \text{ with probability } 1-2\epsilon \qquad \text{(by Claim 3)}, \end{aligned} \tag{5}$$

and

$$\begin{aligned} CS^{(5)}(p, z \mid H) &\leq^{+} CS^{(4)}(p \mid H) + CS^{(4)}(z \mid p, H) \\ &\qquad \text{(chain rule)} \\ &\leq^{+} CS^{(3)}(x \mid y, H) + CS^{(4)}(z \mid p, H) \\ &\qquad (p \text{ can be computed from } x, H \text{ and its length}) \\ &\leq^{+} CS^{(3)}(x \mid y) + CS^{(4)}(z \mid p, H) \end{aligned} \tag{6}$$

Combining inequalities (6) and (5), we obtain

$$CS^{(4)}(z \mid p, H) \geq^{+} CS^{(8)}(x) - CS^{(3)}(x \mid y) \text{ with probability } 1 - 2\epsilon. \tag{7}$$

Using inequality (4), we finally obtain

$$CS^{(4)}(z \mid p, H) \geq^{+} |z| - \Delta \text{ with probability } 1 - 4\epsilon, \tag{8}$$

where $\Delta = CS^{(-2)}(x) - CS^{(8)}(x)$. The conclusion follows after rescaling ϵ. ◁

4 Proof of Theorem 2

We first present *extractors*, which have been studied in the theory of pseudorandomness (for example, see [14]). A particular type of extractor, *prefix extractor*, is used in the protocol in the proof of Theorem 2 for hashing.

We recall that a (k, ϵ) extractor is a function $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with the property that for every subset $B \subseteq \{0,1\}^n$ of size at least 2^k and for every subset $A \subseteq \{0,1\}^m$:

$$\left|\mathrm{Prob}[\mathrm{E}(\mathrm{U_B}, \mathrm{U_d}) \in \mathrm{A}] - \frac{|\mathrm{A}|}{\mathrm{M}}\right| < \epsilon, \tag{9}$$

where U_B and U_d are independent random variables that are uniformly distributed over B and, respectively, $\{0,1\}^d$.

It is useful to view an extractor E as a bipartite graph G, whose set of left nodes is $\{0,1\}^n$, the set of right nodes is $\{0,1\}^m$, and each left node x has 2^d (not necessarily distinct) right neighbors $\{E(x,w) \mid w \in \{0,1\}^d\}$. The right node $E(x,w)$, for random $w \in \{0,1\}^d$, is viewed as the random fingerprint of the left node x.

As usual, we use *explicit* extractors. An explicit extractor is a family of extractors $\{E_n\}_{n\in\mathbb{N}}$ as above, indexed by n, and with the rest of the parameters k, d, m, ϵ being functions of n, such that there exists an algorithm that computes $E_n(x,w)$ in time polynomial in n. Actually, for us it is more important the space complexity of the algorithm that computes the extractor.

We denote $D = 2^d, M = 2^m$. Let B be a set of left nodes. The average numbers of neighbors in B of a right node (called the average B-degree) is $avg = |B| \cdot D/M$. We say that a right node p is ϵ-heavy for B if it has more $(1/\epsilon) \cdot avg$ left neighbors in B. We say that a left node $u \in \{0,1\}^n$ is ϵ-poor for B if a fraction larger than 2ϵ of its right neighbors are ϵ-heavy for B. Intuitively, a heavy p is a fingerprint that causes many collisions, and u is poor if many of its fingerprints produce many collisions.

The relevant property of extractors is presented in the next lemma. The point is that an ϵ-poor string is difficult to handle because a random fingerprint of it produces many collisions. The lemma gives a criterion which guarantees that a string is not ϵ-poor.

▶ **Lemma 5.** *There exist constants $\lambda > 1$ and c with the following property:*

Let $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a $(k-c, \epsilon)$ extractor computable in space $S(n)$ (in the above sense). Let x be an n-bit string (which in the protocol is Alice's input) and y be a string (which is Bob's input), such that $C^{S(n)}(x \mid y, n, k) \leq k$ and $C^{\lambda S(n)}(x \mid y, n, k-1) > k-1$, and let $B = \{u \in \{0,1\}^n \mid C^{S(n)}(u \mid y, n, k) \leq k\}$. Then x is not ϵ-poor for B.

Proof. Let A be the set of strings that are ϵ-heavy for B. By counting the edges between B and A from left-to-right and from right-to-left, we obtain that $|A|/M \leq \epsilon$. Let POOR be the set of nodes that are ϵ-poor for B. Note that

$$\mathrm{Prob}(\mathrm{E}(\mathrm{U_{POOR}}, \mathrm{U_d}) \in \mathrm{A}) > 2\epsilon \geq |\mathrm{A}|/\mathrm{M} + \epsilon,$$

It follows that POOR has size less than 2^{k-c}, because otherwise the set POOR would violate the property that E is a $(k-c,\epsilon)$-extractor.

Given y, n, k, c, the set POOR can be enumerated using space $S(n)+O(n)$ (we need the second term to maintain several counters which require $O(n)$ space). Taking into account the additional space needed by the universal machine, it follows that for some constant λ, if u is an ϵ-poor node then

$$C^{\lambda S(n)}(u \mid y, n, k, c) \leq k - c + O(1),$$

which implies $C^{\lambda S(n)}(u \mid y, n, k-1) \leq k-1$, for sufficiently large c. It follows that x is not ϵ-poor, which proves the lemma. ◀

We need to use a *prefix extractor*, which is a a function $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ with the property that for every $k \leq n$, the function E_k obtained by retaining only the prefix of length k of $E(x,w)$ is a (k,ϵ) randomness extractor. Raz, Reingold and Vadhan [12] have obtained an explicit extractor E_{RRV} of this type with $d = O(\log^3(n/\epsilon))$. $E_{\text{RRV}}(x,w)$ can be computed in time polynomial in n (recall that $n = |x|$). Let $p_0(n)$ be the polynomial that bounds the *space* used in the computation of $E_{\text{RRV}}(x,w)$.

In the protocol, we use the Raz-Reingold-Vadhan prefix extractor E_{RRV}. We denote by E_k, the k-prefix of E_{RRV}, and, abusing notation, also the bipartite graph corresponding to the (k,ϵ) extractor E_k.

In addition to E_{RRV}, we use a hash function h, based on congruences modulo prime numbers. We view a string x as an integer (in some canonical way) and define $h_t(x) = (x \bmod q, q)$, where q is a prime number chosen at random among the first t prime numbers. The properties of h_t follow from the following lemma.

▶ **Lemma 6** ([2]). *Let $x_1, x_2 \ldots, x_s$ be distinct n-bit strings, which we view in some canonical way as integers $< 2^{n+1}$. Let $t = (1/\epsilon) \cdot s \cdot n$. Let q be a prime number chosen uniformly at random among the first t prime numbers. Then, with probability $(1-\epsilon)$,*

$$x_1 \bmod q \notin \{x_2 \bmod q, \ldots, x_s \bmod q\}.$$

We now present the protocol. Recall that at the beginning of the protocol, Alice holds an n-bit string x, and Bob holds a string y. We fix the parameters as follows. Let λ and c be the constants guaranteed by Lemma 5, let $s = (1/\epsilon) \cdot 2^{c+1} \cdot D$, where $D = 2^d = 2^{O(\log^3(n/\epsilon))}$ is the degree of the E_{RRV} extractor, and let $t = (1/\epsilon) \cdot s \cdot n^2$. We use the space bound $S(n)$ and the constant $\lambda > 1$, given by Lemma 5 applied to the E_{RRV} extractor. We assume that the polynomial p_0 and the constant c, promised by Lemma 5, are large enough so that for every string x and every condition string u, $C^{p_0(|x|)}(x \mid u) \leq |x| + c$. As we did earlier, we use the abbreviated notation $CS^{(i)}(\ldots)$ for $C^{\lambda^i \cdot S(n)}(\ldots)$.

Phase 1: Information reconciliation. In Round 0, Alice sends to Bob, n and $h_t(x)$, where h_t is the hash function introduced above.

Next, Alice computes $p' = E_{\text{RRV}}(x,w)$ for a random $w \in \{0,1\}^d$.

Alice sends to Bob the string p' (or rather a prefix of it), one bit per round, till Bob announces that he does not need more bits.

Suppose we are at round k, after Alice has sent the k-th bit of p'. Thus, by now Bob has received p_k, the k-th bit long prefix of p'. He calculates, as we explain next, a set of candidate strings, which he thinks might be x. A string x' is a candidate at round k if

1. $x' \in B = \{u \in \{0,1\}^n \mid CS^{(n-k)}(u \mid y, n, k+c) \leq k+c\}$, and
2. x' is a neighbor of p_k, when viewing x' as a left node and p_k as a right node in the graph E_k, and

3. x' is among the first (in some canonical order) s strings with the above two properties. If no candidate has the fingerprint $h_t(x)$, then Bob asks for the next bit of p'. Otherwise, there is one candidate string x' so that $h_t(x') = h_t(x)$. Then Bob believes that x' is Alice's x, and he responds to Alice that he does not need further bits. The Phase 1 (information reconciliation) of the protocol is over.

We now analyze Phase 1 (information reconciliation). We show that with high probability, at the end of Phase 1, Bob obtains x.

Let $k^* = \min\{k \mid CS^{(n-k)}(x \mid y, n, k+c) \leq k+c\}$. By the above largeness assumptions for c and $p_0(n)$, it follows that $k^* \leq n$. Let $\mathcal{E}$ be the event that there exists x' other than x that is a candidate at one of the rounds $1, 2, \ldots, k^*$ and has the same fingerprint as x (i.e., $h_t(x') = h_t(x)$). The total number of candidates from rounds $1, 2, \ldots, k^*$ is at most $k^* \cdot s \leq n \cdot s$. It follows from Lemma 6, that $\mathcal{E}$ has probability at most ϵ. Conditioned on $\mathcal{E}$ not holding, either Bob finds correctly x before round k^* (this happens if x is a candidate at one of these earlier rounds), in which case we are done, or Phase 1 reaches round k^*.

Suppose Phase 1 reaches round k^*. Let $B = \{u \in \{0,1\}^n \mid CS^{(n-k^*)}(u \mid y, n, k^*+c) \leq k^*+c\}$. Clearly, by the definition of k^*,

$$CS^{(n-k^*)}(x \mid y, n, k^*+c) \leq k^*+c$$

and

$$CS^{(n-k^*+1)}(x \mid y, n, k^*+c-1) > k^*+c-1.$$

Now we use Lemma 5 for the pair (x, y), the (k^*, ϵ) extractor $E_{k^*} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^{k^*}$ and the set B. The size of B is less than 2^{k^*+c+1} and the average B-degree of a right node is $avg = |B| \cdot D/2^{k^*} \leq 2^{c+1} \cdot D$. By Lemma 5 and the two inequalities above, x is not ϵ-poor, which means that with probability $1-2\epsilon$, p_{k^*} is a right neighbor of x that is not heavy, i.e., it has at most $(1/\epsilon) \cdot avg \leq (1/\epsilon) \cdot 2^{c+1} \cdot D = s$ neighbors in B. Therefore, conditioned on non $\mathcal{E}$, with probability $1-2\epsilon$, x is a candidate at round k^*, and Bob finds it. We conclude that with probability larger than $1-3\epsilon$, Bob correctly obtains x.

Let p be the part of the protocol's transcript that Alice has sent to Bob. For the analysis of Phase 2, we need to evaluate the length of p. The string p consists of n, $h_t(x)$ and the prefix of p' that Alice has sent bit-by-bit before Bob told her that he does not need any further bits. By the analysis above, with probability $1-3\epsilon$, the length of the prefix of p' is at most k^*. Let $k = C^{S(n)}(x \mid y) - c$. Note that $k \leq n$. Since

$$CS^{(n-k)}(x \mid y, n, k+c) \leq CS^{(0)}(x \mid y) = k+c,$$

it follows from the definition of k^* that $k^* \leq k$. Next, the length of n and $h_t(x)$ is $O(\log^3(n/\epsilon))$ because the t-th largest prime number is less than $t \log t$. We conclude that

$$|p| \leq^+ CS^{(0)}(x \mid y). \tag{10}$$

The communication complexity is $2|p|$, because it consists of p and of Bob's responses $00\ldots01$.

Phase 2: Secret key construction. Alice and Bob compute by exhaustive search from x and p a program z of x given p in space $S(n)$ of minimal length $CS^{(0)}(x \mid p)$.

We now show that the protocol satisfies the requirements of Theorem 2, and we start with part (i). We let $\geq^+$ hide a loss of precision of $O(\log^3(n/\epsilon))$. We have

$$\begin{aligned} CS^{(0)}(x) &\leq^+ |p| + |z| && \text{(because } x \text{ is computed from } p \text{ and } z \text{ in space } S(n)) \\ &\leq^+ CS^{(0)}(x \mid y) + |z| && \text{(by (10))} \end{aligned}$$

Hence, $|z| \geq^+ CS^{(0)}(x) - CS^{(0)}(x \mid y)$.

Next we show part (ii) in Theorem 2. First notice that, by the chain rule,

$$|z| = CS^{(0)}(x \mid p) \leq^+ CS^{(-1)}(x,p) - CS^{(0)}(p). \tag{11}$$

Next,

$$\begin{aligned} CS^{(0)}(z \mid p) &\geq^+ CS^{(1)}(z,p) - CS^{(0)}(p) \\ &\qquad \text{(chain rule)} \\ &\geq^+ CS^{(1)}(x,p) - CS^{(0)}(p) \\ &\qquad \text{(because } x \text{ can be computed from } z \text{ and } p \text{ in space } S(n)) \\ &= CS^{(-1)}(x,p) - CS^{(0)}(p) - (CS^{(-1)}(x,p) - CS^{(1)}(x,p)) \\ &\geq^+ |z| - \Delta, \end{aligned}$$

where $\Delta = CS^{(-1)}(x,p) - CS^{(1)}(x,p)$. Since p can be computed from x and the seed of the extractor and the random prime number q used by h_t in space $p_0(n) \leq S(n)$, we have

$$\Delta \leq^+ CS^{(0)}(x) - CS^{(1)}(x).$$

The transcript of the protocol consists of p and Bob's sequence of responses $00\ldots01$, which has complexity bounded by $\log n$. Therefore

$$CS^{(0)}(z \mid \mathit{transcript}) \geq^+ CS^{(0)}(z \mid p) \geq^+ |z| - \Delta,$$

which proves part (ii) of Theorem 2. ◀

5 Final comments

As we have mentioned in the Introduction, the main results are of theoretical, rather than practical, relevance. The secret key agreement protocols in Theorem 1 and Theorem 2 produce a key that looks random to an adversary whose computation is space-bounded by $S(n)$, and, on the other hand, in both theorems, the two legal parties (i.e., Alice and Bob) execute the protocol in space larger than $S(n)$. For this reason, the protocols do not seem to be suitable for real cryptographic applications.

Another observation regards the key length. In Theorem 2, the protocol, on inputs the n-bit string x and the string y, runs in space bounded by $\lambda^n S(n)$ (we take into account the space used by the two parties combined) for some constant $\lambda > 1$ and produces a secret key z of length $|z| \approx C^{S(n)}(x) - C^{S(n)}(x \mid y)$ and having the randomness deficiency of z conditioned by the trancript as stated in the theorem. Recall that the randomness deficiency Δ is defined by $\Delta = |z| - C^{S(n)}(z \mid \mathit{transcript})$. Is the length of z optimal? It is known from [13], that no computable protocol can produce a key longer than $C(x) - C(x \mid y)$, the mutual information of the inputs hold by the two parties. We have not been able to obtain a similarly clean result for protocols that run in space $S(n)$. By adapting the arguments in [13], it can be shown, that if a protocol runs in space $S(n)$ then, for every pair of inputs (x,y) with length bounded by n, it produces a key z with $C^{\lambda^2 S(n)}(z \mid \mathit{transcript}) \leq C^{S(n)}(x) - C^{\lambda^3 S(n)}(x \mid y)$, for some constant $\lambda > 1$. Thus we obtain the following upper bound: If a secret key agreement protocol runs in space $S(n)$ and on input (x,y), with $|x|, |y| \leq n$, it produces a secret key z with randomness deficiency Δ, then

$$|z| \leq C^{S(n)}(x) - C^{\lambda^3 S(n)}(x \mid y) + \Delta + \Delta_1,$$

where $\Delta_1 = C^{(S(n)}(z \mid \mathit{transcript}) - C^{\lambda^2 S(n)}(z \mid \mathit{transcript})$ and $\lambda > 1$ is a constant.

References

1 Rudolf Ahlswede and Imre Csiszár. Common randomness in information theory and cryptography - I: secret sharing. *IEEE Trans. Information Theory*, 39(4):1121–1132, 1993. doi:10.1109/18.243431.

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 Charles H. Bennett, Gilles Brassard, and Jean-Marc Robert. Privacy amplification by public discussion. *SIAM Journal on Computing*, 17(2):210–229, 1988.

4 Mark Braverman and Anup Rao. Information equals amortized communication. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 748–757. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.86.

5 Harry Buhrman, Michal Koucký, and Nikolai K. Vereshchagin. Randomised individual communication complexity. In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity, CCC 2008, 23-26 June 2008, College Park, Maryland, USA*, pages 321–331. IEEE Computer Society, 2008. doi:10.1109/CCC.2008.33.

6 Alexander Kozachinskiy. On Slepian-Wolf theorem with interaction. *Theory Comput. Syst.*, 62(3):583–599, 2018. doi:10.1007/s00224-016-9741-x.

7 Sik Kow Leung-Yan-Cheong. Multi-user and wiretap channels including feedback, July 1976. Tech. Rep. No. 6603-2, Stanford Univ.

8 Ueli M. Maurer. Secret key agreement by public discussion from common information. *IEEE Trans. Information Theory*, 39(3):733–742, 1993. doi:10.1109/18.256484.

9 Andrei A. Muchnik. Conditional complexity and codes. *Theor. Comput. Sci.*, 271(1-2):97–109, 2002. doi:10.1016/S0304-3975(01)00033-0.

10 D. Musatov, A. E. Romashchenko, and A. Shen. Variations on Muchnik's conditional complexity theorem. *Theory Comput. Syst.*, 49(2):227–245, 2011. doi:10.1007/s00224-011-9321-z.

11 Prakash Narayan and Himanshu Tyagi. Multiterminal secrecy by public discussion. *Foundations and Trends in Communications and Information Theory*, 13(2-3):129–275, 2016. doi:10.1561/0100000072.

12 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.

13 Andrei E. Romashchenko and Marius Zimand. An operational characterization of mutual information in algorithmic information theory. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 95:1–95:14, 2018.

14 Salil P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1-3):1–336, 2012. doi:10.1561/0400000010.

15 Marius Zimand. Kolmogorov complexity version of Slepian-Wolf coding. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 22–32. ACM, 2017. doi:10.1145/3055399.3055421.

Using Statistical Encoding to Achieve Tree Succinctness Never Seen Before

Michał Gańczorz

Institute of Computer Science, University of Wrocław, Poland
mga@cs.uni.wroc.pl

Abstract

We propose new entropy measures for trees, the known ones are $H_k(\mathcal{T})$, the k-th order (tree label) entropy (Ferragina at al. 2005), and tree entropy $H(\mathcal{T})$ (Jansson et al. 2006), the former considers only the tree labels and the latter only tree shape. The proposed entropy measures, $H_k(\mathcal{T}|L)$ and $H_k(L|\mathcal{T})$, exploit the relation between the labels and the tree shape. We prove that they lower bound label entropy and tree entropy, respectively, i.e. $H_k(\mathcal{T}|L) \leq H(\mathcal{T})$ and $H_k(L|\mathcal{T}) \leq H_k(L)$. Besides being theoretically superior, the new measures are significantly smaller in practice.

We also propose a new succinct representation of labeled trees which represents a tree $\mathcal{T}$ using one of the following bounds: $|\mathcal{T}|\,(H(\mathcal{T}) + H_k(L|\mathcal{T}))$ or $|\mathcal{T}|\,(H_k(\mathcal{T}|L) + H_k(L))$. The representation is based on a new, simple method of partitioning the tree, which preserves both tree shape and node degrees. The previous state-of-the-art method of compressing the tree achieved $|\mathcal{T}|\,(H(\mathcal{T}) + H_k(L))$ bits, by combining the results of Ferragina at al. 2005 and Jansson et al. 2006; so proposed representation is not worse and often superior. Moreover, our representation supports standard tree navigation in constant time as well as more complex queries. Such a structure achieving this space bounds was not known before: aforementioned solution only worked for compression alone, our structure is the first which achieves $H_k(\mathcal{T})$ for $k > 0$ and supports such queries. Lastly, our data structure is fairly simple, both conceptually and in terms of the implementation, moreover it uses known tools, which is a counter-argument to the claim that methods based on tree-partitioning are impractical.

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

Keywords and phrases succinct data structures, labeled tree, ordered tree, entropy, tree entropy

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.22

Related Version `http://arxiv.org/abs/1807.06359`

Funding Supported under National Science Centre, Poland project number 2017/26/E/ST6/00191.

1 Introduction

String entropy. For a string S its (*zeroth order*) *entropy*, denoted by $H_0(S)$, is defined as $|S|H_0(S) = -\sum_{s\in\Sigma} t_s \log \frac{t_s}{|S|}$, where t_s is a number of occurrences of character s in S. It is convenient to think that $-\log \frac{t_s}{|S|}$ assigned to a symbol s is the optimal cost of encoding this symbol (in bits); those "values" are usually not natural numbers.

The standard extension of this measure is the k-th order entropy, denoted by H_k, in which the (empirical) probability of s is conditioned by k preceding letters, i.e. the cost of single occurrence of letter s is equal to $-\log \mathbb{P}(s|w)$, where $\mathbb{P}(s|w) = \frac{t_{ws}}{t_w}$, $|w| = k$ and t_v is the number of occurrences of a word v in given word S. We call $\mathbb{P}(s|w)$ the *empirical probability* of a letter s occurring in a k-letter context. Then $|S|H_k(S) = -\sum_{s\in\Sigma, w\in\Sigma^k} t_{ws} \log \mathbb{P}(s|w)$. The cost of encoding the first k letters is ignored when calculating the k-th order entropy. This is acceptable, as k is (very) small compared to $|S|$, for example most tools based on popular context-based compressor family PPM use $k \leq 16$. There are multiple methods compressing given string to the size of at most $|S|H_k(S)$ bits (plus smaller order terms),

like aforementioned PPM or BWT with compression boosting, there are also compressed structures which consume such space, and moreover allow to perform operations on texts, like access [16], insertion/deletion [26] rank/select [4, 25] or even pattern matching [24].

Tree Label entropy. In case of labeled trees, 0-th order entropy of tree labels has a natural definition: it is a zeroth order entropy of string made by concatenation of labels of vertices. However, for k-th order entropy the situation is more involved, as we have to somehow define the context.

Ferragina et al. [13] proposed a definition of k-th order entropy of labeled trees. The context is defined as the k labels from the node to the root. Such definition of entropy takes into the account only the labels, and so we call it the *k-th order entropy of labels*. It is claimed to be good measure both in practice [14] (as similarly labeled nodes descend from similarly labeled contexts [13]) and in theory, as it is related to notion of tree sources [10].

Ferragina at al. [13] not only introduced the concept of k-th order entropy but also proposed a novel transform, XBWT, which works by generating a single string from tree labels by employing BWT-like transform on trees. It can be used to compress the tree using $|\mathcal{T}|(2 + H_k(\mathcal{T}) + \mu) + o(|\mathcal{T}|)$ bits [13], for a reasonable k; where the $2|\mathcal{T}|$ bits are needed for encoding the tree shape and μ is a small constant, an artifact of used method of compression boosting (well-known in text compression [15]). The major drawback of this approach is that it works only for the case of compression alone, i.e. this approach does not generalize to a structure on which we can perform queries, as to this end rank/select structure over alphabet of size σ is required, which seems hard to combine with compression boosting. Dropping the compression boosting gives the structure which achieves $H_0(\mathcal{T})$ and even then it supports only navigational queries (in particular, it does not support level_ancestor or depth), though it also support some basic label-related operations, as well as path pattern matching (i.e. pattern matching on strings made by concatenating labels of vertices on paths from root to leaves). The other XBWT-related result is by Ferragina et al. [14], who proposed a solution which encodes the labels of the tree using k-th order entropy of string produced by XBWT and support some navigational queries for $k = o(\log_\sigma |\mathcal{T}|)$; the k-th order entropies of the tree and of string produced by XBWT seem loosely related and no formal relation is known between the two, but they argued that intuitively they are connected, because they both similarly cluster node labels. Still, this structure supports a very limited set of operations, has provably non-constant query time (due to the lower bounds on rank/select indices [4, 25]) and large additional space consumption.

Tree entropy. The (k-th order) entropy of labels ignores the shape of the tree and the information carried by it, and we still need to represent the tree structure somehow. A counting argument shows that $2|\mathcal{T}|$ bits are needed for a random unlabeled tree [32], and indeed there are representations [32, 31, 6] and succinct structures achieving such bounds [42, 40, 12, 39, 43, 6, 31]. Yet, the $2|\mathcal{T}|$ bits bound seems sub-optimal, as similarly as in the case of strings, real data is rarely a random tree drawn from the set of all trees, for example XML files are shallow and some tree shapes repeat, like in the Figure 1. For this reason Jansson et al. [33] introduced the notion of *tree entropy* with the idea that it takes into the account the probability of a node having a particular degree, i.e. it measures the number of trees under some degree distribution. Formally, it is defined as: $|\mathcal{T}|H(\mathcal{T}) = -\sum_{i=0}^{|\mathcal{T}|} d_i \log \frac{d_i}{|\mathcal{T}|}$, where d_i is the number of vertices of degree i in $\mathcal{T}$. Up to $\Theta(\log |\mathcal{T}|)$ additive summand, tree entropy is an information-theoretic lower bound on the number of bits needed to represent unlabeled tree that has some fixed degree distribution [33]. Similarly, as in the case of string

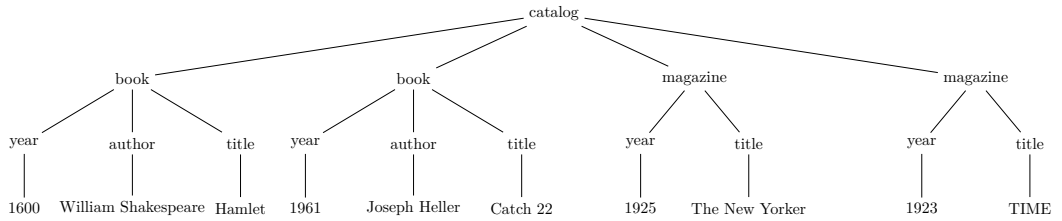


Figure 1 Sample XML file structure.

entropy, tree entropy refines the simpler estimations, in the sense that $|\mathcal{T}|H(\mathcal{T}) \leq 2|\mathcal{T}|$ [33]. Jansson et al. [33] also showed a structure for unlabeled trees which supported most of the tree operations and use at most $|\mathcal{T}|H(\mathcal{T})$ bits. This approach can be combined with aforementioned XBWT with compression boosting to compress the tree to roughly $|\mathcal{T}|H(\mathcal{T})+|\mathcal{T}|H_k(\mathcal{T})$ bits, but again it works only for the case of compression alone and needs a small additional linear factor, $\mu|\mathcal{T}|$, same as when using only XBWT with compression boosting.

Our contribution. Label entropy and tree entropy treat labels and tree structure separately, and so did most of the previous approaches to labeled tree data structures [13, 33, 14, 28]. Yet, the two are most likely correlated: one can think of XML document representing the collection of different entities such as books, magazines etc., see Figure 1. Knowing that the label of some node is equal to "book", and that each book has an author, a year and a title, determines degree of the vertex to be three (cf. tree grammar compression model, where we explicitly assume that the label uniquely defines the arity of node [9]). On the other hand, knowing the degree of the vertex can be beneficial for information on labels.

Motivated by this, we start by defining two new measures of entropy of trees, which take into the account *both* tree structure *and* tree labels: $H_k(\mathcal{T}|L)$ and $H_k(L|\mathcal{T})$. Those measures lower bound previous measures, i.e tree entropy [33] and H_k [13], respectively. We also devise the encoding achieving newly defined values. To this end, we propose a new way of partitioning the tree. In contrast to previous approaches, (i.e. succinct representations [22, 12] and tree grammar compression [19]), this partition preserves *both* the shape of the tree *and* the degrees of the nodes. We show that by applying an entropy coder to tree partition we can bound the size of the tree encoding by both $|\mathcal{T}|H_k(\mathcal{T}|L) + |\mathcal{T}|H_k(L) + \mathcal{O}(|\mathcal{T}|k \log \sigma / \log_\sigma |\mathcal{T}| + |\mathcal{T}| \log \log_\sigma |\mathcal{T}| / \log_\sigma |\mathcal{T}|)$ and $|\mathcal{T}|H_k(L|\mathcal{T}) + |\mathcal{T}|H(\mathcal{T}) + \mathcal{O}(|\mathcal{T}|(k + 1) \log \sigma / \log_\sigma |\mathcal{T}| + |\mathcal{T}| \log \log_\sigma |\mathcal{T}| / \log_\sigma |\mathcal{T}|)$ bits. Note that the additional summands are $o(|\mathcal{T}| \log \sigma)$ if $k = o(\log_\sigma |\mathcal{T}|)$. This is the first method not based on XBWT achieving bounds related to H_k, moreover it does not need additional $\mu|\mathcal{T}|$ bits.

Using standard techniques we can augment our tree encoding, at the cost of increasing the constants hidden in the $\mathcal{O}$ notation (the overall complexity is the same and only constants in the "lower order terms" are slightly larger), so that most of the queries are supported in constant time, thus getting the first structure which achieves H_k for trees and supports queries on compressed representation. Previous state-of-the-art methods based on tree grammars [18] and high-order compressed XBWT [13], do not have this property.

Then we show that we can further reduce the redundancy to $\mathcal{O}(|\mathcal{T}| \log \log |\mathcal{T}| / \log_\sigma |\mathcal{T}|)$ bits, at the cost of increasing query time to $\mathcal{O}(\log |\mathcal{T}| / \log \log |\mathcal{T}|)$, assuming $k = \alpha \log_\sigma |S|$ where $\alpha < 1$. To this end we combine ideas from compression boosting [15] and compressed text representations [26].

Note that the high-order compressed structures for strings supporting access queries achieve analogous bit-size and query time: i.e. they either use $|S|H_k(S) + \mathcal{O}(|S|k\log\sigma/\log_\sigma|S| + |S|\log\log_\sigma|S|/\log_\sigma|S|)$ bits and support queries in $\mathcal{O}(1)$ time or they use $|S|H_k(S) + \mathcal{O}(|S|\log\log|S|/\log_\sigma|S|)$ bits and have the query time of $\mathcal{O}(\log|S|/\log\log|S|)$ [16, 23, 26], and it is an open problem whether this trade-off is optimal. It is straightforward that improving our bounds even when the structure supports only preorder_rank/preorder_select operations yields the improvement for the high-order compressed structures for strings.

To compare, the original method of XBWT consumed $|\mathcal{T}|H_0(L)$ bits for structure supporting navigation queries, also previously mentioned result by Ferragina et al. [14], which considered k-th order entropy of XBWT string instead k-th order label entropy, worked only for $k = o(\log_\sigma|S|)$ and had a redundancy $o(|S|\log\sigma)$.

Moreover, we can use known tools to support label-related operations (like childrank(v, a) or childselect(v, i, a)). As those two are generalization of rank/select on large alphabets, the obtainable redundancy is larger than aforementioned bounds achieved by our structures, we also have to allow for a non-constant query time (see lower bounds in [4, 25]). In this case we get the same redundancy and query time for label operations as one of the previously mentioned results by Ferragina et al. [14], (it used space proportional to H_k of XBWT string and supported only navigational queries), but in contrast to this result, our structure still supports all non-label related operations in constant time, thus we get the structure which outperforms previously known ones.

Our methods can also be applied to unlabeled trees to achieve tree entropy [33], in which case we get the same (best known) additional space as in [33] and our techniques in case of standard operations are less complex than other methods achieving tree entropy [33, 11].

Lastly, our structure allows to retrieve the tree in optimal, $\mathcal{O}(|\mathcal{T}|/\log_\sigma|\mathcal{T}|)$ time (assuming machine words of size $\Theta(\log|\mathcal{T}|)$), using $\mathcal{O}(1)$ memory, in contrast to XBWT-based methods.

The comparison of our results and previous results based on XBWT is in Table 2. Table 1 gives an empirical comparison of our measures with previous ones.

Table 1 Data is taken from *XMLCompBench (http://xmlcompbench.sourceforge.net)*. The last four columns correspond to the bits per node used by compressed data structures (ignoring the smaller order terms); note that the proposed structure is the only one to (simultaneously) attain the last two. Note that sometimes the measures do not decrease when k grows, this is because some files have shallow structure.

File	k	$H(\mathcal{T})$	$H_k(L)$	$H_k(\mathcal{T}\|L)$	$H_k(L\|\mathcal{T})$	$2 + H_k(L)$	$H(\mathcal{T}) + H_k(L)$	$H_k(\mathcal{T}\|L) + H_k(L)$	$H(\mathcal{T}) + H_k(L\|\mathcal{T})$
EnWikiNew	0	0.233	0.537	0.031	0.335	2.537	0.77	0.568	0.568
	1	0.233	0.311	0.031	0.216	2.311	0.544	0.342	0.45
	2	0.233	0.311	0.031	0.216	2.311	0.544	0.342	0.45
Nasa	0	0.33	0.849	0.06	0.579	2.849	1.178	0.909	0.909
	1	0.33	0.232	0.056	0.174	2.232	0.562	0.288	0.503
	2	0.33	0.23	0.056	0.172	2.23	0.559	0.286	0.501
Treebank	0	1.812	4.616	0.692	3.495	6.616	6.428	5.308	5.308
	1	1.812	3.234	0.656	2.259	5.234	5.046	3.89	4.072
	2	1.812	3.073	0.64	2.109	5.073	4.886	3.713	3.922
	4	1.812	2.957	0.626	1.997	4.957	4.77	3.584	3.809

2 Definitions

We denote the input alphabet by Σ, and size of the input alphabet as $\sigma = |\Sigma|$. For a tree $\mathcal{T}$ we denote by $|\mathcal{T}|$ the number of its nodes, the same applies to forests. We consider rooted (i.e. there is a designated root vertex), ordered (i.e. children of a given vertex have a left-to-right order imposed on them) Σ-labeled (i.e. each node has a label from Σ) trees; label *does not* determine node degree nor vice versa. We assume that bit sequences of length $\log |\mathcal{T}|$ fit into $\mathcal{O}(1)$ machine words and we can perform operations on them in $\mathcal{O}(1)$ time.

Tree Label entropy. The tree label entropy [13] defines the context of a node as the concatenations of k labels from the node to the root. Similarly, as in the case of first k letters in strings, this is undefined for nodes whose path to the root is of length less than k, which can be large, even when k is small. There are two ways of dealing with this problem: in the first we allow the node to have the whole path to the root as its context (when this path is shorter than k); in the second we pad the too short context with some fixed letters. Our algorithms can be applied to both approaches, resulting in the same (asymptotic) redundancy; for the sake of the argument we choose the first one, as the latter can be easily reduced to the former.

The tree label entropy is formally defined as $|L|H_k(L) = -\sum_{v\in\mathcal{T}} \log \mathbb{P}(l_v|K_v)$, where l_v is label of vertex v, K_v is the word made by last k labels of nodes on the path from root of $\mathcal{T}$ to v (or less if the path from the root to v is shorter than k) and, as in the case of strings, $\mathbb{P}(l_v|K_v)$ is the empirical probability of label l_v conditioned that it occurs in context K_v.

Mixed entropy. We define the mixed entropies as follows:

- $|\mathcal{T}|H_k(L|\mathcal{T}) = -\sum_{v\in\mathcal{T}} \log \mathbb{P}(l_v|K_v, d_v)$, where $\mathbb{P}(l_v|K_v, d_v)$ is the empirical probability of node v having label l_v conditioned that it occurs in the context K_v and the node degree is d_v, that is, $\mathbb{P}(l_v|K_v, d_v) = \frac{t_{K,l_v,d_v}}{t_{K,d_v}}$, where t_{K,l_v,d_v} is a number of nodes in $\mathcal{T}$ with context K, having degree d_v and a label l_v, and t_{K,d_v} is number of nodes in $\mathcal{T}$ preceded by the context K, and having degree d_v.

- $|\mathcal{T}|H_k(\mathcal{T}|L) = -\sum_{v\in\mathcal{T}} \log \mathbb{P}(d_v|K_v, l_v)$, where $\mathbb{P}(d_v|K_v, l_v)$, is the empirical probability of node v having degree d_v conditioned that v has a context K_v and a label l_v, defined similarly as above.

We show that we can represent a tree using either $|\mathcal{T}|H_k(\mathcal{T}|L)+|\mathcal{T}|H_k(L)$ or $|\mathcal{T}|H_k(L|\mathcal{T})+|\mathcal{T}|H(\mathcal{T})$ bits (plus some small order terms), see Section 5.

The new measures lower bound the old ones. This follows directly from *log sum inequality*: intuitively increasing number of contexts can only reduce the entropy.

▶ **Lemma 1.** *The following inequalities hold:*

$$|\mathcal{T}|H_k(\mathcal{T}|L) \le |\mathcal{T}|H(\mathcal{T}) \quad \textit{and} \quad |\mathcal{T}|H_k(L|\mathcal{T}) \le |\mathcal{T}|H_k(L) \ .$$

Table 2 Comparison of the results, multiple values in the same operation set means that we can choose one of the possibilities,
$\rho_a = \mathcal{O}((k \log \sigma + \log\log_\sigma |\mathcal{T}|)/\log_\sigma |\mathcal{T}|)$; $\rho_b = \mathcal{O}(\log\log|\mathcal{T}|/\log_\sigma|\mathcal{T}|)$; $\gamma = \mathcal{O}(\log\sigma/\log_\sigma|\mathcal{T}|)$;
$g_k = \text{polylog}(\sigma)\cdot\sigma^k$
Navigation={parent(v), firstchild(v), nextsibling(v), childrank(u), child(u,i)};
Extended = {lca(u,v) and depth(u)};
Label = {childrank(v, a) and childselect(v, i, a)}, i.e. versions of childrank(u) and child(u,i) which take label into account;
Level-Ancestor = {level_ancestor(v, i)}.
XBWT(L) refers to the XBWT string [14]. T_0(rank/select))/S_0(rank/select) and T_k(rank/select))/S_k(rank/select) refer to the time and space required for structures supporting rank/select over strings over σ-sized alphabets, the first pair refers to query time/additional space required when the string is compressed with at most $|S|H_0(S)$ bits, respectively the second refers to the case when string is compressed using $|S|H_k(S)$ bits. Those values depends on the alphabet size, in general, for strings compressed with $|S|H_k(S)$ bits either S_k(rank/select) is larger than $|\mathcal{T}|\rho_a$ and $|\mathcal{T}|\rho_b$ or query time is not constant [4].

Operation set	Our structure		XBWT-based	
	time	space	time	space
Compression	—	$\|\mathcal{T}\|(H(\mathcal{T}) + H_k(L) + \rho_b)$ $\|\mathcal{T}\|(H_k(\mathcal{T}\|L) + H_k(L) + \rho_b)$ $\|\mathcal{T}\|(H(\mathcal{T}) + H_k(L\|\mathcal{T}) + \rho_b + \gamma)$	—	$\|\mathcal{T}\|(H(\mathcal{T}) + H_k(L) + \Theta(1)) + g_k$
Navigation	$\mathcal{O}(1)$	$\|\mathcal{T}\|(H(\mathcal{T}) + H_k(L) + \rho_a)$ $\|\mathcal{T}\|(H_k(\mathcal{T}\|L) + H_k(L) + \rho_a)$ $\|\mathcal{T}\|(H(\mathcal{T}) + H_k(L\|\mathcal{T}) + \rho_a + \gamma)$	$\mathcal{O}(\mathrm{T}_0$(rank/select))	$\|\mathcal{T}\|(2 + H_0(L))$ + S_0(rank/select)
	$\mathcal{O}\left(\frac{\log\|\mathcal{T}\|}{\log\log\|\mathcal{T}\|}\right)$	$\|\mathcal{T}\|(H(\mathcal{T}) + H_k(L) + \rho_b)$ $\|\mathcal{T}\|(H_k(\mathcal{T}\|L) + H_k(L) + \rho_b)$ $\|\mathcal{T}\|(H(\mathcal{T}) + H_k(L\|\mathcal{T}) + \rho_b + \gamma)$	$\mathcal{O}(\mathrm{T}_k$(rank/select))	$\|\mathcal{T}\|(2 + H_k(\mathsf{XBWT}(L)))$ + S_k(rank/select)
Extended	as Navigation			not supported
Level-Ancestor	as Navigation plus $\mathcal{O}(\|\mathcal{T}\|(\log\log\|\mathcal{T}\|)^2/\log_\sigma\|\mathcal{T}\|)$ bits			not supported
Label	Depends on the alphabet size, see Section 7			as Navigation

3 Tree clustering

We present a new clustering method which preserves both node labels and vertex degrees.

Clustering. The idea of grouping nodes was used before in the context of compressed tree indices [22, 27], also some dictionary compression methods like tree-grammars or top-trees and other carry some similarities [8, 18, 29, 21]. Yet, from our perspective, their main disadvantage is that the node degrees in the internal representation were very loosely connected to the node degrees in the input tree, thus the tree entropy and mixed entropies are hardly usable in upper bounds on space usage. We propose a new clustering method, which preserves the node degrees and the tree structure much better: most vertices inside clusters have the same degree as in $\mathcal{T}$, and the rest have their degree zeroed. As we show, this property implies a bound on the used space both in terms of the label/tree entropy and mixed entropy of $\mathcal{T}$, when an entropy coder is used to compress the multiset of clusters.

The idea of our clustering technique is that we group nodes into clusters of $\Theta(\log_\sigma|\mathcal{T}|)$ nodes, and collapse each cluster into a single node, thus obtaining a tree $\mathcal{T}'$ of $\mathcal{O}(|\mathcal{T}|/\log_\sigma|\mathcal{T}|)$ nodes. We label its nodes so that the new label uniquely determines the cluster that it represents and separately store the description of the clusters.

The clustering uses a parameter m, $2m-1$ is the maximum size of the cluster. Each node of the tree is in exactly one cluster and there are two types of nodes in a cluster: *port* and *regular* nodes. A port node is a leaf in a cluster and a non-leaf in $\mathcal{T}$, for a regular node

all its children in $\mathcal{T}$ are also in the same cluster (in the same order as in $\mathcal{T}$); in particular, its degree in the cluster is the same as in $\mathcal{T}$. Observe that this implies that each node with degree larger than $2m$ will be a port node.

The desired properties of the clustering are:

(C1) there at least $\frac{|\mathcal{T}|}{2m} - 1$ and at most $\frac{2|\mathcal{T}|}{m} + 1$ clusters;
(C2) each cluster is of size at most $2m - 1$;
(C3) each cluster is a forest of subtrees (i.e. connected subgraphs) of $\mathcal{T}$, roots of trees in this forest are consecutive siblings in $\mathcal{T}$;
(C4) each node in a cluster C is either a port node or a regular node; each port node is a leaf in C and non-leaf in $\mathcal{T}$, each regular node has the same degree in C as in $\mathcal{T}$. In particular, if node u belongs to some cluster C, then either all or none of its children are in C.

A clustering satisfying (C1–C4) can be found using a natural bottom-up greedy algorithm. Note that the previously known tree factorizations [22, 12] do not satisfy the above properties, especially the (C3) and (C4), which are crucial for our structure.

▶ **Lemma 2.** *Let $\mathcal{T}$ be a labeled tree. For any $m \leq |\mathcal{T}|$ we can construct in linear time a partition of nodes of $\mathcal{T}$ into clusters satisfying conditions (C1–C4).*

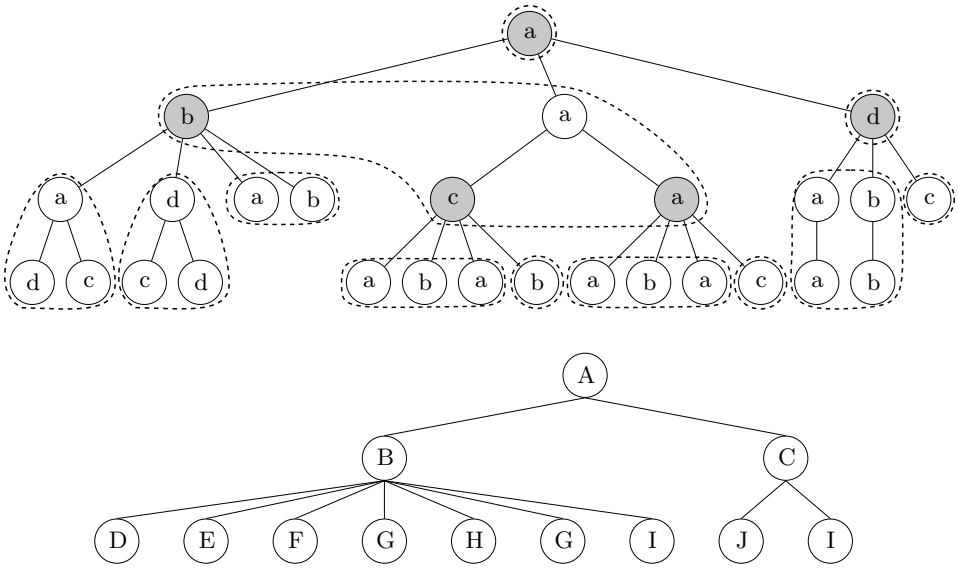


Figure 2 Clustering of tree for parameter $m = 3$ and tree created by replacing clusters with new nodes. Marked nodes are port nodes.

Building the cluster tree. We build the *cluster tree* out of the clustering satisfying (C1–C4): we replace each cluster with a new node and put edges between new nodes if there was an edge between some nodes in the corresponding clusters. To retrieve the original tree $\mathcal{T}$ from the cluster tree and its labels we need to know the degree of each port node in the original tree (note that this depends not only on the cluster, but also on the particular cluster node, e.g. two clusters may have the same structure but can have different port node degrees in $\mathcal{T}$). Thus we store for a cluster node with k ports a *degree sequence* $d_1, d_2, \ldots, d_k$, where d_i is the number of clusters containing children of i-th port node. For an illustration, see cluster replaced by cluster node labeled B in Figure 2, its degree sequence is $3, 2, 2$. In section 5 we show that degree sequence can be stored efficiently and along with cluster tree it is sufficient to retrieve and navigate $\mathcal{T}$.

▶ **Definition 3** (Cluster structure). *For a labeled tree $\mathcal{T}$ and parameter m a* cluster structure *$C(\mathcal{T})$ consists of:*

- *Ordered, rooted, labeled tree $\mathcal{T}'$ (called* cluster tree*) with $\Theta(|\mathcal{T}|/m)$ nodes, where each node represents a cluster, different labels correspond to different clusters and the induced clustering of $\mathcal{T}$ satisfies (C1–C4).*
- *For each node $v' \in \mathcal{T}'$, a degree sequence $d_{v',1}, \ldots d_{v',j}$, where $d_{v',i}$ means that i-th port node in left-to-right order on leaves of cluster represented by v' connects to $d_{v',i}$ clusters of $\mathcal{T}'$.*
- *Look-up tables, which for a label of $\mathcal{T}'$, allow to retrieve the corresponding cluster C.*

▶ **Lemma 4.** *For a labeled tree $\mathcal{T}$ and a parameter m we can construct in time $\mathcal{O}(|\mathcal{T}|)$ cluster structure $C(\mathcal{T})$.*

4 Entropy estimation

We show that entropy of labels of tree of $C(\mathcal{T})$ is upper bounded by the mixed entropy of the input tree, up to some small additive factor. Due to the technical details, in the last case the redundancy depends on $k+1$ and not on k.

▶ **Theorem 5.** *Let $\mathcal{T}$ be a labeled tree and let $\mathcal{T}'$ be a tree of cluster structure $C(\mathcal{T})$ from Lemma 4 for parameter m. Let P be a string obtained by concatenation of labels of $\mathcal{T}'$. Then* all *the following inequalities simultaneously hold:*

$$|P|H_0(P) \leq |\mathcal{T}|H(\mathcal{T}) + |\mathcal{T}|H_k(L) + \mathcal{O}\left(\frac{|\mathcal{T}|k\log\sigma}{m} + \frac{|\mathcal{T}|\log m}{m}\right), \tag{1}$$

$$|P|H_0(P) \leq |\mathcal{T}|H_k(\mathcal{T}|L) + |\mathcal{T}|H_k(L) + \mathcal{O}\left(\frac{|\mathcal{T}|k\log\sigma}{m} + \frac{|\mathcal{T}|\log m}{m}\right), \tag{2}$$

$$|P|H_0(P) \leq |\mathcal{T}|H(\mathcal{T}) + |\mathcal{T}|H_k(L|\mathcal{T}) + \mathcal{O}\left(\frac{|\mathcal{T}|(k+1)\log\sigma}{m} + \frac{|\mathcal{T}|\log m}{m}\right). \tag{3}$$

Moreover, if $m = \Theta(\log_\sigma|\mathcal{T}|)$, the additional terms are bounded by: $\mathcal{O}\left(\frac{|\mathcal{T}|k\log\sigma}{\log_\sigma|\mathcal{T}|} + \frac{|\mathcal{T}|\log\log_\sigma|\mathcal{T}|}{\log_\sigma|\mathcal{T}|}\right)$ and $\mathcal{O}\left(\frac{|\mathcal{T}|(k+1)\log\sigma}{\log_\sigma|\mathcal{T}|} + \frac{|\mathcal{T}|\log\log_\sigma|\mathcal{T}|}{\log_\sigma|\mathcal{T}|}\right)$, respectively. For $k = o(\log_\sigma|\mathcal{T}|)$ both those values are $o(|\mathcal{T}|\log\sigma)$.

We prove Theorem 5 in two steps. First, we devise a special representation of nodes of $\mathcal{T}'$. The entropy of this representation is not larger than the entropy of P, thus we upper bound the entropy of P. To this end we use the following corollary from Gibbs' inequality, see [1] for a simple proof.

▶ **Lemma 6** ([1]). *Let $w \in \Gamma^*$ be a string and $q : \Gamma \to \mathbb{R}^+$ be a function such that $\sum_{s\in\Gamma} q(s) \leq 1$. Then $|w|H_0(w) \leq -\sum_{s\in\Gamma} t_s \log q(s)$, where t_s is the number of occurrences of s in w.*

Lemma 6 should be understood as follows: we can assign each letter in a string a "probability" and calculate the "entropy" for a string using those "probabilities." The obtained value is not smaller than the true empirical entropy. Thus, to upper-bound the entropy, it is enough to devise an appropriate function q.

Cluster representation. The desired property of the representation is that for each node v in the cluster we can uniquely determine its context (i.e. labels of k nodes on the path from v to the root) in original tree $\mathcal{T}$.

▶ **Definition 7** (Cluster description). *Given a cluster C occurring in context K and consisting of subtrees $\mathcal{T}_1, \ldots, \mathcal{T}_l$, the* cluster description*, denoted by $R_{K,C}$, is a triplet (K, N_C, V_C), where*

- *K is a context (of size at most k) preceding roots of the trees in cluster, i.e. for each root r of tree in a cluster $K = K_r$ holds, where K_v is the context of vertex v in $\mathcal{T}$; roots of trees in a cluster have the same context in $\mathcal{T}$, as they have the same parent.*
- *N_C is the total number of nodes in this cluster;*
- *V_C is a list of descriptions of nodes of C, according to preorder ordering. If a node v is a port then its description is $(1, l_v)$, if it is a regular node, then it is $(0, l_v, d_v)$, where d_v is the degree of the node v (in the cluster) and l_v is its label.*

Note that we do not store the degrees of port nodes, as they are always 0.

▶ **Example 8.** The description for $k = 1$ and central node (the one which is labeled B in cluster tree) from Figure 2 is: $(K = a, N = 4, V = \{(1, b), (0, a, 2), (1, c), (1, a)\})$.

▶ **Lemma 9.** *Cluster description of a cluster C uniquely defines a cluster C and context K_v in $\mathcal{T}$ for each vertex v in C.*

To prove Theorem 5, instead of estimating the entropy of P we will estimate the entropy of string where letters are descriptions of each cluster. To this end we employ Lemma 6: we assign each description a value q in a way similar to the adaptive arithmetic coding: we assign each element of the description a separate value of q, which depends on both the element and previous elements of a description, then we multiply all such values. Note that storing, which nodes are port nodes inside a cluster description, uses additional $\mathcal{O}(|\mathcal{T}| \log m/m)$ bits, if done naively (e.g. in a separate structure) it would use the same memory.

Proof of Theorem 5. Let P be a sequence of labels. Let $R \in \Gamma_R^*$ be a sequence of description of clusters of $\mathcal{T}'$ in preorder ordering, so that description $R[i]$ of cluster C corresponds to label $P[i]$. Then $H_0(P) \leq H_0(R)$, in particular $|P|H_0(P) \leq |R|H_0(R)$. To see this, note that each label of P may correspond to a few descriptions from R, but not the other way around: if some nodes have different description, then they have different labels in $\mathcal{T}'$.

Thus it is enough to upper-bound $|R|H_0(R)$. To this end we apply Lemma 6 for appropriately defined function q; different estimations require different variants of q. The function q is defined on each $R[i]$. The assignment of values of q can be thought as a procedure that starts with value 1 and then looks at each element of description and multiplies by a value which depends on this element in description (or some previous ones), i.e. as in adaptive arithmetic coding.

We begin with a proof for Case 2. Let $R_{K,C} = (K, N_C, V_C)$ be a description of the cluster C. We define $q(R_{K,C}) = q(K_C) \cdot q(N_C) \cdot q(V_C)$, where q for each coordinate is defined as follows:

- $q(K_C) = 1/\left((k+1) \cdot \sigma^{|K_C|}\right)$
- $q(N_C) = 1/(2m)$
- $q(V_C) = \prod_{v \in V_C} q(v)$, where

$$q(v) = \begin{cases} \frac{1}{m} \cdot \mathbb{P}(l_v | K_v) & \text{, if } v = (1, l_v) \\ \frac{m-1}{m} \cdot \mathbb{P}(l_v | K_v) \cdot \mathbb{P}(d_v | K_v, l_v) & \text{, if } v = (0, l_v, d_v) \end{cases}$$

Note that K_v is the context of v in original tree, i.e. $\mathcal{T}$.

It is left to show that q summed over all cluster descriptions is at most 1 as required by Lemma 6. To this end we will show that we can partition the interval $[0, 1]$ into subintervals and assign each element of Γ_R a subinterval of length $q(R_{K,C})$, such that for two symbols

of Γ_R their intervals are pairwise disjoint. It is analogous to applying adaptive arithmetic coder. We start with interval $I = [0, 1]$. We process description of cluster by coordinates, at each step we partition I into disjoint subintervals and choose one as the new I:

- For K_C we partition interval into $k + 1$ equal subintervals, each one corresponding to different context length, then we choose one corresponding to $|K_C|$. Then we partition the I into $\sigma^{|K_C|}$ disjoint and equal subintervals, one for each different context of length $|K_C|$ and choose one corresponding to K_C. Clearly the length of the current I at this point is $1/\left((|K_C| + 1) \cdot \sigma^{|K_C|}\right)$, also different contexts are assigned different intervals.
- For N_C we partition the I to $2m$ equal intervals, it is enough, as there are at most $2m$ different cluster sizes.
- Then we make a partition for each $v \in V_C$. We process vertex descriptions in V_C according to the order they occur in list V_C, i.e. the order is the preorder ordering of corresponding vertices in C. We partition the interval into two, one of length $\frac{|I|}{m}$, second $\frac{|I|(m-1)}{m}$. If $v = (1, l_v)$ we choose the first one, otherwise we choose the second one. Then we partition the $|I|$ into σ different intervals (some may be of 0 length), one for each letter a of original alphabet Σ; the subinterval for letter a has length $|I| \cdot \mathbb{P}(a|K_v)$. It is a proper partition, as all of the above values sum up to 1 by definition. We choose the one corresponding to the letter l_v. Lastly, if $i = (0, l_v, d_v)$ then we partition the interval into intervals corresponding to different degrees of nodes, again of lengths $|I| \cdot \mathbb{P}(d_v|K_v, l_v)$.

By construction the interval assigned to $R_{K,C}$ has length $q(R_{K,C})$. Also, for two different clusters C_1, C_2, having different preceding contexts K_{C_1}, K_{C_2}, their intervals are disjoint. To see this consider the above procedure which assigns intervals and the first point where descriptions $R_{K_{C_1},C_1}$, R_{K_{C2},C_2} differ (there must be such point by Lemma 9). Observe that up to this point both clusters were assigned the same intervals. What is more already processed elements of $R_{K,C}$ uniquely define context (or label) for current vertex, i.e. when we want to partition by $\mathbb{P}(l_v|K_v)$, then all nodes on the path to v were already processed (this is due to the preorder ordering of nodes in description). This guarantees that, if the descriptions for two clusters C_1, C_2 were equal up to this point, then the interval will be partitioned in the same way for C_1 and C_2. At this point R_{C_1}, R_{C_2} will be assigned different, disjoint intervals.

Now we are ready to apply Lemma 6. By C_v and K_v we denote the cluster represented by v and context of this cluster in $\mathcal{T}$, respectively; additionally let $n = |\mathcal{T}|$.

$$
\begin{aligned}
|P|H_0(P) &\leq |R|H_0(R) \\
&\leq -\sum_{v \in \mathcal{T}'} \log q(R_{K_v,C_v}) && \text{by Lemma 6} \\
&= -\sum_{v \in \mathcal{T}'} \log \left(q(K_v) \cdot q(N_{C_v}) \cdot q(V_{C_v})\right) \\
&\leq -\sum_{v \in \mathcal{T}'} \log \left(\frac{1}{(k+1)\sigma^k} \cdot \frac{1}{m} \cdot q(V_{C_v})\right) \\
&= |\mathcal{T}'| \left(k \log \sigma + \log(k+1) + \log m\right) - \sum_{v \in \mathcal{T}'} \log \left(q(V_{C_v})\right) \\
&\leq -\sum_{v \in \mathcal{T}'} \log \left(q(V_{C_v})\right) + \mathcal{O}\left(\frac{nk \log \sigma}{m} + \frac{n \log m}{m}\right) .
\end{aligned}
$$

Now:

$$
-\sum_{v \in \mathcal{T}'} \log q(V_{C_v}) = -\sum_{u \in \mathcal{T}} \log q(u)
$$

$$
\begin{aligned}
&= -\sum_{\substack{u\in\mathcal{T}\\ u:\text{port}}} \log\left(\frac{1}{m}\cdot\mathbb{P}(l_v|K_v)\right) - \sum_{\substack{u\in\mathcal{T}\\ u:\text{regular}}} \log\left(\frac{m-1}{m}\cdot\mathbb{P}(l_v|K_v)\cdot\mathbb{P}(d_v|K_v,l_v)\right)\\
&\leq nH_k(L) + nH_k(\mathcal{T}|\mathcal{L}) + \mathcal{O}(|\mathcal{T}'|\log m) + n\log\frac{m}{m-1}\\
&\leq nH_k(L) + nH_k(\mathcal{T}|\mathcal{L}) + \mathcal{O}\left(\frac{n\log m}{m}\right) + \frac{n}{m-1}\log\left(1+\frac{1}{m-1}\right)^{m-1}\\
&\leq nH_k(L) + nH_k(\mathcal{T}|\mathcal{L}) + \mathcal{O}\left(\frac{n\log m}{m}\right) + \mathcal{O}\left(\frac{n}{m}\right) \quad ,
\end{aligned}
$$

which ends the proof for the Case 2. In the estimation we have used the fact that the total number of port nodes is at most $\mathcal{O}(n/m)$, since it cannot exceed the number of clusters.

The proof of Case 1 can be carried out in a similar manner, by replacing $\mathbb{P}(d_v|K_v,l_v)$ with $\mathbb{P}(d_v|K_v)$; alternatively it follows from Lemma 1.

The Case 3 requires slight modification of assignment of q to vertices, which reflects the different estimation:

$$
q(v) = \begin{cases} \frac{1}{m}\cdot\frac{1}{\sigma} & \text{, if } v=(1,l_v)\\ \frac{m-1}{m}\cdot\mathbb{P}(d_v|K_v)\cdot\mathbb{P}(l_v|K_v,d_v) & \text{, if } v=(0,l_v,d_v)\end{cases} .
$$

Now, to show that values q sum to at most one (i.e. they satisfy conditions of Lemma 6) the assignment of intervals must be changed, so that it reflects the current q: we first partition the interval by $\mathbb{P}(d_v|K_v)$ and then by $\mathbb{P}(l_v|K_v,d_v)$. The invariant is that when partitioning the interval for $\mathbb{P}(d_v|K_v)$ previous elements of $R_{K,C}$ uniquely determine K_v and when for $\mathbb{P}(l_v|K_v,d_v)$ then they uniquely determine K_v,d_v. However, this may not be true for port nodes: if v is a port node then we cannot extract the information on original degree of v from the cluster description alone. Hence we cannot partition the appropriate interval by $\mathbb{P}(d_v|K_v)$ for port nodes. This is why we have $q(v)=\frac{1}{m}\cdot\frac{1}{\sigma}$, instead of $\frac{1}{m}\cdot\mathbb{P}(d_v|K_v)$ for port nodes. This adds additional $\mathcal{O}\left(\frac{n\log\sigma}{m}\right)$ term (since there are at most $\mathcal{O}(n/m)$ port nodes), hence we have $\mathcal{O}\left(\frac{n(k+1)\log\sigma}{m}\right)$ in the third case. ◀

5 Application – succinct data structure for labeled trees

We demonstrate how our tree clustering technique can be used for compressed representation of labeled trees. Given a tree $\mathcal{T}$ we choose $m=\beta\log_\sigma|\mathcal{T}|$, for some constant β to be determined later. For appropriate β the number of different clusters is $\mathcal{O}(|\mathcal{T}|^{1-\alpha})$ for some constant $\alpha>0$, moreover for such m the cluster tree $\mathcal{T}'$ from $C(\mathcal{T})$ can be stored using any succinct representation (like balanced parentheses or DFUDS) in space $\mathcal{O}(|\mathcal{T}|/\log_\sigma|\mathcal{T}|)=o(|\mathcal{T}|)$ (for small enough σ). At the same time clusters are small enough so that we can preprocess them and answer all relevant queries within the clusters in constant time, the needed space is also $o(|\mathcal{T}|)$.

Let $\mathcal{T}'$ denote the unlabeled tree of $C(\mathcal{T})$, i.e. the cluster tree (C1) stripped of node labels. Our structure consists of:

- **(T1)** Unlabeled tree $\mathcal{T}'$, $|\mathcal{T}'|=\mathcal{O}(|\mathcal{T}|/\log_\sigma|\mathcal{T}|)$.
- **(T2)** String P obtained by concatenating labels of the cluster tree of $C(\mathcal{T})$ in preorder ordering.
- **(T3)** Degree sequences for each node of $\mathcal{T}'$.
- **(T4)** Precomputed arrays for each operation, for each cluster (along with look-up table from $C(\mathcal{T})$ to decode cluster structure from labels).

We encode each of (T1–T4) separately, using known tools.

(T1): Encoding tree $\mathcal{T}'$. There are many succinct representation for unlabeled trees which allow fast navigational queries [40, 2, 33, 12, 22]. They all use $2|\mathcal{T}'| + o(|\mathcal{T}'|)$ bits for tree of size $|\mathcal{T}'|$, the exact function suppressed by the $o(|\mathcal{T}'|)$ depends on the data structure. Since in our case the tree $\mathcal{T}'$ is already of size $\mathcal{O}(|\mathcal{T}|/\log_\sigma |\mathcal{T}|)$, we can use $\mathcal{O}(|\mathcal{T}'|)$ bits for the encoding of $\mathcal{T}'$, so we do not care about exact function hidden in $o(|\mathcal{T}'|)$. This is of practical importance, as the data structures with asymptotically smallest memory consumption, like [40], are very sophisticated, thus hard to implement and not always suitable for practical purposes. Thus we can choose theoretically inferior, but more practical data structure [2], we can even use a constant number of such data structures, as we are interested only in $\mathcal{O}(|\mathcal{T}'|)$ bound.

Choose one method, say [37], for the sake of argument. We use it to encode $\mathcal{T}'$ on $\mathcal{O}(|\mathcal{T}'|)$ bits, this encoding supports the following queries in constant time: parent(v) — parent of v; firstchild(v) — leftmost child of v; nextsibling(v) — right sibling of v; preorder-rank(v) — preorder rank of v; preorder-select(i) — returns a node whose preorder rank is i; lca(u, v) — returns the lowest common ancestor of u, v; childrank(v) — number of siblings preceding a node v; child(v, j) — j-th child of v; preorder_rank(v) — rank of node v in preorder ordering of nodes; preorder_select(i) — i-th node in preorder ordering of nodes; leaf_rank(v) — number of leaves to the left of u plus one ; leaf_select(i) — i-th leaf counting from the left; depth(v) — distance from the root to v; level_ancestor(v, i) — ancestor at distance i from v.

(T2): Encoding preorder sequence of labels. By Theorem 5 it is enough to encode the sequence P using roughly $|P|H_0(P)$ bits, in a way that allows for $\mathcal{O}(1)$ time access to its elements. This problem was studied extensively, and many (also practical) solutions were developed [23, 16, 26, 24]. Most of these methods are not overly complex: they assign a prefix code to consecutive groups of elements, concatenate prefix codes and use some simple structure for storing information, where the code words begin/end.

However, we must take into the account that alphabet of $|P|$ can be large (though, as shown later, not larger than $|\mathcal{T}|^{1-\alpha}$, $0 < \alpha < 1$). This renders some of previous results inapplicable, for example the simplest (and most practical) structure for alphabet of size σ' need additional $\mathcal{O}(|P| \log\log |P|/\log_{\sigma'} |P|)$ which can be as large as $\mathcal{O}(|P| \log\log |P|)$ [16]. As $|P| = |\mathcal{T}|/\log_\sigma |\mathcal{T}|$ this would be slightly above bound from Theorem 12: ($\mathcal{O}(|\mathcal{T}| \log\log |\mathcal{T}|/\log_\sigma |\mathcal{T}|)$ vs $\mathcal{O}(|\mathcal{T}| \log\log_\sigma |\mathcal{T}|/\log_\sigma |\mathcal{T}|)$).

Still, there are structures achieving $|P|H_0(P) + o(|P|)$ bits for alphabets of size $|\mathcal{T}|^{1-\alpha}$, for example the well-known one by Pătraşcu [41, Theorem 1].

(T3): Encoding the degree sequence. To navigate the tree, we need to know which children of a cluster belong to which port node. We do it by storing degree sequence for each cluster node of $\mathcal{T}'$ and design a structure which, given a node u' of $\mathcal{T}'$ and index of a port node u in the cluster represented by u', returns the range of children of u' which contain children of u in $\mathcal{T}$. We do it by storing rank/select structure for bitvector representing degrees of vertices of $\mathcal{T}'$ in unary.

▶ **Lemma 10.** *We can encode all degree sequences of nodes of $\mathcal{T}'$ using $\mathcal{O}(|\mathcal{T}'|)$ bits in total, such that given node u of $\mathcal{T}'$ and index of port node v in cluster represented by u the structure returns a pair of indices i_1, i_2, such that the children of v (in $\mathcal{T}$) are exactly the roots of trees in clusters in children $i_1, i_1 + 1, \ldots, i_2 - 1$ of u. Moreover we can answer reverse queries, that is, given an index x of x-th child of $u \in \mathcal{T}'$ find port node which connects to this child. Both operations take $\mathcal{O}(1)$ time.*

(T4): Precomputed tables. We want to answer all queries in each cluster in constant time, we start by showing that there are not many clusters of given size.

▶ **Lemma 11.** *There are at most* $2^{2m'}\sigma^{m'}$ *different clusters of size* m'.

It is sufficient to choose m in clustering as $m = \frac{1}{8}\log_\sigma |\mathcal{T}|$ (or 1 if this is smaller than 1) assuming that $2 \leq \sigma \leq |\mathcal{T}|^{1-\alpha}$, $\alpha > 0$, then the number of different clusters of size at most $2m-1$ is $\sum_{i=1}^{2m-1} 2^{2i}\sigma^i \leq \mathcal{O}(|\mathcal{T}|^{1-\alpha})$.

We precompute and store the answers for each query for each cluster. As every query takes constant number of arguments and each argument ranges over m values, this uses at most $\mathcal{O}(|\mathcal{T}|^{1-\alpha}) \cdot \mathcal{O}(\log_\sigma^c |\mathcal{T}|) = o(|\mathcal{T}|)$ bits, where c is a constant. Additionally we make tables for accessing i-th (in left-to-right order on leaves) port node of each cluster, as we will need this later to support more involved queries.

Putting it all together. The above structures can be combined into a succinct data structure for trees. Note that we used the fact that $\sigma \leq |\mathcal{T}|^{1-\alpha}$, we generalize for arbitrary σ in the full version.

▶ **Theorem 12.** *Let* $\mathcal{T}$ *be a labeled tree with labels from an alphabet of size* $\sigma \leq |\mathcal{T}|^{1-\alpha}, \alpha > 0$. *Then we can build in linear time a tree structure whose bit size is bounded by* all *of the below values:*

$$|\mathcal{T}|H(\mathcal{T}) + |\mathcal{T}|H_k(L) + \mathcal{O}\left(\frac{|\mathcal{T}|k\log\sigma}{\log_\sigma|\mathcal{T}|} + \frac{|\mathcal{T}|\log\log_\sigma|\mathcal{T}|}{\log_\sigma|\mathcal{T}|}\right)$$
$$|\mathcal{T}|H_k(\mathcal{T}|L) + |\mathcal{T}|H_k(L) + \mathcal{O}\left(\frac{|\mathcal{T}|k\log\sigma}{\log_\sigma|\mathcal{T}|} + \frac{|\mathcal{T}|\log\log_\sigma|\mathcal{T}|}{\log_\sigma|\mathcal{T}|}\right)$$
$$|\mathcal{T}|H(\mathcal{T}) + |\mathcal{T}|H_k(L|\mathcal{T}) + \mathcal{O}\left(\frac{|\mathcal{T}|(k+1)\log\sigma}{\log_\sigma|\mathcal{T}|} + \frac{|\mathcal{T}|\log\log_\sigma|\mathcal{T}|}{\log_\sigma|\mathcal{T}|}\right)$$

It supports firstchild(u), parent(u), nextsibling(u), lca(u,v), childrank(u), child(u,i), depth(u), preorder_rank(u), preorder_select(i), leaf_rank(u) *and* leaf_select(i). *operations in* $\mathcal{O}(1)$ *time; at the expense of additional* $\mathcal{O}(|\mathcal{T}|(\log\log|\mathcal{T}|)^2/\log_\sigma|\mathcal{T}|)$ *bits it can support* level_ancestor(v, i) *query in* $\mathcal{O}(1)$ *time.*

The main idea of Theorem 12 is that for each query if both arguments and the answer are in the same cluster then we can use precomputed tables, for other we can query the structure for $\mathcal{T}'$ and reduce it to the former case using previously defined structures. A similar idea was used in other tree partition based structures like [22] (yet this solution used different tree partition method).

6 Even succincter structure

So far we have obtained the redundancy of $\mathcal{O}(|\mathcal{T}|\log\log_\sigma|\mathcal{T}|/\log_\sigma|\mathcal{T}|) + \mathcal{O}(|\mathcal{T}|k\log\sigma/\log_\sigma|\mathcal{T}|)$. As the recent lower bound for zeroth-order entropy coding of a string partition [20] (assuming certain partition properties) also applies to trees, we can conclude that our structure in worst case requires $\Omega(|\mathcal{T}|k\log\sigma/\log_\sigma|\mathcal{T}|)$ additional bits. Yet, this lower bound only says that the above factor is necessary when zeroth-order entropy coder is used, not that this factor is required in general. Indeed, for strings, there are methods of compressing the text S (with fast random access) using $|S|H_k(S) + o(|S|) + f(k,\sigma)$ bits [26, 38], also methods related to compression boosting achieve $|S|H_k(S) + \mathcal{O}(|S|) + f(k,\sigma)$ bits [15], where $f(k,\sigma)$ is some function that depends only on k and σ. Similarly, the method of compressing the tree

using combination of XBWT and compression boosting gives redundancy of $\mathcal{O}(|S|) + f(k, \sigma)$ bits [13]. In all of the above cases $f(k, \sigma)$ can be bounded by $\mathcal{O}(\sigma^k \cdot \text{polylog}(\sigma))$. This is more desirable than $\mathcal{O}(|\mathcal{T}|k \log \sigma / \log_\sigma |\mathcal{T}|)$, as in many applications k and σ are fixed and so this term is constant, moreover the redundancy is a sum of two functions instead of a product. Furthermore, achieving such redundancy allows us to relax our assumptions, i.e. we obtain additional $o(|\mathcal{T}|)$ factor for $k = \alpha \log_\sigma |\mathcal{T}|, 0 < \alpha < 1$, while so far our methods only gave $o(|\mathcal{T}| \log \sigma)$ for $k = o(\log_\sigma |\mathcal{T}|)$.

We can decrease the redundancy to $\mathcal{O}(\sigma^k \cdot \text{polylog}(\sigma))$ at the cost of increasing the query time to $\mathcal{O}(\log n / \log \log n)$ (note that previously mentioned compressed text storages [26] also did not support constant access). The proof of Theorem 5 suggests that we lose up to $k \log \sigma$ bits per cluster (as a remainder: we assign each cluster value q and we "pay" $\log q$ bits), so it seems to be the bottleneck of our solution. In case of text compression [26] improvements were obtained by partitioning the text into blocks and encoding string made of blocks of size $\Theta(\log_\sigma n)$ with first order entropy coder; to support retrieval in time $\mathcal{O}(d)$, for some d, every d-th block was stored explicitly. For $d = \log |S| / \log \log |S|$ and assuming that each block has at most $\log_\sigma |S|$ characters this gives $\mathcal{O}(|S| \log \log |S| / \log_\sigma |S|)$ bits of redundancy.

We would like to generalize this idea to labeled trees, yet there are two difficulties: first, the previously mentioned solution for strings required that context of each block is stored wholly in some previous block, second, as there is no linear order on clusters, we do not know how to choose $|\mathcal{T}'|/d$ clusters. Our approach for solving this problem combines ideas from both compression boosting techniques [15] and for compressed text representation [26]: for $\mathcal{O}(|\mathcal{T}'|/d)$ nodes, where $d = \log |\mathcal{T}| / \log \log |\mathcal{T}|$, we store the context explicitly using $k \log \sigma$ bits. The selection of nodes is simple: by counting argument for some $1 \leq i < d$ the tree levels $i, i + d, \ldots i + dj$ of the cluster tree $\mathcal{T}'$ have at most $|\mathcal{T}'|/d$ nodes. This allows to retrieve context of each cluster by traversing (first up and then down) at most d nodes. Then we partition the clusters in the classes depending on their preceding context and use zeroth order entropy for each class (similarly to compression boosting [15] or some text storage methods [23]), i.e. we encode each cluster as if we knew its preceding context. To decode, we first retrieve the context and next decode zeroth order code from given class.

▶ **Theorem 13.** *Let $\mathcal{T}$ be a labeled tree and $k = \alpha \log_\sigma |\mathcal{T}|$, for a constant $0 < \alpha < 1$. Then we can build a tree structure which requires a number of bits bounded by* all *of the chosen value from the list below:*

- $|\mathcal{T}|H(\mathcal{T}) + |\mathcal{T}|H_k(L) + \mathcal{O}\left(\frac{|\mathcal{T}| \log \log |\mathcal{T}|}{\log_\sigma |\mathcal{T}|}\right)$;
- $|\mathcal{T}|H_k(\mathcal{T}|L) + |\mathcal{T}|H_k(L) + \mathcal{O}\left(\frac{|\mathcal{T}| \log \log |\mathcal{T}|}{\log_\sigma |\mathcal{T}|}\right)$;
- $|\mathcal{T}|H(\mathcal{T}) + |\mathcal{T}|H_k(L|\mathcal{T}) + \mathcal{O}\left(\frac{|\mathcal{T}| \log \sigma}{\log_\sigma |\mathcal{T}|} + \frac{|\mathcal{T}| \log \log |\mathcal{T}|}{\log_\sigma |\mathcal{T}|}\right)$.

This data structure supports the following operations in $\mathcal{O}(\log |\mathcal{T}| / \log \log |\mathcal{T}|)$ time: firstchild(u), parent(u), nextsibling(u), lca(u, v), child(v, i), childrank(u), depth(u), preorder_rank(u), preorder_select(i), leaf_rank(u) *and* leaf_select(i). *It can support* level_ancestor(v, i) *query in $\mathcal{O}(\log |\mathcal{T}| / \log \log |\mathcal{T}|)$ time with additional $\mathcal{O}(|\mathcal{T}|(\log \log |\mathcal{T}|)^2 / \log_\sigma |\mathcal{T}|)$ bits.*

7 Label-related operations

Using additional memory we can support some label-related operations previously considered for succinct trees [44, 28]. Even though we do not support all of them, in all cases we support at least the same operations as XBWT, i.e. childrank(v, a) (which returns v's rank among

children labeled with a) and $\mathsf{childselect}(v, i, a)$ (which returns the ith child of v labeled with a). To this end we employ $\mathsf{rank}/\mathsf{select}$ structures for large alphabets [3, 4]. Note, that we do not have constant time for every alphabet size and required additional space is larger than for other operations (i.e. $o(|\mathcal{T}| \log \sigma)$), but this is unavoidable [4, 25]. Moreover, as the last point of Theorem 14 use structures which are not state-of-the-art, it is likely that better structures [4] are applicable, but they are more involved and it is not clear if they are compatible with our tree storage methods.

▶ **Theorem 14.** *We can augment structures from Theorem 12 and Theorem 13 so that:*

- *For $\sigma = \mathcal{O}(1)$ we can perform:* $\mathsf{childrank}(v, a)$, $\mathsf{childselect}(v, i, a)$, $\mathsf{level\_ancestor}(v, i, a)$, $\mathsf{depth}(v, a)$ *for structure from Theorem 12 in $\mathcal{O}(1)$ time and for structure from Theorem 13 in $\mathcal{O}(\log |\mathcal{T}| / \log\log |\mathcal{T}|)$ time using asymptotically the same additional memory.*
- *For $\sigma = \mathcal{O}(\log^{1+o(1)} |\mathcal{T}|)$ and $\sigma = \omega(1)$ we can perform:* $\mathsf{childrank}(v, a)$ *and* $\mathsf{childselect}(v, i, a)$ *for structure from Theorem 12 in $\mathcal{O}(1)$ time and for structure from Theorem 13 in $\mathcal{O}(\log |\mathcal{T}| / \log\log |\mathcal{T}|)$ time using additional $o(|\mathcal{T}| \log \sigma)$ bits.*
- *For arbitrary σ we can perform:* $\mathsf{childrank}(v, a)$ *and* $\mathsf{childselect}(v, i, a)$ *for structure from Theorem 12 in $\mathcal{O}(\log \log^{1+\epsilon} \sigma)$ time and for structure from Theorem 13 in $\mathcal{O}((\log \log^{1+\epsilon} \sigma) \log |\mathcal{T}| / \log\log |\mathcal{T}|)$ time; using additional $o(|\mathcal{T}| \log \sigma)$ bits.*

8 Open problems

There are a few open questions. First, can our analysis be applied to recently developed dictionary compression methods for trees like Top-Trees/Top-Dags [8, 29] or other dictionary based methods on trees? Related is the problem of finding good compression measures for repetitive trees, for instance for the case of text we have LZ77 and BWT-run, for which we can build efficient structures (like text indices) based on this representations [36, 17] *and* find relation with information-theoretic bounds (like k-th order entropy) or even show that they are close in information-theoretic sense to each other [35]. Even though we have tree representation like LZ77 for trees [21] or tree grammars [34] we do not know relation between them nor how they correspond to tree entropy measures (with the exception of recent paper on tree grammars [30], yet this approach works only in the restricted case of binary trees and the presented definition of tree entropy is different and only valid for the binary trees). One could also measure tree repetitiveness with number of runs (i.e. number of phrases in run-length encoding) in string generated by some linearization of the tree. The XBWT seems to not be a good choice in this case, because, as mentioned before, it does not capture the tree shape, moreover even on repetitive tree (i.e. trees with many repeated subtrees) XBWT does not contain many long runs. Still, it would be interesting to develop new linearization techniques or improve XBWT so it would apply for repetitive trees.

Second, we do not support all of the label-related operations, moreover we do not achieve optimal query times. The main challenge is to support more complex operations, like labeled level_ancestor, while achieving theoretical bounds considered in this work. Previous approaches partitioned (in a rather complex way) the tree into subtrees, by node labels (i.e. one subtree contained only nodes with same label) and achieved at most zeroth-order entropy of labels [44, 28], it may be possible to combine these methods with ours.

Next, it should be possible that the presented structure can be made to support dynamic trees, as all of the used structures have their dynamic equivalent [26, 7, 40]. Still, it is not entirely trivial as we need to maintain the clustering.

Next, as Ferragina et al. [13] mentioned, in some applications nodes can store strings, rather than single labels, where the context for a letter is defined by labels of ancestor nodes and previous letters in a node. It seems that our method should apply in this scenario, contrary to XBWT.

Finally, can the additional space required for level_ancestor query be lowered? We believe that this is the case, as our solution did not use the fact that all weights sum up to $|\mathcal{T}|$; moreover it should be possible to apply methods from [40] to obtain $\mathcal{O}(|\mathcal{T}| \log\log|\mathcal{T}|/\log_\sigma|\mathcal{T}|)$ additional space for level_ancestor.

References

1 Janos Aczél. On Shannon's inequality, optimal coding, and characterizations of Shannon's and Rényi's entropies. In *Symposia Mathematica*, volume 15, pages 153–179, 1973.

2 Diego Arroyuelo, Rodrigo Cánovas, Gonzalo Navarro, and Kunihiko Sadakane. Succinct trees in practice. In *2010 Proceedings of the Twelfth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 84–97. SIAM, 2010.

3 Jérémy Barbay, Meng He, J. Ian Munro, and S. Srinivasa Rao. Succinct indexes for strings, binary relations and multi-labeled trees. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 680–689. SIAM, 2007. URL: `http://dl.acm.org/citation.cfm?id=1283383.1283456`.

4 Djamal Belazzougui and Gonzalo Navarro. Optimal lower and upper bounds for representing sequences. *ACM Trans. Algorithms*, 11(4):31:1–31:21, April 2015. `doi:10.1145/2629339`.

5 Michael A Bender and Martın Farach-Colton. The level ancestor problem simplified. *Theoretical Computer Science*, 321(1):5–12, 2004.

6 David Benoit, Erik D Demaine, J Ian Munro, Rajeev Raman, Venkatesh Raman, and S Srinivasa Rao. Representing trees of higher degree. *Algorithmica*, 43(4):275–292, 2005.

7 Philip Bille, Anders Roy Christiansen, Nicola Prezza, and Frederik Rye Skjoldjensen. Succinct partial sums and fenwick trees. In Gabriele Fici, Marinella Sciortino, and Rossano Venturini, editors, *String Processing and Information Retrieval - 24th International Symposium, SPIRE 2017, Palermo, Italy, September 26-29, 2017, Proceedings*, volume 10508 of *Lecture Notes in Computer Science*, pages 91–96. Springer, 2017. `doi:10.1007/978-3-319-67428-5_8`.

8 Philip Bille, Inge Li Gørtz, Gad M Landau, and Oren Weimann. Tree compression with top trees. In *International Colloquium on Automata, Languages, and Programming*, pages 160–171. Springer, 2013.

9 Giorgio Busatto, Markus Lohrey, and Sebastian Maneth. Efficient memory representation of XML document trees. *Information Systems*, 33(4–5):456–474, 2008.

10 R. D. Cameron. Source encoding using syntactic information source models. *IEEE Transactions on Information Theory*, 34(4):843–850, July 1988. `doi:10.1109/18.9782`.

11 Arash Farzan and J Ian Munro. A uniform paradigm to succinctly encode various families of trees. *Algorithmica*, 68(1):16–40, 2014.

12 Arash Farzan, Rajeev Raman, and S Srinivasa Rao. Universal succinct representations of trees? In *International Colloquium on Automata, Languages, and Programming*, pages 451–462. Springer, 2009.

13 Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S Muthukrishnan. Structuring labeled trees for optimal succinctness, and beyond. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 184–193. IEEE, 2005.

14 Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S Muthukrishnan. Compressing and indexing labeled trees, with applications. *Journal of the ACM (JACM)*, 57(1):4, 2009.

15 Paolo Ferragina and Giovanni Manzini. Compression boosting in optimal linear time using the Burrows-Wheeler transform. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, pages 655–663, Philadelphia, PA, USA, 2004. Society for

Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=982792.982892.

16 Paolo Ferragina and Rossano Venturini. A simple storage scheme for strings achieving entropy bounds. *Theor. Comput. Sci.*, 372(1):115–121, 2007. doi:10.1016/j.tcs.2006.12.012.

17 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Optimal-time text indexing in BWT-runs bounded space. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 1459–1477, Philadelphia, PA, USA, 2018. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=3174304.3175401.

18 Moses Ganardi, Danny Hucke, Artur Jeż, Markus Lohrey, and Eric Noeth. Constructing small tree grammars and small circuits for formulas. *Journal of Computer and System Sciences*, 86:136–158, 2017.

19 Moses Ganardi, Danny Hucke, Markus Lohrey, and Eric Noeth. Tree compression using string grammars. *Algorithmica*, 80(3):885–917, 2018. doi:10.1007/s00453-017-0279-3.

20 Michał Gańczorz. Entropy lower bounds for dictionary compression. In *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019, June 18-20, 2019, Pisa, Italy.*, pages 11:1–11:18, 2019. doi:10.4230/LIPIcs.CPM.2019.11.

21 Paweł Gawrychowski and Artur Jeż. LZ77 factorisation of trees. In Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen, editors, *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, December 13-15, 2016, Chennai, India*, volume 65 of *LIPIcs*, pages 35:1–35:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.35.

22 Richard F Geary, Rajeev Raman, and Venkatesh Raman. Succinct ordinal trees with level-ancestor queries. *ACM Transactions on Algorithms (TALG)*, 2(4):510–534, 2006.

23 Rodrigo González and Gonzalo Navarro. Statistical encoding of succinct data structures. In Moshe Lewenstein and Gabriel Valiente, editors, *Combinatorial Pattern Matching, 17th Annual Symposium, CPM 2006, Barcelona, Spain, July 5-7, 2006, Proceedings*, volume 4009 of *Lecture Notes in Computer Science*, pages 294–305. Springer, 2006. doi:10.1007/11780441_27.

24 Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 841–850. Society for Industrial and Applied Mathematics, 2003.

25 Roberto Grossi, Alessio Orlandi, and Rajeev Raman. Optimal trade-offs for succinct string indexes. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part I*, volume 6198 of *Lecture Notes in Computer Science*, pages 678–689. Springer, 2010. doi:10.1007/978-3-642-14165-2_57.

26 Roberto Grossi, Rajeev Raman, Srinivasa Rao Satti, and Rossano Venturini. Dynamic compressed strings with random access. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, pages 504–515. Springer, 2013. doi:10.1007/978-3-642-39206-1_43.

27 Meng He, J Ian Munro, and S Srinivasa Rao. Succinct ordinal trees based on tree covering. In *International Colloquium on Automata, Languages, and Programming*, pages 509–520. Springer, 2007.

28 Meng He, J Ian Munro, and Gelin Zhou. A framework for succinct labeled ordinal trees over large alphabets. *Algorithmica*, 70(4):696–717, 2014.

29 Lorenz Hübschle-Schneider and Rajeev Raman. Tree compression with top trees revisited. In *International Symposium on Experimental Algorithms*, pages 15–27. Springer, 2015.

30 D. Hucke, M. Lohrey, and L. S. Benkner. Entropy bounds for grammar-based tree compressors. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 1687–1691, July 2019. doi:10.1109/ISIT.2019.8849372.

31 Guy Jacobson. Space-efficient static trees and graphs. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 549–554. IEEE, 1989.

32 Guy Joseph Jacobson. *Succinct Static Data Structures*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1988. AAI8918056.

33 Jesper Jansson, Kunihiko Sadakane, and Wing-Kin Sung. Ultra-succinct representation of ordered trees. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 575–584. Society for Industrial and Applied Mathematics, 2007.

34 Artur Jeż and Markus Lohrey. Approximation of smallest linear tree grammar. *Inf. Comput.*, 251:215–251, 2016. doi:10.1016/j.ic.2016.09.007.

35 Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: String attractors. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 827–840, New York, NY, USA, 2018. ACM. doi:10.1145/3188745.3188814.

36 Sebastian Kreft and Gonzalo Navarro. Self-indexing based on LZ77. In *Annual Symposium on Combinatorial Pattern Matching*, pages 41–54. Springer, 2011.

37 Hsueh-I Lu and Chia-Chi Yeh. Balanced parentheses strike back. *ACM Transactions on Algorithms (TALG)*, 4(3):28, 2008.

38 J. Ian Munro and Yakov Nekrich. Compressed data structures for dynamic sequences. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 891–902, 2015. doi:10.1007/978-3-662-48350-3_74.

39 J Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses, static trees and planar graphs. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 118–126. IEEE, 1997.

40 Gonzalo Navarro and Kunihiko Sadakane. Fully functional static and dynamic succinct trees. *ACM Transactions on Algorithms (TALG)*, 10(3):16, 2014.

41 Mihai Pătraşcu. Succincter. In *FOCS'08. IEEE 49th Annual IEEE Symposium on Foundations of Computer Science, 2008.*, pages 305–313. IEEE, 2008.

42 Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms (TALG)*, 3(4):43, 2007.

43 Rajeev Raman and Satti Srinivasa Rao. Succinct dynamic dictionaries and trees. In *International Colloquium on Automata, Languages, and Programming*, pages 357–368. Springer, 2003.

44 Dekel Tsur. Succinct representation of labeled trees. *Theoretical Computer Science*, 562:320–329, 2015.

A Additional proofs for Section 2

Proof of Lemma 1. The proof follows by straightforward application of the log sum inequality. To prove the case for $|\mathcal{T}|H_k(\mathcal{T}|L)$ we use the fact that $\sum_l \sum_K t_{K,l} = |\mathcal{T}|$ and that for a fixed d we have $\sum_K \sum_l t_{K,l,d} = t_d$.

$$\begin{aligned}
|\mathcal{T}|H_k(\mathcal{T}|L) &= -\sum_{v\in\mathcal{T}} \log \mathbb{P}(d_v|K_v, l_v) \\
&= -\sum_{v\in\mathcal{T}} \log \frac{t_{K_v,l_v,d_v}}{t_{K,l_v}} \\
&= -\sum_d \sum_K \sum_l t_{K,l,d} \log \frac{t_{K,l,d}}{t_{K,l}} \\
&\leq -\sum_d t_d \log \frac{t_d}{|\mathcal{T}|}
\end{aligned}$$

The proof for the case $|\mathcal{T}|H_k(L|\mathcal{T})$ is analogous. ◀

B Additional proofs for Section 3

Proof of Lemma 2. We build the clustering by a simple dfs-based method, starting at the root r.

For a node v, the procedure returns a tree c_v rooted in v along with its size (and also creates some clusters). The actions on v are as follows: we recursively call the procedure on v's children $v_1, \ldots, v_j$, let the returned trees be $c_1, \ldots, c_j$ and their sizes $s_1, \ldots, s_j$. We have two possibilities:

- $\sum_{i=1}^{j} s_i < m$, then return a tree rooted at v with all of the returned trees $c_1, \ldots, c_j$ rooted at its children.
- $\sum_{i=1}^{j} s_i \geq m$, then we group returned trees greedily: We process trees from left to right, at the beginning we create cluster containing $C = \{c_1\}$, while $|C| < m$ we add consecutive c_i's to C (recall that $|C|$ denotes the number of nodes in trees in C). Then at some point we must add c_j such that $|C| \geq m$. We output the current cluster C, set $C = \{c_{j+1}\}$ and continue grouping the trees. At the end we return tree containing just one vertex: v.

Finally, we make a cluster of the tree returned by the root of $\mathcal{T}$, regardless of its size.

We now show that the above algorithm satisfies (C1–C4).

Observe that the above procedure always returns a tree, its size is at most m: either it is only a vertex v (in the second case) or the sum of sizes of subtrees is at most $m-1$, plus 1 for the v (in the first case). This implies (C2): when we make a cluster out of $c_i, \ldots, c_{j+1}$ then $\sum_{\ell=i}^{j} s_\ell$ is at most $m-1$ by the algorithm and $s_{j+1} \leq m$ by the earlier observation.

It is easy to see from the algorithm that a cluster contains trees rooted at the consecutive siblings, so (C3) holds.

By an easy induction we can also show that in each returned tree the degree of the node v is either 0 (in the second case, when we return tree which consists of only v) or equal to the degree in the input tree $\mathcal{T}$ (in the first case, when the tree contains v and all of its children), thus (C4) holds.

Concerning the total number of clusters, first recall that they are of size at most $2m-1$, thus there are at least $\frac{|\mathcal{T}|}{2m-1} \geq \frac{|\mathcal{T}|}{2m} - 1$ clusters. To upper bound the number of clusters observe that by the construction the clusters are of size at least m except two cases: the cluster rooted at the root of the whole tree and the clusters that include the last child of the node (but not the node, i.e. they are created in the second case). For the former, there is at most one such a cluster. For the latter, before creating such cluster we created at least one other cluster from trees rooted in siblings whose size is at least m, thus for each cluster of size less than m there must exist corresponding (previously created) cluster with size at least m. Hence the number of clusters of size smaller than m is at most half the number of total clusters plus one, thus there are at most $2\frac{|\mathcal{T}|}{m} + 1$ clusters, as claimed in (C1). ◀

Proof of Lemma 4. Given a tree $\mathcal{T}$ we build a cluster of its nodes using procedure from Lemma 2, create a node for each cluster and add an edge between two nodes u and v if and only if in $\mathcal{T}$ there was an edge from some vertex from cluster C_u to some vertex in C_v. We label the cluster nodes consistently, i.e. u and v get the same label if and only if their clusters are identical, also in the sense which nodes are port nodes (note though, that the port nodes can have different degree in the input tree). For each label representing the cluster we store its cluster. For simplicity, we assume that the assigned labels are from an ordered set (i.e. set of numbers).

As mentioned before, we store the previously defined degree sequence. ◀

C Additional proofs for Section 4

Proof of Lemma 9. It is a known fact that from sequence of degrees in preorder ordering we can retrieve shape of the tree (we can do it by simple dfs-procedure, which first creates node with given degree, then calls itself recursively; when we recurse back we know which node is next, etc.). From cluster description we can retrieve the sequence of its degrees in preorder ordering. We also can retrieve labels and information which nodes are port. Now for each vertex v in cluster we know its original context K_v in T, as we explicitly store context for roots of trees, and other nodes have their context either fully in cluster, or their context is concatenation of some suffix of K and some path in the cluster. ◀

D Additional material for Section 5

Proof of Lemma 10. First we concatenate degree sequences for each node in $\mathcal{T}'$, according to preorder ordering, obtaining a sequence $D = d_{v_1,1}, \dots d_{v_1,j_1}, d_{v_2,1}, \dots d_{v-2,j_2}, d_{v_{|\mathcal{T}'|},1}, \dots d_{v_{|\mathcal{T}'|},j_{|\mathcal{T}'|}}$. Sum of all $d_{v,j}$'s in the sequence is bounded by $|\mathcal{T}'|$, as each $d_{v,j}$ corresponds to $d_{v,j}$ edges. We encode each number in the sequence in unary: $D_u = 0^{d_{v_1,1}}1 \dots 0^{d_{v_{|\mathcal{T}'|},j_{|\mathcal{T}'|}}}1$. Then we build a separate sequence B_u, which marks the borders between nodes in the degree sequence, i.e. $B_u[z] = 1$ if and only if at index z starts the unary degree sequence of some node.

Consider the following example: for nodes a, b, c, d, e and corresponding degree sequences $(0), (3, 1), (2), (1, 2), (2, 2)$ we have (the vertical lines | denote borders of degree sequences and are added for increased readability):

$$D_u = 1|000101|001|01001|001001$$
$$B_u = 1|100000|100|10000|100000$$

We now construct rank/select data structure for D_u and B_u. There are multiple approaches that, for static bitvectors, use $\mathcal{O}(|D_u| + |B_u|) = \mathcal{O}(|\mathcal{T}'|)$ bits and allow both operations in $\mathcal{O}(1)$ time [42].

We describe how to answer a query for node $u \in \mathcal{T}'$ and port node v in the cluster. Let p_u be preorder index of u.

Let j be the point in D_u where the degree sequence of u starts, i.e. $j = select_1(p_u, B_u)$. Let p_v be a port index of v in the u-cluster (recall that we ordered port nodes in each cluster in left-to-right order on leaves). We want to find the beginning of unary description of d_{u,p_v} (plus one) in D_u: this is the $p_v - 1$-th 1 starting from j-th element in D_u. The next 1 corresponds to the end of unary description. Let u_1, u_2 be the beginning and end of this unary description, we can find them in the following way: $u_1 = select_1(rank_1(j, D_u) + p_v - 1, B_u) + 1$ and $u_2 = select_1(rank_1(j, D_u) + p_v, B_u) - 1$. Observe now that number of 0's in D_u between j and u_1 (with j and u_1) is equal to i_1. Similarly we can get i_2 by counting zeroes between u_1 and u_2. Thus $i_1 = rank_0(u_1, D_u) - rank_0(j)$ and $i_2 = rank_0(u_2, D_u) - rank_0(u_1, D_u) - 1$.

We now proceed to the second query. We find the beginning of description in unary, denoted j as above. We find position u_x of x-th 0 counting from j, we do it by calling $u_x = select_0(rank_0(j) + x)$. Now we calculate the numbers of 1's between j and u_x and simply return this value (+1). That is, we return $rank_1(u_x) - rank_1(j) + 1$. ◀

Proof of Lemma 11. Each cluster can be represented by: a number of nodes in the cluster, written as a unary string of length $m' + 1$, a bitvector indicating which nodes are port nodes of length m', balanced parentheses representation of cluster structure, string of labels of length m'. ◀

We give more general version of Theorem 12.

▶ **Theorem 15** (Full version of Theorem 12)**.** *Let $\mathcal{T}$ be a labeled tree with labels from an alphabet of size $\sigma \leq |\mathcal{T}|^{1-\alpha}, \alpha > 0$. Then we can build a tree structure which requires a number of bits bounded by* all *of the chosen value from the list below:*

- $|\mathcal{T}|H(\mathcal{T}) + |\mathcal{T}|H_k(L) + \mathcal{O}\left(\frac{|\mathcal{T}|k\log\sigma}{\log_\sigma|\mathcal{T}|} + \frac{|\mathcal{T}|\log\log_\sigma|\mathcal{T}|}{\log_\sigma|\mathcal{T}|}\right)$;
- $|\mathcal{T}|H_k(\mathcal{T}|L) + |\mathcal{T}|H_k(L) + \mathcal{O}\left(\frac{|\mathcal{T}|k\log\sigma}{\log_\sigma|\mathcal{T}|} + \frac{|\mathcal{T}|\log\log_\sigma|\mathcal{T}|}{\log_\sigma|\mathcal{T}|}\right)$;
- $|\mathcal{T}|H(\mathcal{T}) + |\mathcal{T}|H_k(L|\mathcal{T}) + \mathcal{O}\left(\frac{|\mathcal{T}|(k+1)\log\sigma}{\log_\sigma|\mathcal{T}|} + \frac{|\mathcal{T}|\log\log_\sigma|\mathcal{T}|}{\log_\sigma|\mathcal{T}|}\right)$.

For general σ we can build the structure which size is bounded by any of the values below:

- $|\mathcal{T}|H(\mathcal{T}) + |\mathcal{T}|H_k(L) + \mathcal{O}\left(\frac{|\mathcal{T}|k\log\sigma}{\log_\sigma|\mathcal{T}|} + \frac{|\mathcal{T}|\log\log|\mathcal{T}|}{\log_\sigma|\mathcal{T}|}\right)$;
- $|\mathcal{T}|H_k(\mathcal{T}|L) + |\mathcal{T}|H_k(L) + \mathcal{O}\left(\frac{|\mathcal{T}|k\log\sigma}{\log_\sigma|\mathcal{T}|} + \frac{|\mathcal{T}|\log\log|\mathcal{T}|}{\log_\sigma|\mathcal{T}|}\right)$;
- $|\mathcal{T}|H(\mathcal{T}) + |\mathcal{T}|H_k(L|\mathcal{T}) + \mathcal{O}\left(\frac{|\mathcal{T}|(k+1)\log\sigma}{\log_\sigma|\mathcal{T}|} + \frac{|\mathcal{T}|\log\log|\mathcal{T}|}{\log_\sigma|\mathcal{T}|}\right)$.

It supports firstchild(u)*,* parent(u)*,* nextsibling(u)*,* lca(u, v)*,* childrank(u)*,* child(u,i) *and* depth(u) *operations in $\mathcal{O}(1)$ time; moreover with additional $\mathcal{O}(|\mathcal{T}|(\log\log|\mathcal{T}|)^2/\log_\sigma|\mathcal{T}|)$ bits it can support* level_ancestor(v, i) *query in $\mathcal{O}(1)$ time.*

Proof of Theorem 12 (and 15). We start by proving the part for operations firstchild(u), parent(u), nextsibling(u), lca(u, v), as the rest requires additional structures.

First we consider the case for $\sigma \leq |\mathcal{T}|^{1-\alpha}$.

To bound the memory consumption we sum needed space for (T1–T4). (T1), (T3) and (T4) take at most $\mathcal{O}(|\mathcal{T}'|) = \mathcal{O}(\frac{|\mathcal{T}|}{\log_\sigma|\mathcal{T}|})$ bits. We bound space for (T2) by Theorem 5, observe that this theorem gives us the same bound as in the claim, moreover this summand dominates bounds for (T1), (T3) and (T4).

One of the crucial part while performing operations on our compressed tree structures is that we can perform preorder-rank and preorder-select in constant time, this allows us to retrieve tree node labels from preorder sequence P of $C(\mathcal{T})$ (and thus to retrieve the cluster), given node of $\mathcal{T}'$, in constant time.

We now give the description of operations, let u denote the node and u' the name of its cluster. If the answer can be calculated using only the cluster of u (i.e. when u and the answer is in the same cluster) we return the answer using precomputed tables. Thus in the following we give the description when the answer cannot be computed within the cluster u' alone.

firstchild(u): Using the structure for degree sequence (T3) we find index i of child of u' which represents cluster containing first child of u. We call child(u, i) on the structure for unlabeled tree $\mathcal{T}'$ (T1), to get this cluster, the answer is the root of the first tree in this cluster.

parent(u): We call childrank(u') on structure for $\mathcal{T}'$. This gives us index i such that u' is i-th node of node v' representing the cluster containing parent(u). Now we query degree sequence structure, as it supports also reverse queries (see Lemma 10) obtaining index of port node. Finally, we use precomputed table, i.e. we query the table which for given index of port node and given cluster returns this port node.

nextsibling(u): We call nextsibling(u') on structure for $\mathcal{T}'$ (T1) and take root of the first tree in the cluster and verify that it has the same parent as u.

lca(u, v): let v' be the cluster of v. We use the structure for $\mathcal{T}'$ (T1): the answer is in the cluster which is represented by node $l = \text{lca}(u', v')$ of $\mathcal{T}'$ but we still need to determine the actual node inside the cluster. To this end we find nodes u'', v'' of $\mathcal{T}'$ such that: both are

children of l, they are ancestors of u' and v' respectively, and u'' and v'' connect to some (port) nodes x, y such that x, y are in the cluster represented by node l and $\mathsf{lca}(x, y) = \mathsf{lca}(u, v)$. They can be computed as follows: u''= $\mathsf{level\_ancestor}(u', \mathsf{depth}(\mathsf{lca}(u', v'))\text{-}\mathsf{depth}(u')\text{-}1)$, the case for v'' is analogous. Having u'' and v'' we can, as in the case for $\mathsf{parent}(v)$, reverse query the structure for degree sequence (T3) obtaining indices of x and y. Finally we use precomputed table (as we want to find lowest common a for port nodes with indices x and y in given cluster).

Note that most tree structures allow to find u'', v'' without calling depth and $\mathsf{level\_ancestor}$, as they are rank/select structures on balanced parenthesis, and it is easy to express this operation using such structures [40].

For general σ we need to only slightly modify our solution.

To encode labels the string P we use results from [16], which achieves $|P|H_0(P) + |P| \log\log|P|$ bits. As $|P| = \mathcal{O}(|\mathcal{T}|/\log_\sigma|\mathcal{T}|)$ this gives required bound.

Additionally, if $\sigma = \Omega(|\mathcal{T}|)$ we do not have to use precomputed tables, as every cluster has constant number of nodes in it, thus we can perform operations inside the clusters in constant time.

In other case we use tables of size $\mathcal{O}(|\mathcal{T}|)$, this is still within bounds, as this is dominated by $\mathcal{O}(|\mathcal{T}| \log\log|\mathcal{T}|/\log_\sigma|\mathcal{T}|)$. ◀

Note, that even Theorem 12 for the case $\sigma = |\mathcal{T}|^{1-\alpha}$, the guarantee on the redundancy is $\mathcal{O}(n)$, which is of the same magnitude as the size of the encoding of the tree using parentheses, so for large alphabets this dominates tree entropy.

Also, in Theorem 12 for the case for arbitrary σ we get a slightly worse redundancy, i.e. we have $\mathcal{O}(|\mathcal{T}| \log\log|\mathcal{T}|/\log_\sigma|\mathcal{T}|)$ factor instead of $\mathcal{O}(|\mathcal{T}| \log\log_\sigma|\mathcal{T}|/\log_\sigma|\mathcal{T}|)$. Still, even in the case of $\sigma = \omega(n^{1-\alpha})$ we can get better bounds (more precisely: $\mathcal{O}(|\mathcal{T}| \log\log_\sigma|\mathcal{T}|/\log_\sigma|\mathcal{T}|)$, or $\mathcal{O}(n)$ for large alphabets) by encoding string of labels P using structure like [4]; but at the cost that operations are slower than $\mathcal{O}(1)$.

We now prove the rest of Theorem 12, that is we can add even more operations, for more complex operations we will need more involved data structures.

Succinct partial sums. To realize more complex operations we make use of structure for succinct partial sums. This problem was widely researched, also in dynamic setting [7]. For our applications, however, it is enough to use a basic, static structure by Raman et al. (RRR) [42].

▶ **Lemma 16.** *For a table $|T| = n_t$ of nonnegative integers such that $\sum_i T[i] \leq n$ and $\frac{n}{n_t} \leq \mathcal{O}(\log^c n)$, for some constant c, we can construct a structure which answers the following queries in constant time:* $\mathsf{sum}(i, j)$*: $\sum_{y=i}^{j} T[y]$;* $\mathsf{find}(x)$*: find first i such that $\sum_{y=1}^{y=i} T[y] \geq x$; and consumes $\mathcal{O}(n_t \log \frac{n}{n_t})$ bits.*

Proof of Lemma 16. We exploit the fact that all the numbers sum up to n. This allows us to store T unary, i.e. as string $0^{T[i]}1 \ldots 0^{T[n_t]}1$. Now using rank/select we can realize desired operations, see [42] for details. For a string with n zeros and n_t ones this structure takes $\log\binom{n+n_t}{n_t} + o(n_t)$ bits. This can be estimated as: $\log\binom{n+n_t}{n_t} + o(n_t) \leq n_t \log(\mathrm{e}(n+n_t)/n_t) + o(n) = \mathcal{O}(n_t \log n/n_t)$, as claimed. ◀

childrank(v), child(v, i). Observe first that if v and its parent are in the same cluster then $\mathsf{childrank}(v)$ can be answered in constant time, as we preprocess all clusters. The same applies to v and its children in case of $\mathsf{child}(v, i)$. Thus in the following we consider only the case when v is a root of a tree in a cluster (for $\mathsf{childrank}(v)$) or it is a port (for $\mathsf{child}(v, i)$).

The problem with those operations is that one port node p can connect to multiple clusters, and each cluster can have multiple trees. We solve it by storing for each port node p a sequence $T_p = t_{p,1}, \ldots, t_{p,j}$, where $t_{p,j}$ is the number of children of p in the j-th (in left to right order) cluster connecting to p.

Observe that all sequences T_p contain in total $|\mathcal{T}'|-1$ numbers, as each number corresponds to one cluster. To make a structure we first concatenate all sequences according to preorder of nodes in $\mathcal{T}'$, and if multiple nodes are in some cluster, we break the ties by left-to-right order on port nodes. Call the concatenated sequence T. Using structure from Lemma 10, for port node p we can find indices i_1, i_2 which mark where the subsequence corresponding to T_p starts and ends in T, i.e. $T[i_1 \,..\, i_2 - 1] = T_p$.

We build the structure from Lemma 16 for T, this takes $\mathcal{O}(|\mathcal{T}| \log\log_\sigma |\mathcal{T}| / \log_\sigma |\mathcal{T}|)$ bits, as $|T| = \mathcal{O}\left(\frac{|\mathcal{T}|}{\log_\sigma |\mathcal{T}|}\right)$ and all elements in T sum up to at most $|\mathcal{T}|$. We realize $\mathsf{childrank}(v)$ as follows: let $v' \in \mathcal{T}'$ be a node representing cluster containing v (by the assumption: as a root). First we find indices i_1, i_2 corresponding to subsequence T_p, where p is port node which connects to v'. Let $j = \mathsf{childrank}(v')$ in $\mathcal{T}'$. Now it is enough to get $\mathsf{sum}(i_1,\ j-1)$, as this corresponds to number of children in first $j - i_1$ clusters connected to p, and add to the result the rank of v in its cluster, the last part is done using look-up tables.

The $\mathsf{child}(v,\ i)$ is analogous: we find indices i_1, i_2 corresponding to T_p. Then we call $\mathsf{find}(i$ + $\mathsf{sum}(1,\ i_1\text{-}1))$ to get cluster containing $\mathsf{child}(v,\ i)$, as we are interested in first index j such that $T[i_1] + \ldots + T[j] > i$. This way we reduced the problem to find i'-th node in given cluster, this can be done using precomputed tables.

depth(v). The downside of clustering procedure is that we lose information on depth of vertices. To fix this, we assign to each edge a non-negative natural *weight* in the following way: Let $v \in \mathcal{T}'$ be any vertex and p be a port node in cluster represented by $\mathsf{parent}(v)$. For an edge $(v,\ \mathsf{parent}(v))$ we assign depth of p in cluster represented by $\mathsf{parent}(v)$. For example in Figure 2 for edge (D, B) we assign 1, and for edge (I, B) we assign 2. In this way the depth of the cluster C (alternatively: depth of roots of trees in C) in $\mathcal{T}$ is the sum of weights of edges from root to node representing C.

A data structure for calculating depths is built using a structure for partial sums: Consider balanced parentheses representation of $\mathcal{T}'$. Then we assign each opening parenthesis corresponding to node v weight $w(v, parent(v))$ and each closing parenthesis weight $-w(v, parent(v))$. This creates the sequence of numbers, for example, for tree from Figure 2 we have:

(	(	(	)	(	)	(	)	(	)	(	)	(	)	(	)	)	(	(	)	(	)	)	)
0	1	1	-1	1	-1	1	-1	2	-2	2	-2	2	-2	2	-2	-1	1	1	-1	1	-1	-1	0

Then we can calculate depth of a cluster by calculating the prefix sum. Observe that our partial sums structure does not work on negative numbers, but we can solve that by creating two structures, one for positive and one for negative number and subtract the result. Finally we use look-up table to find the depth in the cluster

The total memory consumption is bounded by $\mathcal{O}(|\mathcal{T}| \log\log_\sigma |\mathcal{T}| / \log_\sigma |\mathcal{T}|)$, as all weights sum to at most $|\mathcal{T}|$.

preorder_rank(v), preorder_select(i). We again use the structure for succinct partial sums. We create two sequences: First, for each port node consider its preorder rank in $\mathcal{T}$, and arrange these ranks in the sequence according to the preorder ordering: $\mathcal{S}_p = \mathsf{preorder\_rank}(p_1)$, $\mathsf{preorder\_rank}(p_2)$, $\ldots$, $\mathsf{preorder\_rank}(p_{\#\text{ports}})$. Now consider the sequence of the increments, i.e.: $\mathsf{preorder\_rank}(p_1)$, $\mathsf{preorder\_rank}(p_2) - \mathsf{preorder\_rank}(p_1)$, $\ldots$, $\mathsf{preorder\_rank}(p_{\#\text{ports}}) - \mathsf{preorder\_rank}(p_{\#\text{ports-1}})$. With the structure for succinct partial sums created for this sequence we are able to get the preorder_rank in $\mathcal{T}$ for any port node assuming that we know its position in the sequence $\mathcal{S}_p$. We can get this position in the same way as we did for the childrank operation: we store additional partial sum structure where elements are number of port nodes in given cluster (see the childrank operation description for details). Next, we create the sequence of cluster sizes $\mathcal{S}_c = |C_1|, |C_2|, \ldots |C_{|\mathcal{T}'|}|$, where the order of clusters is determined by the preorder ordering of nodes of $\mathcal{T}'$ (recall that each node of $\mathcal{T}'$ corresponds to some cluster).

Now, for a given v to get the $\mathsf{preorder\_rank}(v)$ we sum up three values: preorder rank in $\mathcal{T}'$ of port node which is a connected to cluster C_v containing v, preorder rank of the node v in cluster C_v and the sum of sizes of clusters C_r forming subtrees which are connected to the port nodes p in C_v such that p is before v in preorder ordering of nodes in C_v. To get the first summand, we use previously described structure, to get the second summand we use precomputed tables. Now observe the sum of sizes of clusters C_r form consecutive interval in $\mathcal{S}_c$. Thus it is enough to get the preorder rank in $\mathcal{T}'$ of first and last such cluster. This is straightforward when the structure for $\mathcal{T}'$ supports preorder_rank and rightmost_leaf operation (e.g. we can use the structure from [40]): let p and p' be first and last port nodes which are before v in C_v, we find the preorder rank of the leftmost child of p and preorder rank of rightmost cluster in subtree connected to p. As the elements in sequences sum up to at most $|\mathcal{T}|$, the space is bounded by $\mathcal{O}(|\mathcal{T}| \log\log_\sigma |\mathcal{T}| / \log_\sigma |\mathcal{T}|)$.

The $\mathsf{preorder\_select}(i)$ operation is more involved. For each cluster C we define $p_r(C)$ — the position in preorder ordering of leftmost root of C (note that C can be a forest) according to preorder ordering of vertices of $\mathcal{T}$. Consider the sequence of increments $\mathcal{D} = p_r(C_1), p_r(C_2) - p_r(C_1), p_r(C_3) - p_r(C_2), \ldots, p_r(C_{|\mathcal{T}'|-1}) - p_r(C_{|\mathcal{T}'|})$, where the clusters are ordered to preorder ordering according to $\mathcal{T}'$.

Now, by using succinct partial sums structure on the sequence D, for a given i we can find two (consecutive in preorder ordering on $\mathcal{T}'$) clusters C_j, C_{j+1} such that $p_r(C_j) \leq i < p_r(C_{j+1})$. Observe that there are two possibilities: either C_j is parent of C_{j+1} in $\mathcal{T}'$ and C_{j+1} is the first child of C_j, or C_j is the rightmost node of subtree rooted in parent C_p of C_{j+1} in $\mathcal{T}'$. If C_j is a parent of C_{j+1}, it is sufficient to find a node with preorder rank of $i - p_r(C_{j+1})$, as C_{j+1} is the first child of C_j, for this we can use precomputed tables.

If C_j is a rightmost node of subtree rooted in C_p (and hence a leaf) we have three possibilities: either the requested node is in C_j or in C_p or in parent of C_p. If $|C_j| + p_r(C_j) \leq i$ the node is in C_j and so we use the precomputed tables as in former case.

Otherwise we find two (not necessarily different) port nodes in C_p, n_j and n_{j+1}, which are the nodes on the path from C_j to C_p and C_{j+1} to C_p, respectively. Let n_{r_j} and $n_{r_{j+1}}$ be the roots of subtrees of C_p which contain n_j and n_{j+1}, respectively. Now the searched node is either a parent of n_{r_j} and $n_{r_{j+1}}$ (observe that they must share a parent), or in one of the subtrees in C_p rooted at either n_{r_j} or $n_{r_{j+1}}$. We check the former case by simply calling preorder_rank for the parent of n_{r_j}. For the latter case it is enough to get the node which is $i - (p_r(C_j) + |C_j| - 1)$ positions after n_j in C_p according to preorder ordering of nodes in C_p. For this we can use precomputed tables (i.e. we store tables which allow to call preorder_rank and preorder_select for vertices of a given cluster).

leaf_rank(v), leaf_select(i). Consider the sequence of leaves of $\mathcal{T}$, $L = v_1, v_2, \ldots, v_{|L|}$ where leaves are ordered from left to right. Now consider the grouping of leaves such that two leaves are in the same group if and only if they are in the same cluster in $\mathcal{T}'$, call the obtained sequence of groups $L_G = G_1, G_2, \ldots G_{|L|}$. We build the succinct partial sum structure for sequence of group sizes, $LS_G = |G_1|, |G_2|, \ldots |G_{|L|}|$.

Assuming that our structure for $\mathcal{T}'$ supports leaf_rank(u) and leaf_select(j) (this can be achieved by using the structure from [40] for $\mathcal{T}'$) we can realize the operations as follow: For leaf_rank(v) we first call the leaf_rank(u), where u is vertex in $\mathcal{T}'$ representing cluster C_v, this way we know how many clusters containing leaves are to the left of C_v in $\mathcal{T}'$. We use the structure for partial sums to get the number of leaves in clusters to the left of v, for the vertices in C_v to the left of v we use precomputed tables. Analogously, we realize leaf_select(i) by first using our structure for partial sums, this allows to identify the index i' in LS_G, then we call leaf_select(i') on structure for $\mathcal{T}'$, we also use precomputed tables to identify the leaf inside the cluster.

level_ancestor(v, i) We assign weights to edges as in the case of depth operation. This reduces the level ancestor in $\mathcal{T}$ to *weighted level ancestor* in $\mathcal{T}'$; in this problem we ask for such ancestor w of v that sum of weights on the path from w to v is at least i and w is closest node to v in the terms of number of nodes on the path (note that there may not exist a node for which the sum is equal exactly to i). The redundancy obtained for level_ancestor operation is slightly worse than for previous operations, but not worse than most of the other structures [33, 22] supporting this operation. Observe that each edge has weight of order $\mathcal{O}(\log_\sigma |\mathcal{T}|)$. From the following theorem we get that additional $\mathcal{O}(|\mathcal{T}|(\log\log|\mathcal{T}|)^2/\log_\sigma|\mathcal{T}|)$ bits is sufficient.

▶ **Lemma 17.** *Let $\mathcal{T}', |\mathcal{T}'| = t$ be a tree where each edge is assigned a weight of at most $\mathcal{O}(\log n)$, for some n. We can build structure which consumes $\mathcal{O}(t(\log\log n)^2)$ bits of memory and allows to answer weighted level ancestor queries in $\mathcal{O}(1)$ time.*

With the structure from Lemma 17 the query level_ancestor(v, i) is easy: first we check if the answer is in the same cluster using preprocessed array. If not we find cluster containing the answer, we do it by asking for level_ancestor(C, i-depth(C, v)), where depth(C, v) is depth of v in cluster C containing v. There is similar problem as in the case of lca query, that is we also need to find the port node on path from given vertex v to its i-th ancestor. This may be solved in the same manner as in the case for lca.

Now we give the construction for weighted level ancestor structure. Note that there are multiple ways of doing this [22, 40, 33]. We use the tree partitioning approach, yet the one that operates on sequence of numbers, as in case for depth, should also be applicable, [33] shows similar method (and uses same additional space), yet for simplicity we choose to stick with solution which partition the tree into subtrees as we already defined most of the required machinery. Note that tree partitioning method [22], which we refer to, partitioned the tree a few times, we do it once and use stronger result [40] for the simplicity of proof. Also, the partitioning from [22] may be used instead of our method.

Proof of Lemma 17. We use the idea from [22]. We first cluster the tree according to Lemma 2 with $m = \Theta(\log^3 n)$. We obtain a smaller tree $\mathcal{T}''$ of size $|\mathcal{T}''| = \mathcal{O}\left(\frac{t}{\log^3 n}\right)$. We store labels of $\mathcal{T}''$ and the descriptions of clusters naively, without the entropy coder. We also store additional structure for navigation of $\mathcal{T}''$, including degree sequences, observe that it takes at most $\mathcal{O}(t)$ bits.

Similarly, as in the case for depth, we assign weights to edges of $\mathcal{T}''$ but we have to remember that the input tree is weighted as well. Let $v \in \mathcal{T}''$ be any vertex and p be a port node in a cluster represented by $\mathsf{parent}(v)$, call the cluster C_p. For an edge $(v, \mathsf{parent}(v))$ we assign the sum of the weights on the path from p to the root of the tree containing p in C_p (i.e. the weighted depth of p in C_p).

Observe that the weights in the $\mathcal{T}''$ are of order $\mathcal{O}(\log^4 n)$. For such weighted tree we can build structure which supports $\mathsf{level\_ancestor}$ queries in $\mathcal{O}(1)$ time and use $\mathcal{O}(|\mathcal{T}''| \log^2 n) = o(n)$ bits, using result from [5].

Now for each smaller tree we can build structure from [40]. For a tree of size t' and with weights limited by $\mathcal{O}(\log^4 n)$ this structure takes $\mathcal{O}(t' \log t' \log(t' \log^4 n)) = \mathcal{O}(t'(\log\log n)^2)$ bits. Summing over all trees, we get $\mathcal{O}(t(\log\log n)^2)$, as claimed. For each such tree we additionally store the information, which nodes are port nodes in a way that allow to retrieve i-th port node. This can be achieved by storing the bitmap for each small tree (in which each j-th element indicates if j leaf is port node or not) and applying rank/select structure (this consumes $\mathcal{O}(t)$ bits for all trees). Observe that we can even explicitly list all of the port nodes: we do not have to use space efficient solution, as there are at most $|\mathcal{T}''|$ such nodes, so even consuming $\mathcal{O}(\log^2 n)$ bits for port node is sufficient.

We perform $\mathsf{level\_ancestor}$ operation in the same manner as previously described when applying Lemma 17, i.e. we first check if the answer is in the same tree, if yes we can output the answer as we can perform operations on small trees in $\mathcal{O}(1)$ time [40], if not we use combination of depth and $\mathsf{level\_ancestor}$ queries on structure for $\mathcal{T}''$ in the same way as we did for lca query (see proof of Theorem 12). It is possible as structure [5] supports all of the required operations (or can be easily adapted to support by adding additional tree structure, as the structure for $\mathcal{T}''$ can consume up to $\mathcal{O}(\log^2 n)$ bits per node, in particular this means that we can even preprocess all answer for depth queries for $\mathcal{T}''$).

The only nontrivial thing left is that we would like to not only find a node in our structure but also find corresponding node in structure for $\mathcal{T}'$. To this end we show that we can return preorder position of given node in $\mathcal{T}'$, this is sufficient as structure for $\mathcal{T}'$ has $\mathsf{preorder\text{-}select}$ operation.

To this end we explicitly store preorder and subtree size for each port node, observe that this consumes at most $\mathcal{O}(\mathcal{T}'' \log t) = \mathcal{O}(t)$ bits. Now given a node v in some cluster to find preorder rank of this node in $\mathcal{T}'$ we sum the following values: preorder rank of v in cluster C containing v, the rank of port node p_v which is connected to cluster C, and the sizes of subtrees T_i which are connected to port nodes c_i of C, such that c_i precedes v in preorder ordering in C. To find the sum of sizes of c_i we first find how many c_i's precede v in preorder ordering in C, to this end we use rank/select structure for binary vector B_C where $B_C[i] = 1$ if and only if i-th node according to preorder ordering in C is a port node. Then we use the structure for partial sums (again, we do not have to use succinct structure as there are at most $\mathcal{O}(|\mathcal{T}''|)$ elements in total). ◀

E Additional material for Section 6

The following Lemmas says that if we partition the clusters into groups according to their k-letter contexts and encode each group separately with zeroth-order entropy we can get better estimation than encoding them together.

▶ **Lemma 18.** *Let $\mathcal{T}'$ be a labeled cluster tree from Lemma 4 for parameter m, obtained from $\mathcal{T}$. For each k-letter context K_i let P_{K_i} be a concatenation of labels of $\mathcal{T}'$ which are preceded by this context (i.e. each root v in each cluster is preceded by the context K_i in $\mathcal{T}$). Then* all *of the following inequalities hold:*

1. $\sum_i |P_{K_i}| H_0(P_{K_i}) \leq |\mathcal{T}| H(\mathcal{T}) + |\mathcal{T}| H_k(L) + \mathcal{O}\left(\frac{|\mathcal{T}| \log m}{m}\right)$;
2. $\sum_i |P_{K_i}| H_0(P_{K_i}) \leq |\mathcal{T}| H_k(\mathcal{T}|L) + |\mathcal{T}| H_k(L) + \mathcal{O}\left(\frac{|\mathcal{T}| \log m}{m}\right)$;
3. $\sum_i |P_{K_i}| H_0(P_{K_i}) \leq |\mathcal{T}| H(\mathcal{T}) + |\mathcal{T}| H_k(L|\mathcal{T}) + \mathcal{O}\left(\frac{|\mathcal{T}| \log \sigma}{m} + \frac{|\mathcal{T}| \log m}{m}\right)$.

Proof of Lemma 18. The proof is very similar to the proof of Theorem 5. For each P_{K_i} we apply the Lemma 6: we use almost the same values of q function for each cluster but we do not need to multiply it by $q(K_C)$, i.e. we define $q(C) = q(N_C) \cdot q(V_C)$. For detailed definition of q see proof of Theorem 5. It is easy to check that without this factor we arrive at the claim. ◀

▶ **Lemma 19.** *Let $\mathcal{T}'$ be a labeled cluster tree from Lemma 4 for parameter m, obtained from $\mathcal{T}$. Let P be a string obtained by concatenation of labels of $\mathcal{T}'$. Then we can encode P in a way, that, given context $K_{P[i]}$, we can retrieve $P[i]$ in constant time. The encoding is bounded by* all *of the following values:*

1. $|\mathcal{T}| H(\mathcal{T}) + |\mathcal{T}| H_k(L) + \mathcal{O}\left(\frac{|\mathcal{T}|(\log m + \log\log|\mathcal{T}|)}{m} + \sigma^{k+m} \cdot 2^m \cdot \log|\mathcal{T}|\right)$;
2. $|\mathcal{T}| H_k(\mathcal{T}|L) + |\mathcal{T}| H_k(L) + \mathcal{O}\left(\frac{|\mathcal{T}|(\log m + \log\log|\mathcal{T}|)}{m} + \sigma^{k+m} \cdot 2^m \cdot \log|\mathcal{T}|\right)$;
3. $|\mathcal{T}| H(\mathcal{T}) + |\mathcal{T}| H_k(L|\mathcal{T}) + \mathcal{O}\left(\frac{|\mathcal{T}| \log \sigma}{m} + \frac{|\mathcal{T}|(\log m + \log\log|\mathcal{T}|)}{m} + \sigma^{k+m} \cdot 2^m \cdot \log|\mathcal{T}|\right)$.

Proof of Lemma 19. We use Lemma 18. For each P_{K_i} we generate codes using Huffman encoding, this allows us to encode each P_{K_i} using $|P_{K_i}| H_0(P_{K_i}) + |P_{K_i}|$ bits, as we lose at most 1 bit per code (see [16] for example), plus additional $\mathcal{O}(2^m \sigma^m \log|\mathcal{T}'|) \leq \mathcal{O}(2^m \sigma^m \log|\mathcal{T}|)$ bits for Huffman dictionary. Summing this over all contexts K_i yields the bound, by Lemma 18.

Denote c_v as Huffman code for vertex v of $\mathcal{T}'$. Let T_C be the concatenation of all codes c_v according to order in string P. As each code is of length at most $\mathcal{O}(\log|\mathcal{T}'|)$, given its start and end in T_C we can decode it in constant time. We store the bitmap of length $|T_C|$ where $T_C[j] = 1$ if and only if at j-th is the beginning of some code. This bitmap has length at most $|\mathcal{T}'| \log|\mathcal{T}|$ and has $|\mathcal{T}'|$ ones. For such a bitmap we build rank/select structure, using result by Raman et al. [42] this takes $\mathcal{O}(|\mathcal{T}'| \log\log|\mathcal{T}|)$ bits. Using rank/select we can retrieve the starting position of i-th code by simply calling select(i) (same idea was used in [23]). ◀

Now we would like to simply apply Lemma 19, which says that we can encode labels of $\mathcal{T}'$ more efficiently, yet there is one major difficulty: the Lemma states that to decode $P[i]$ we need to know the context. The idea is that we choose $|\mathcal{T}'|/d$ nodes for which we store the context, for rest we can retrieve the contexts in time $\mathcal{O}(d)$ by traversing $\mathcal{T}'$ and decoding them on the way.

Proof of Theorem 13. We use almost the same structures as in the simpler case, i.e. the structure from Theorem 12, the only difference is that instead encoding preorder sequence P with structure using space proportional to zeroth order entropy we apply Lemma 19. We choose $m = \beta \log_\sigma |\mathcal{T}|$ so that $2\beta + \alpha < 1$ and $\beta < \frac{1}{8}$, so that the precomputed tables use $o(|\mathcal{T}|)$ space. As each operation in Theorem 12 accessed elements of P constant number

of times it is sufficient to show how to access it in time $\mathcal{O}(\log|\mathcal{T}|/\log\log|\mathcal{T}|)$ time. By Lemma 19 this leaves us with problem of finding context for each node in aforementioned complexity.

Let $d = \lceil \log|\mathcal{T}|/\log\log|\mathcal{T}|\rceil$. We choose at most $\mathcal{O}(|\mathcal{T}'|\log\log|\mathcal{T}|/\log|\mathcal{T}|)$=$\mathcal{O}(|\mathcal{T}'|/d)$ nodes, for which we store the context explicitly, in the following way: We store the context for the root using $\lceil k\log\sigma\rceil$ bits. We partition the nodes into d classes C_i depending on their depth modulo d, i.e. in the class C_i there are nodes at depth $dj+i, j\geq 0$. Then there is a class having at most $\mathcal{T}'/d$ such nodes, we choose all nodes in this class and store their contexts.

Now we show, assuming we know the contexts for chosen nodes, that given a node we can retrieve its context in $\mathcal{O}(d)$ time. If we want to compute the context for some node v we first check whether it is stored explicitly. If not, we look at nodes on path from v to the root, until we find a node u which has its context stored. Call the visited nodes $v, v_1, v_2, \ldots, v_i, u$. Observe that we visited at most $\mathcal{O}(d)$ nodes that way. As we know the context for u, now we can decode the node u, and determine the context for $v_i, v_{i-1}, \ldots, v_1, v$. To read the labels in u which precede v_i in constant time we first find port node which connects to u (as in the proof of Theorem 12) and use the precomputed tables.

The only nontrivial thing left to explain is how to store the contexts for chosen $\mathcal{O}(|\mathcal{T}'|/d)$ nodes and check which nodes have their contexts stored. We concatenate all contexts for chosen $\mathcal{O}(|\mathcal{T}'|/d)$ nodes according to their order in preorder ordering. On top of that we store binary vector B which satisfies $B[i]=1$ if and only if i-th node of $\mathcal{T}'$ in preorder ordering has its context stored. We build rank/select structure for B, as we have preorder-rank and preorder-select operation for $\mathcal{T}'$ in constant time, for a given node we can check in constant time if the node have its index stored or not. As each context has the same bit-length to decode context for node which is j-th in preorder ordering we look at position $(j-1)\lceil k\log\sigma\rceil$.

The total space for storing the context is $\mathcal{O}(|\mathcal{T}'|k\log\sigma/d) = \mathcal{O}(|\mathcal{T}'|\log\log|\mathcal{T}|)$, summing that up with space bound from Lemma 19 yields the claim. ◀

F Additional material for Section 7

Proof of Theorem 14. The first part of the theorem is easy: if σ is constant we can construct a separate structure for each letter. For each letter a we build a separate degree sequence, level_ancestor structure and depth structure; observe that all of those structures support the weighted case when we assign each vertex weight of 0 or 1, so it is sufficient to assign nodes labeled with a value 1 and for the rest value 0. Similar idea was mentioned in [44, 28, 22].

For the next two parts we show how to adapt rank/select structures over large alphabets to support childrank/childselect queries.

For the second part, when $\sigma=\mathcal{O}(\log^{1+o(1)}|\mathcal{T}|)$, we use result by Belazzougui et al. [4]. They show (at discussion at above Theorem 5.7 in [4]) how for the sequence S divided into $\mathcal{O}(|S|/m)$ blocks of length at most m, for some m, construct rank/select structure for large alphabets, assuming that we can answer queries in time $\mathcal{O}(t)$ in blocks, such that it takes $\mathcal{O}(t)$ time for query and consumes additional $|S|\log\frac{\sigma}{m}+\mathcal{O}\left(|S|+\frac{(\sigma|S|/s)\log\log(\sigma|S|/s)}{\log\sigma|S|/s}\right)$ bits (in [4] the assumption is that we can answer queries in blocks in $\mathcal{O}(1)$ time but as the operations on additional structure cost constant time, our claim also holds). The solution uses only succinct bitmaps by Raman et al. [43] and precomputed tables for additional data. Now we can define sequence S as concatenation of labels of roots of cluster, where clusters are ordered by preorder ordering, this gives our blocked sequence (where blocks correspond to clusters). Observe that labeled childrank/childselect operations can easily be reduced to

labeled rank/select in string S, all we need to do is to know where the sequence for children of a given vertex v begins in S. Fortunately, this can be done in same manner as in Lemma 10, that is, we use structure for degree sequence. As in our case $m = \log_\sigma |\mathcal{T}|$, we can store precomputed tables to answer rank/select queries for each cluster. The additional space is $o(|S| \log \sigma)$ and clearly $|S| \leq |\mathcal{T}|$.

For the last part we use Lemma 3 from [3]. The lemma states that for a string $|S|$ if, for a given i, we can access i-th element in time $\mathcal{O}(t)$ then we can, using additional $o(|S| \log \sigma)$ bits, support labeled rank/select operations in time $\mathcal{O}(t \log \log^{1+\epsilon} \sigma)$. We use the same reduction as in the case for $\sigma = \mathcal{O}(\log^{1+o(1)} |S|)$, i.e. we set S as concatenation of labels of roots of clusters, where clusters are ordered by preorder ordering. As in previous case, we use node degree sequence and tree structure to retrieve i-th character in $|S|$ (i.e. we first find appropriate cluster and use precomputed tables). ◀

Quantum Distributed Algorithm for Triangle Finding in the CONGEST Model

Taisuke Izumi
Graduate School of Engineering, Nagoya Institute of Technology, Japan

François Le Gall
Graduate School of Mathematics, Nagoya University, Japan

Frédéric Magniez
Université de Paris, IRIF, CNRS, France

Abstract

This paper considers the triangle finding problem in the CONGEST model of distributed computing. Recent works by Izumi and Le Gall (PODC'17), Chang, Pettie and Zhang (SODA'19) and Chang and Saranurak (PODC'19) have successively reduced the classical round complexity of triangle finding (as well as triangle listing) from the trivial upper bound $O(n)$ to $\tilde{O}(n^{1/3})$, where n denotes the number of vertices in the graph. In this paper we present a quantum distributed algorithm that solves the triangle finding problem in $\tilde{O}(n^{1/4})$ rounds in the CONGEST model. This gives another example of quantum algorithm beating the best known classical algorithms in distributed computing. Our result also exhibits an interesting phenomenon: while in the classical setting the best known upper bounds for the triangle finding and listing problems are identical, in the quantum setting the round complexities of these two problems are now $\tilde{O}(n^{1/4})$ and $\tilde{\Theta}(n^{1/3})$, respectively. Our result thus shows that triangle finding is easier than triangle listing in the quantum CONGEST model.

2012 ACM Subject Classification Theory of computation → Quantum computation theory

Keywords and phrases Quantum computing, distributed computing, CONGEST model

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.23

Acknowledgements The authors are grateful to anonymous reviewers for helpful comments. TI was partially supported by JST SICORP grant No. JPMJSC1606 and JSPS KAKENHI grant No. JP19K11824. FLG was supported by JSPS KAKENHI grants Nos. JP15H01677, JP16H01705, JP16H05853, JP19H04066 and by the MEXT Quantum Leap Flagship Program (MEXT Q-LEAP) grant No. JPMXS0118067394. FM was partially supported by the ERA-NET Cofund in Quantum Technologies project QuantAlgo and the French ANR Blanc project QuData.

1 Introduction

Background. The problem of detecting triangles in graphs has recently become the target of intensive research by the distributed computing community [1, 5, 6, 7, 8, 9, 18, 25]. This problem comes in two main variants: the *triangle finding problem* and the *triangle listing problem*. Given as input a graph $G = (V, E)$, the triangle finding problem asks to decide[1] whether the graph contains a triangle (i.e., three vertices $u, v, w \in V$ such that $\{u, v\}, \{u, w\}, \{v, w\} \in E$), while the triangle listing problem asks to output all the triangles of G. A solution to the latter version, obviously, gives a solution to the former version. Besides its theoretical interest, another motivation for considering these problems is that

[1] Another version of the triangle finding problem asks to output one triangle of G (or report that the graph has no triangle). It is easy to see that the two versions are essentially equivalent: a triangle can be found by applying $O(\log |V|)$ times an algorithm solving the decision version.

for several graph problems faster distributed algorithms are known over triangle-free graphs (e.g., [16, 26]). The ability to efficiently check whether the network is triangle-free (or detect which part of the network is triangle-free) is essential when considering such algorithms.

One of the main models to study graph-theoretic problems in distributed computing is the CONGEST model. In this model the graph $G = (V, E)$ represents the topology of the network, the computation proceeds with round-based synchrony and each vertex can send one $O(\log n)$-bit message to each adjacent vertex per round, where n denotes the number of vertices. Initially, each vertex knows only the local topology of the network, i.e., the set of edges incident to itself. The triangle finding and listing problems ask, respectively, to decide if G contains a triangle and to list all triangles of G. The trivial strategy is for each vertex to send the list of all its neighbors to each neighbor (all the triangles can then be listed locally, i.e., without further communication). Since each list can contain up to n vertices, this requires $O(n)$ rounds in the CONGEST model.

In 2017, Izumi and Le Gall [18] gave the first nontrivial distributed algorithms for triangle detection in the CONGEST model: they constructed a $\tilde{O}(n^{2/3})$-round randomized algorithm[2] for triangle finding and a $\tilde{O}(n^{3/4})$-round randomized algorithm for triangle listing. This was soon improved by Chang, Pettie and Zhang [6], who obtained a $\tilde{O}(\sqrt{n})$-round randomized algorithm for both the triangle finding and listing problems. The key idea leading to this improvement was to decompose the graph into components with low mixing time, and then apply recent routing techniques [13, 14] that make possible to achieve efficient routing in graphs with low mixing time. The complexity of the resulting distributed algorithm was dominated by the cost required to compute the graph decomposition. Very recently, Chang and Saranurak [7] developed a much more efficient method to decompose the graph into components with low mixing time, which immediately leads to $\tilde{O}(n^{1/3})$-round randomized algorithms for the triangle finding and listing problems. Since a matching lower bound $\tilde{\Omega}(n^{1/3})$ is known for triangle listing [25], the randomized round complexity of the triangle listing problem is thus now settled, up to possible polylogarithmic factors.[3] For the triangle finding problem, on the other hand, essentially no nontrivial lower bound is known. Two exceptions are, first, the very weak lower bound obtained by Abboud, Censor-Hillel, Khoury and Lenzen [1] in the CONGEST model and, second, a lower bound obtained by Drucker, Kuhn and Oshman [9] for the much weaker CONGEST-BROADCAST model (where at each round the vertices can only broadcast a single common message to all other vertices) under a conjecture in computational complexity theory. This leads to the following intriguing question: is triangle finding easier than triangle listing?

Another related, but much stronger, model is the CONGEST-CLIQUE model. In this model at each round messages can even be sent between non-adjacent vertices, which makes bandwidth management significantly easier. In the CONGEST-CLIQUE model, Dolev, Lenzen and Peled [8] first showed that the triangle listing problem (and thus the triangle finding problem as well) can be solved deterministically in $\tilde{O}(n^{1/3})$ rounds for general graphs, which is tight since the lower bound $\tilde{\Omega}(n^{1/3})$ by Pandurangan, Robinson and Scquizzato [25] mentioned above holds in this model as well. Note that this lower bound also means that triangle listing in the CONGEST-CLIQUE model is not easier than triangle listing in the CONGEST model, at least as far as randomized algorithms are concerned. For the triangle

[2] In this paper the notations $\tilde{O}(\cdot)$ and $\tilde{\Omega}(\cdot)$ remove $\mathrm{poly}(\log(n))$ factors.

[3] An interesting open problem, however, is to determine the deterministic round complexity of these problems. To our knowledge, no deterministic algorithm with sublinear round complexity is known in the CONGEST model.

Table 1 Prior results on the round complexity of distributed triangle finding and listing, and our new result. Here n denotes the number of vertices of the graph. Note that any upper bound for the listing problem (in particular, the upper bound from [7]) holds for the finding problem as well. Similarly, note that any lower bound for the quantum CONGEST-CLIQUE model (in particular, the lower bound from [25]) holds for the weaker classical and quantum CONGEST models as well.

Model	Setting	Problem	Complexity	Paper
CONGEST-CLIQUE	Classical	Listing	$\tilde{O}(n^{1/3})$	Dolev et al. [8]
CONGEST-CLIQUE	Classical	Finding	$O(n^{0.1572})$	Censor-Hillel et al. [5]
CONGEST	Classical	Listing	$\tilde{O}(n^{1/3})$	Chang and Saranurak [7]
CONGEST	Quantum	Finding	$\tilde{O}(n^{1/4})$	This paper
CONGEST-BROADCAST	Classical	Finding	$\Omega\left(\frac{n}{e^{\sqrt{\log n}}\log n}\right)$	Drucker et al. [9]
CONGEST-CLIQUE	Quantum	Listing	$\Omega\left(\frac{n^{1/3}}{\log n}\right)$	Pandurangan et al. [25]

finding problem, on the other hand, the better upper bound $O(n^{0.1572})$ has been obtained by Censor-Hillel et al. [5] by implementing fast matrix multiplication algorithms in the distributed setting. The CONGEST-CLIQUE model is thus a setting in which triangle finding is easier than triangle listing.

Table 1 summarizes the best known bounds on the round complexity of triangle finding and listing discussed so far.

Quantum distributed computing. Quantum versions of the main models studied in distributed computing can be easily defined by allowing quantum information, i.e., quantum bits (qubits), to be sent through the edges of the network instead of classical information, i.e., bits. In particular, in the quantum version of the CONGEST model, which we will simply call the "quantum CONGEST model" below, each vertex can send one message of $O(\log n)$ qubits to each adjacent vertex per round. While a seminal work by Elkin et al. [10] showed that for many important graph-theoretical problems the quantum CONGEST model is not more powerful than the classical CONGEST model, Le Gall and Magniez [20] recently showed that one problem can be solved more efficiently in the quantum setting: computing the diameter of the network. More precisely, they constructed a $\tilde{O}(\sqrt{nD})$-round quantum algorithm for the exact computation of the diameter of the network (here D denotes the diameter), while it is known that any classical algorithm in the CONGEST model requires $\tilde{\Omega}(n)$ rounds, even for graphs with constant diameter [11]. In the CONGEST-CLIQUE model as well, a quantum algorithm faster than the best known classical algorithms has been obtained recently for the All-Pair Shortest Path problem [17]. In the LOCAL model, which is another fundamental model in distributed computing, separations between the computational powers of the classical and quantum versions have also been reported [12, 21].

When discussing the classical randomized complexity of triangle listing in the CONGEST and CONGEST-CLIQUE models, we mentioned the $\tilde{\Omega}(n^{1/3})$-round lower bound by Pandurangan, Robinson and Scquizzato [25]. This lower bound is based on an information-theoretic argument that actually holds even in the quantum setting. In view of the recent matching upper bound in the classical setting [7], we can conclude that for triangle listing the quantum CONGEST model is not more powerful than the classical CONGEST model. An intriguing question is whether quantum communication can help solving faster the triangle finding problem in the CONGEST model. In particular, can we break the $\tilde{O}(n^{1/3})$ barrier for triangle finding in the quantum setting?

Our result. In this paper we break this barrier. Our main result is the following theorem.

▶ **Theorem 1.** *In the quantum CONGEST model, the triangle finding problem can be solved with probability at least* $1 - 1/\mathrm{poly}(n)$ *in* $\tilde{O}(n^{1/4})$ *rounds, where* n *denotes the number of vertices in the network.*

In comparison, as already explained, in the classical CONGEST model the best known upper bound on the randomized round complexity of triangle finding is $\tilde{O}(n^{1/3})$. Our result thus gives another example of quantum algorithm beating the best known classical algorithms in distributed computing. It also exhibits an interesting phenomenon: while in the classical setting the best known upper bounds for the triangle finding and listing problems are both $\tilde{O}(n^{1/3})$, in the quantum setting the round complexity of the former problem becomes $\tilde{O}(n^{1/4})$ while the round complexity of the latter problem remains $\tilde{\Theta}(n^{1/3})$. Theorem 1 thus shows that triangle finding is easier than triangle listing in the quantum CONGEST model.

Overview of our approach. Our approach starts similarly to the classical algorithms developed by Chang, Pettie and Zhang [6] and Chang and Saranurak [7]. As in [6], by using an expander decomposition of the network, the triangle finding problem over the whole network is reduced to the task of detecting whether the subnetwork induced by a set of edges $E^{\mathrm{in}} \cup E^{\mathrm{out}}$ contains a triangle. We denote the latter problem FindTriangleInSubnetwork. Here E^{in} and E^{out} are two subsets of edges that satisfy specific conditions. In particular, the subnetwork induced by the edges in E^{in} is guaranteed to have low mixing time (but in general nothing can be said about the mixing time of the subnetwork induced by $E^{\mathrm{in}} \cup E^{\mathrm{out}}$). We use the algorithm from [7] to compute the expander decomposition efficiently, which implies that an efficient algorithm for FindTriangleInSubnetwork gives an efficient algorithm for the triangle finding problem over the whole network. More precisely, a $\tilde{O}(n^{1/4})$-round algorithm for FindTriangleInSubnetwork leads to a $\tilde{O}(n^{1/4})$-round algorithm for the triangle finding problem. The details of this reduction are described in Section 3.

Our main approach to solve the problem FindTriangleInSubnetwork is to apply the framework for quantum distributed search developed in [20]. We briefly sketch the main ideas. Let us write $\mathcal{V} \subseteq V$ the set of vertices of the graph induced by the edges in $E^{\mathrm{in}} \cup E^{\mathrm{out}}$. We will partition $\mathcal{V}$ into $t = \tilde{\Theta}(\sqrt{n})$ subsets $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_t$ each containing $\tilde{O}(\sqrt{n})$ vertices. Let us write $\Lambda = [t] \times [t] \times [t]$. We will try to find a triple $(i, j, k) \in \Lambda$ for which there exist $u \in \mathcal{V}_i$, $v \in \mathcal{V}_j$ and $w \in \mathcal{V}_k$ such that $\{u, v, w\}$ is a triangle. To do this, we partition the set Λ into t subsets $\Lambda_1, \cdots, \Lambda_t$ each containing $t^2 = \tilde{\Theta}(n)$ triples and consider the following search problem: find an index $\ell \in [t]$ such that the set Λ_ℓ contains a triple (i, j, k) for which there exist $u \in \mathcal{V}_i$, $v \in \mathcal{V}_j$ and $w \in \mathcal{V}_k$ such that $\{u, v, w\}$ is a triangle. Quantum distributed search enables us to solve this problem in $\tilde{O}(\sqrt{t}\delta)$ rounds in the quantum CONGEST model if a checking procedure (i.e., a procedure that on input ℓ checks if the set Λ_ℓ contains a triple (i, j, k) for which there exist $u \in \mathcal{V}_i$, $v \in \mathcal{V}_j$ and $w \in \mathcal{V}_k$ such that $\{u, v, w\}$ is a triangle) can be implemented in δ rounds.

The checking procedure distributes the $\tilde{\Theta}(n)$ triples in Λ_ℓ among the vertices of the network proportionally to the degree of the vertices. In particular, vertices with very low degree do not receive any triple. This technique is essentially the same as for the main procedure in the classical algorithm by Chang, Pettie and Zhang [6]. Next, each vertex owning a triple (i, j, k) checks whether there exist $u \in \mathcal{V}_i$, $v \in \mathcal{V}_j$ and $w \in \mathcal{V}_k$ such that $\{u, v, w\}$ is a triangle, which requires gathering information of all the edges with extremities in these sets. Since the subnetwork induced by the edges in E^{in} has low mixing time, we would like to use the same classical routing techniques [13, 14] as used in the main procedure

of the classical algorithms [6, 7]. Special care is nevertheless required to ensure that we can apply those routing techniques. (This was not needed in [6, 7] since these prior works used a different approach: each vertex simply loaded all the necessary edges from Λ in $\tilde{O}(n^{1/3})$ rounds. In comparison, we need to guarantee that the necessary edges from Λ_ℓ, for a fixed ℓ, can be loaded in negligible time.) We solve this difficulty by carefully defining the sets Λ_ℓ so that the the same edge is not requested by two distinct vertices at the same time (this is the contents of Lemma 10 in Section 4.2).

Another technical difficulty is how to handle vertices with very high degree. In the classical setting this was trivial, since it was enough to gather in $\tilde{O}(n^{1/3})$ rounds all the information about the edges of the network at one of these high-degree vertices. Since we want to construct a $\tilde{O}(n^{1/4})$-round quantum algorithm we cannot use this approach. Instead, we develop an approach based on the well-known protocol from the two-party quantum computation complexity of the disjointness function [4]. This is explained in Section 4.1.

Other related works. The triangle finding problem is also a central problem in quantum query complexity. While many quantum query algorithms have been designed in this setting [2, 19, 22, 23], they are based on quantum techniques (e.g., quantum walk search and learning graphs) that do not seem to lead to efficient algorithms in the distributed setting.

2 Preliminaries

Graph theory. All the graphs considered in this paper are undirected and unweighted. For any graph $G = (V, E)$ and any vertex $u \in V$, we denote $\deg(u)$ the degree of u and $\mathcal{N}(u)$ the set of neighbors of u. We write $n = |V|$ and $m = |E|$. For any set $E' \subseteq E$, we denote $\deg_{E'}(u)$ the number of edges in E' incident to u and write $\mathcal{N}_{E'}$ the set of all neighbors $v \in \mathcal{N}(u)$ such that $\{u, v\} \in E'$. We denote $\mathrm{diam}(G)$ the diameter of G and $\mathrm{mix}(G)$ the mixing time of G, i.e., the number of steps of a random walk over G needed to obtain a distribution close to the stationary distribution (we refer to [13] for a precise definition). For any positive integer t, we write $[t] = \{1, 2, \ldots, t\}$.

We will use the following lemma from [6] in our main algorithm.

▶ **Lemma 2** (Lemma 4.2 in [6]). *Consider a graph with m edges and n vertices. Let p be such that $p^2 m \geq 400(\log n)^2$. Suppose that the degree of any vertex of the graph is at most $mp/(20 \log n)$. Generate a subset S by letting each vertex join S independently with probability p. Then with probability at least $1 - 1/\mathrm{poly}(n)$, the number of edges in the subgraph induced by S is at most $6p^2 m$.*

Classical distributed computing. In the classical CONGEST model, the graph $G = (V, E)$ represents the topology of the network. The computation proceeds with round-based synchrony and each vertex can send one $O(\log n)$-bit message to each adjacent vertex per round. All links (corresponding to the edges of G) are reliable and suffer no faults. Each vertex has a distinct identifier from a domain $\mathcal{I}$ with $|\mathcal{I}| = \mathrm{poly}(n)$. It is also assumed that each vertex can access an infinite sequence of local random bits. Initially, each vertex knows nothing about the topology of the network except the set of edges incident to itself and the value of n.

Our quantum algorithm will be based on several classical distributed algorithms from the literature. A first crucial ingredient is the following recent result by Chang and Saranurak [7] that shows how to efficiently compute a good expander decomposition of the graph.

▶ **Theorem 3** ([7]). *In the classical CONGEST model, there is a $O(n^{0.1})$-round algorithm that computes with probability at least $1 - 1/\mathrm{poly}(n)$ a partition*

$$V = V_1 \cup V_2 \cup \cdots \cup V_s$$

of the vertex set V that satisfies the following two conditions:

- *for each $i \in [s]$, the subgraph induced by the vertex set V_i has mixing time $O(\mathrm{poly}(\log n))$;*
- *the number of inter-component edges (i.e., the number of edges with one endpoint in V_i and one endpoint in V_j, for $i \neq j$) is at most $|E|/10$.*

We will also use the following technical lemma from [6] that shows how to compute efficiently a new ID assignment that gives a good estimation of the degree of any vertex of the graph.

▶ **Lemma 4** (Lemma 4.1 in [6]). *In the classical CONGEST model, there is a $O(\mathrm{diam}(G) + \log n)$-round deterministic algorithm that computes a bijective map $\gamma\colon V \to \{1, \ldots, |V|\}$ and a function $d\colon \{1, \ldots, |V|\} \to \{0, 1, \ldots, \lfloor \log_2(n) \rfloor\}$ satisfying the following conditions:*

(i) *$\gamma(u) \leq \gamma(v)$ implies $\lfloor \log_2(\deg(u)) \rfloor \leq \lfloor \log_2(\deg(v)) \rfloor$ for any $u, v \in V$;*

(ii) *$d(\gamma(u)) = \lfloor \log_2(\deg(u)) \rfloor$ for all $u \in V$.*

More precisely, after running the algorithm each vertex u knows $\gamma(u)$ and can locally compute $d(y)$ for any $y \in \{0, 1, \ldots, |V|\}$.

We will also use the following result by Ghaffari, Kuhn and Su [13] about randomized routing in networks with small mixing time (see also the discussion in Section 3 of [7]).

▶ **Theorem 5** ([13]). *In the classical CONGEST model, there exists a $O(\mathrm{mix}(G) \cdot n^{o(1)})$-round algorithm that builds a distributed data structure. This data structure enables the vertices to implement the following routing task with probability at least $1 - 1/\mathrm{poly}(n)$ in $O(\mathrm{mix}(G) \cdot n^{o(1)})$ rounds: given a set of point-to-point routing requests, each given by the IDs of the corresponding source-destination pair and such that each vertex u is the source and the destination of at most $O(\deg(u))$ messages, delivers all the messages.*

Quantum distributed computing. We assume that the reader is familiar with the basic concepts of quantum computation and refer to, e.g., [24] for a good reference. The quantum CONGEST model is defined (see [10, 20] for details) as the quantum version of the classical CONGEST model, where the only difference is that each exchanged message consists of $O(\log n)$ quantum bits instead of $O(\log n)$ bits. In particular, initially the vertices of the network do not share any entanglement.

For the quantum CONGEST model, Le Gall and Magniez [20] introduced a framework for quantum distributed search, which can be seen as a distributed implementation of Grover's search [15], one of the most important centralized quantum algorithms. Let X be a finite set and $f\colon X \to \{0, 1\}$ be a Boolean function over X. Let u be an arbitrary vertex of the network (e.g., an elected leader). Assume that vertex u can evaluate the function f in r rounds: assume that there exists an r-round distributed checking procedure $\mathcal{C}$ such that vertex u, when receiving as input $x \in X$, outputs $f(x)$. Now consider the following problem: vertex u should find one element $x \in X$ such that $f(x) = 1$ (or report that no such element exists). The trivial strategy is to compute $f(x)$ for each $x \in X$ one by one, which requires $r|X|$ rounds. Ref. [20] showed that in the quantum CONGEST model this problem can be solved with probability at least $1 - 1/\mathrm{poly}(|X|)$ in $\tilde{O}(r\sqrt{|X|})$ rounds.

As explained in [20], the procedure $\mathcal{C}$ is often described as a classical (deterministic or randomized) procedure. It can then be quantized using standard techniques: one first transforms it to a reversible map using standard techniques [3] and then converts it into a quantum procedure.

FindTriangleInSubnetwork

Input: a connected subgraph $G^{\text{in}} = (V^{\text{in}}, E^{\text{in}})$ of G and a set of edges $E^{\text{out}} \subseteq E$ joining vertices in V^{in} to vertices in $V \setminus V^{\text{in}}$
(each vertex $u \in G$ knows if $u \in V^{\text{in}}$ and gets $\mathcal{N}_{E^{\text{in}}}(u)$ and $\mathcal{N}_{E^{\text{out}}}(u)$)

Promise: (i) $\text{mix}(G^{\text{in}}) = \text{poly}(\log n)$
(ii) $\deg_{E^{\text{in}}}(u) \geq \deg_{E^{\text{out}}}(u)$ for all $u \in V^{\text{in}}$

Goal: detect if there exists a triangle $\{u, v, w\}$ with $u, v, w \in V^{\text{in}} \cup V^{\text{out}}$ and $\{u, v\}, \{u, w\}, \{v, w\} \in E^{\text{in}} \cup E^{\text{out}}$

Figure 1 Problem FindTriangleInSubnetwork.

3 Reduction to Triangle finding over Subnetworks

In this section we present the reduction by Chang, Pettie and Zhang [6] from triangle finding over the whole network to triangle finding over subnetworks with small mixing time. In this section again, $G = (V, E)$ represents the whole network on which we want to solve the triangle finding problem, and we write $n = |V|$.

Triangle finding over subnetworks with small mixing time. We now present the computational problem considered, which we denote FindTriangleInSubnetwork (the description is also summarized in Figure 1).

The input of FindTriangleInSubnetwork is a connected subgraph $G^{\text{in}} = (V^{\text{in}}, E^{\text{in}})$ of G such that $\text{mix}(G^{\text{in}}) = \text{poly}(\log n)$, and a set of edges $E^{\text{out}} \subseteq E$ joining vertices in V^{in} to vertices in $V \setminus V^{\text{in}}$ that satisfies the following condition:

$$\deg_{E^{\text{in}}}(u) \geq \deg_{E^{\text{out}}}(u) \text{ for all } u \in V^{\text{in}}.$$

We write V^{out} the set of vertices in $V \setminus V^{\text{in}}$ that appear as an endpoint of an edge in E^{out}. The goal is to detect if there is a triangle in the subgraph of G induced by the edge set $E^{\text{in}} \cup E^{\text{out}}$. Note that such a triangle is either a triangle of G^{in}, or consists of two vertices in V^{in} and one vertex in V^{out}. Note that, while G^{in} (the subnetwork induced by E^{in}) has small mixing time, in general nothing can be said about the mixing time of the subnetwork induced by $E^{\text{in}} \cup E^{\text{out}}$.

The reduction. Chang, Pettie and Zhang [6] proved that when a good expander decomposition of the network is known, triangle finding over the whole network can be efficiently reduced to solving several instances of FindTriangleInSubnetwork. Combined with Theorem 3, this gives an efficient reduction from triangle finding to FindTriangleInSubnetwork, which we state in the following theorem. For completeness we include a sketch of the proof (we refer to [6, 7] for the details).

▶ **Theorem 6** ([6, 7]). *Assume that there exists an r-round distributed algorithm $\mathcal{A}$ that solves the problem FindTriangleInSubnetwork with probability at least $1-1/n^3$ and uses only the edges in $E^{in} \cup E^{out}$ for communication. Then there exists a $O(r \log n + n^{0.1})$-round distributed algorithm that solves the triangle finding problem over the whole graph $G = (V, E)$ with probability at least $1 - 1/\text{poly}(n)$.*

Sketch of the proof. The first step of the reduction computes a good decomposition of the whole network G in $O(n^{0.1})$ rounds using Theorem 3. Let $V = V_1 \cup V_2 \cup \cdots \cup V_s$, for some integer $s \leq n$, denote the decomposition and E^{inter} denote the set of inter-component edges. By definition, the set E^{inter} satisfies $|E^{\text{inter}}| \leq 0.1|E|$.

For any index $i \in [s]$, let us write $G_i = (V_i, E_i)$ the subgraph of G induced by V_i. We say that a vertex $u \in V_i$ is good if $\deg_{E_i}(u) \geq \deg_{E^{\text{inter}}}(u)$; otherwise we say that u is bad. Let E_i^{inter} be the set of edges in E^{inter} that are adjacent to a good vertex of G_i. Let E_i^{new} be the set of edges in E_i that are adjacent to a bad vertex of G_i. Define the set

$$E^{\text{new}} = E^{\text{inter}} \cup E_1^{\text{new}} \cup \cdots \cup E_s^{\text{new}}$$

and observe that $|E^{\text{new}}| \leq |E^{\text{inter}}| + 2|E^{\text{inter}}| \leq 0.3|E|$.

The triangles in G can be classified into the following four types.

- Type 1: triangles with three vertices in a same component G_i.
- Type 2: triangles with two vertices in a same component G_i and the third vertex in another component G_j, in which the two vertices in V_i are good.
- Type 3: triangles with two vertices in a same component G_i and the third vertex in another component G_j, in which at least one of the two vertices in V_i is bad.
- Type 4: triangles with three vertices in distinct components.

For each index $i \in [s]$, we use Algorithm $\mathcal{A}$ to solve FindTriangleInSubnetwork on instance $(G^{\text{in}}, E^{\text{out}})$ with $G^{\text{in}} = G_i$ and $E^{\text{out}} = E_i^{\text{inter}}$. A crucial point is that this can be done for all i's in parallel by "doubling" the bandwidth (i.e., by using $2r$ rounds in total), since Algorithm $\mathcal{A}$ on instance $(G^{\text{in}}, E^{\text{out}})$ only uses the edges in $E_i \cup E_i^{\text{inter}}$ for communication. Indeed, the communication networks are disjoint for all instances, except for the intercomponent edges that can be shared by two instances. This detects all the triangles of types 1 and 2 in the graph.

Another crucial observation is that all remaining potential triangles (i.e., the triangles of type 3 and 4) have their three edges contained in the set E^{new}. It is thus enough to recurse on this set, i.e., to repeat the same methodology with E replaced by E^{new}. Since $|E^{\text{new}}| \leq 0.3|E|$, after $O(\log n)$ levels of recursion the algorithm finishes. The overall complexity of this second part is thus $O(r \log n)$ rounds. ◀

4 Main Quantum Algorithm

In the classical CONGEST model, Chang, Pettie and Zhang [6] have shown that the problem FindTriangleInSubnetwork can be solved with high probability in $\tilde{O}(n^{1/3})$ rounds using only the edges in $E^{\text{in}} \cup E^{\text{out}}$ for communication, which leads to a $\tilde{O}(n^{1/3})$-round classical algorithm for triangle finding via Theorem 6. Our main technical result is the following theorem.

▶ **Theorem 7.** *In the quantum CONGEST model, there is a $\tilde{O}(n^{1/4})$-round quantum algorithm that solves the problem FindTriangleInSubnetwork with probability at least $1 - 1/n^3$ and uses only the edges in $E^{in} \cup E^{out}$ for communication.*

Theorem 1 then immediately follows from Theorem 6 and Theorem 7.

The goal of this section is to prove Theorem 7. For brevity we write $\mathcal{V} = V^{\text{in}} \cup V^{\text{out}}$ and $\mathcal{E} = E^{\text{in}} \cup E^{\text{out}}$. We also write $\bar{n} = |\mathcal{V}|$ and $\bar{m} = |\mathcal{E}|$. Note that $\bar{m} \geq \bar{n}/2$ since the graph $(\mathcal{V}, \mathcal{E})$ is connected. Let $S \subseteq \mathcal{V}$ be the subset of all vertices $u \in \mathcal{V}$ such that

$$\deg_{\mathcal{E}}(u) \geq \bar{m}/\sqrt{\bar{n}}.$$

Observe that $|S| \leq 2\sqrt{n}$.

In Section 4.1 below we present a $\tilde{O}(n^{1/4})$-round quantum algorithm that detects the existence of a triangle containing at least one vertex from S. We then describe, in Section 4.2, our main technical contribution: a $\tilde{O}(n^{1/4})$-round quantum algorithm that detects the existence of a triangle under the assumption $S = \emptyset$. The quantum algorithm of Theorem 7 then follows by combining these two algorithms, since (as already observed in prior works [6, 7]) detecting whether there exists a triangle with no vertex in S reduces to the case $S = \emptyset$.

Let us provide some explanations about why detecting the existence of a triangle with no vertex in S reduces to the case $S = \emptyset$. One natural approach is to focus on the subgraph where all the nodes in S are removed. This approach nevertheless does not immediately work since this may make the graph disconnected and may significantly reduce the number of edges (in which case Lemma 2 may not anymore be applicable). Instead, we now briefly describe a method that keeps the graph connected and the number of edges (almost) unchanged. The idea is simply to "virtually" replace each node $u \in S$ by a star of degree $\sqrt{n}$ and spread the $\deg_{\mathcal{E}}(u)$ incident edges of u evenly into the leaves of the star so that each leaf has $\deg_{\mathcal{E}}(u)/\sqrt{n} < \bar{m}/\sqrt{n}$ incident edges. Since $|S| \leq 2\sqrt{n}$, this can be done by introducing only $\Theta(\bar{n})$ virtual nodes.

4.1 Finding a triangle containing (at least) one high-degree vertex

In this subsection we describe how to detect, in $\tilde{O}(n^{1/4})$ rounds, the existence of a triangle with edges in $\mathcal{E}$ that contains at least one vertex from S. We will use the following lemma, which is a straightforward application of the framework for distributed quantum search described in Section 2, but can also be seen as an adaptation of the quantum protocol by Buhrman, Cleve and Wigderson [4] for the disjointness function in two-party quantum communication complexity.

▶ **Lemma 8.** *Consider any two adjacent vertices u and v, each owning a set $T_u \subseteq V$ and a set $T_v \subseteq V$, respectively. There is a quantum algorithm that checks if $T_u \cap T_v \neq \emptyset$ with high probability in $\tilde{O}(\sqrt{\min\{|T_u|, |T_v|\}})$ rounds. Moreover, this algorithm only uses communication along the edge $\{u, v\}$.*

Proof. Consider the subnetwork consisting only of the two vertices u and v and the edge $\{u, v\}$. Set $X = T_u$ and define the function

$$f(x) = \begin{cases} 1 & \text{if } x \in T_v, \\ 0 & \text{otherwise,} \end{cases}$$

for any $x \in X$. Obviously, for any $x \in X$, vertex u can compute the value $f(x)$ in 2 rounds. We can thus apply the quantum distributed search framework of Section 2 with vertex u acting as a leader, which enables u to check whether there exists $x \in X$ such that $x \in T_v$ in $\tilde{O}(\sqrt{|T_u|})$ rounds.

By symmetry there also exists a quantum algorithm that enables vertex v to decide whether $T_u \cap T_v \neq \emptyset$ in $\tilde{O}(\sqrt{|T_v|})$ rounds. Combining these two algorithms gives the claimed complexity. ◀

We now explain how our quantum algorithm works. First of all, observe that since each vertex u receives as input $\mathcal{N}_{E^{\text{in}}}(u)$ and $\mathcal{N}_{E^{\text{out}}}(u)$, each vertex knows whether it is in S or not. Each vertex first tells this to all its neighbors. This requires 1 round of communication.

Each vertex $u \in \mathcal{V}$ then computes, locally, the set

$$T_u = \mathcal{N}_{\mathcal{E}}(u) \cap S = (\mathcal{N}_{E^{\text{in}}}(u) \cup \mathcal{N}_{E^{\text{out}}}(u)) \cap S.$$

Note that for each edge $\{u,v\} \in \mathcal{E}$, there exists $w \in S$ such that $\{u,v,w\}$ is in a triangle with three edges in $\mathcal{E}$ if and only if $T_u \cap T_v \neq \emptyset$. Thus, for each edge $\{u,v\} \in \mathcal{E}$, the vertices u and v use the quantum algorithm of Lemma 8 to decide whether $T_u \cap T_v \neq \emptyset$ or not. Since this algorithm only communicates through the edge $\{u,v\}$, it can be applied in parallel to all edges $\{u,v\} \in \mathcal{E}$. This gives overall round complexity $\tilde{O}(\sqrt{|S|}) = \tilde{O}(n^{1/4})$.

4.2 Finding a triangle with only low-degree vertices

In this subsection we assume that the inequality

$$\deg_{\mathcal{E}}(u) < \bar{m}/\sqrt{\bar{n}}.$$

holds for all vertices $u \in \mathcal{V}$, i.e., we assume that $S = \emptyset$. We show how to detect, in $\tilde{O}(n^{1/4})$ rounds, the existence of a triangle with edges in $\mathcal{E}$ in this case as well.

Partitioning the set $\mathcal{V}$. Let us write $t = \left\lfloor \sqrt{\bar{n}}/(30 \log \bar{n}) \right\rfloor$. We randomly partition the set $\mathcal{V}$ into t subsets $\mathcal{V}_1, \ldots, \mathcal{V}_t$ as follows: each vertex $u \in \mathcal{V}$ selects an integer i uniformly at random in the set $[t]$ and joins the set $\mathcal{V}_i$. Vertex $u \in \mathcal{V}$ then tells its neighbors the value i, which can be done in 1 round. Each vertex therefore learns in which sets its neighbors have been included.

For any $i, j \in [t]$, let $E(\mathcal{V}_i, \mathcal{V}_j)$ denote all the edges in $\mathcal{E}$ with one endpoint in $\mathcal{V}_i$ and one endpoint in $\mathcal{V}_j$. Our analysis will rely on the following lemma.

▶ **Lemma 9.** *With probability $1 - 1/\mathrm{poly}(n)$, the following statement is true: for all $i, j \in [t]$,*

$$|E(\mathcal{V}_i, \mathcal{V}_j)| = O\left(\frac{\bar{m}(\log \bar{n})^2}{\bar{n}}\right).$$

Proof. Let us first consider the case $i = j$. We apply Lemma 2 over the graph generated by the vertex set $\mathcal{V}$ and using the probability $p = 1/t$. Note that

$$p^2 \bar{m} = \frac{\bar{m}}{(\lfloor \sqrt{\bar{n}}/(30 \log \bar{n}) \rfloor)^2} \geq \frac{\bar{n}/2}{(\sqrt{\bar{n}}/(30 \log \bar{n}))^2} \geq 400 (\log \bar{n})^2$$

and

$$\frac{\bar{m}p}{20 \log \bar{n}} = \frac{\bar{m}}{20 \log \bar{n} \left\lfloor \sqrt{\bar{n}}/(30 \log \bar{n}) \right\rfloor} \geq \frac{\bar{m}}{\sqrt{\bar{n}}},$$

which implies that the two conditions in Lemma 2 are satisfied.

In the case $i \neq j$, we apply Lemma 2 over the graph generated by the vertex set $\mathcal{V}$ again, but using the probability $p = 2/t$. The conclusion is the same. ◀

Partitioning the triples of indices. Let us write Λ the set of all triples (i,j,k) with $i, j, k \in [t]$, i.e., $\Lambda = [t] \times [t] \times [t]$. Let us partition this set into t sets $\Lambda_1, \ldots, \Lambda_t$, each containing t^2 triples, as follows. For each $\ell \in [t]$, define the set Λ_ℓ as:

$$\Lambda_\ell = \big\{(i, j, 1 + (i + j + \ell \bmod t)) \mid (i,j) \in [t] \times [t])\big\}.$$

Our analysis will rely on the following lemma, which immediately follows from the definition of the sets Λ_ℓ.

▶ **Lemma 10.** *The following statements are true for all $\ell \in [t]$ and all triples $(i,j,k) \in \Lambda_\ell$:*

- *there is no index $i' \in [t] \setminus \{i\}$ such that $(i', j, k) \in \Lambda_\ell$;*
- *there is no index $j' \in [t] \setminus \{j\}$ such that $(i, j', k) \in \Lambda_\ell$;*
- *there is no index $k' \in [t] \setminus \{k\}$ such that $(i, j, k') \in \Lambda_\ell$.*

Assigning the triples to vertices. For each $\ell \in [t]$, we assign the $t^2 = \Theta(\bar{n}/(\log \bar{n})^2)$ triples in Λ_ℓ to the vertices in V^{in}. The assignment should be made carefully, so that each vertex is assigned a number of triples proportional to its degree and, additionally, all the vertices know to which vertex each triple in Λ_ℓ is assigned. To achieve this goal we use the same approach as in [6], which is based on the ID assignment of Lemma 4.

We first apply Lemma 4 to the subnetwork G^{in} in order to obtain an ID assignment $\gamma\colon V^{\text{in}} \to \{1, \ldots, |V^{\text{in}}|\}$ and the degree estimator function

$$d\colon \{1, \ldots, |V^{\text{in}}|\} \to \{0, 1, \ldots, \lfloor \log_2 |V^{\text{in}}| \rfloor\}$$

satisfying the properties in the lemma. This requires $O(\text{diam}(G^{\text{in}}) + \log n) = O(\text{mix}(G^{\text{in}}) + \log n) = \text{poly}(\log n)$ rounds. For any vertex $u \in V^{\text{in}}$, define the quantity

$$r_u = \frac{2^{d(\gamma(u))}}{\bar{m}/\bar{n}}.$$

Note that since $d(\gamma(u)) = \lfloor \log_2(\deg_{E^{\text{in}}}(u)) \rfloor$, the quantity r_u is an approximation of the ratio between $\deg_{E^{\text{in}}}(u)$ and the average degree of the subgraph $(\mathcal{V}, \mathcal{E})$. Observe that

$$\sum_{u \in V^{\text{in}}} r_u \geq \sum_{u \in V^{\text{in}}} \frac{\deg_{E^{\text{in}}}(u)/2}{\bar{m}/\bar{n}} = \frac{|E^{\text{in}}|}{\bar{m}/\bar{n}} \geq \bar{n}/2,$$

since $|E^{\text{in}}| \geq \bar{m}/2$. Now define the quantity

$$q_u = \begin{cases} 0 & \text{if } r_u \leq 1/4, \\ \lceil r_u \rceil & \text{otherwise,} \end{cases}$$

and observe that

$$\sum_{u \in V^{\text{in}}} q_u \geq \sum_{u \in V^{\text{in}}} r_u \; - \frac{|V^{\text{in}}|}{4} \geq \frac{\bar{n}}{4} \geq t^2. \tag{1}$$

We can now explain the assignment of the triples from Λ_ℓ. We fix an arbitrary order (known to all the vertices of the network) on the triples of each Λ_ℓ. For concreteness, let us choose the lexicographic order. We start by assigning to the vertex $u_1 \in V^{\text{in}}$ such that $\gamma(u_1) = 1$ the first q_{u_1} triples of Λ_ℓ in the lexicographic order, then assign to the vertex $u_2 \in V^{\text{in}}$ such that $\gamma(u_2) = 2$ the next q_{u_2} triples from Λ_ℓ in the lexicographic order, and repeat the process until all the triples of Λ_ℓ have been assigned (Equation (1) guarantees that all triples are assigned by this process). For each vertex $u \in V^{\text{in}}$, let us write $\Lambda_\ell^u \subseteq \Lambda_\ell$ the set of triples assigned to u.

A crucial observation is that each vertex of the network can locally compute, for any $\ell \in [t]$ and any triple $(i, j, k) \in \Lambda_\ell$, the ID of the vertex to which (i, j, k) is assigned, since each vertex knows the value $d(y)$ for all $y \in \{0, 1 \ldots, |V^{\text{in}}|\}$.

Description of the quantum search algorithm. Consider the function

$$f\colon [t] \to \{0, 1\}$$

defined as follows. For any $\ell \in [t]$, we have $f(\ell) = 1$ if and only if there exists a triple $(i, j, k) \in \Lambda_\ell$ such that the graph G has a triangle with one vertex in the set $\mathcal{V}_i$, one vertex in $\mathcal{V}_j$, one vertex in $\mathcal{V}_k$ and its three edges in $\mathcal{E}$. Our quantum algorithm implements the quantum distributed search framework described in Section 2 with $X = [t]$ to detect if there exists one index $\ell \in [t]$ such that $f(\ell) = 1$. This approach obviously detects the existence of a triangle with three edges in $\mathcal{E}$, i.e., it solves our problem. The complexity of this approach, as explained in Section 2, is $\tilde{O}(\sqrt{t}\delta) = \tilde{O}(n^{1/4}\delta)$ rounds, where δ is the round complexity of the checking procedure. We present below a checking procedure with round complexity $\delta = \tilde{O}(\text{mix}(G^{\text{in}}))$. Since $\text{mix}(G^{\text{in}}) = \text{poly}(\log n)$ from our assumption on G^{in}, the overall round complexity is $\tilde{O}(n^{1/4})$, as claimed.

Description of the checking procedure. We now describe a classical randomized checking procedure that enables the leader, on an input $\ell \in [t]$, to evaluate the value $f(\ell)$. As explained in Section 2 such a classical procedure can then be converted into a quantum procedure using standard techniques. In the checking procedure, the leader first broadcasts the information "ℓ" to all the vertices of the network. This can be done in $\mathrm{diam}(G^{\mathrm{in}}) \leq \mathrm{mix}(G^{\mathrm{in}})$ rounds. Then each vertex $u \in V^{\mathrm{in}}$ checks, for each $(i,j,k) \in \Lambda_\ell^u$, whether there exists a triangle with one vertex in $\mathcal{V}_i$, one vertex in $\mathcal{V}_j$, one vertex in $\mathcal{V}_k$ with three edges in $\mathcal{E}$. In order to do that, vertex u simply needs to collect all the edges in $E(\mathcal{V}_i, \mathcal{V}_j) \cup E(\mathcal{V}_i, \mathcal{V}_k) \cup E(\mathcal{V}_j, \mathcal{V}_k)$ for each $(i,j,k) \in \Lambda_\ell^u$. From Lemma 9 and from the definition of the set Λ_ℓ, this requires

$$\tilde{O}\left(\frac{\bar{m}}{\bar{n}} \times q_u\right) = \tilde{O}\left(\frac{\bar{m}}{\bar{n}} \times \lfloor r_u \rfloor\right) = \tilde{O}\left(\deg_{E^{\mathrm{in}}}(u)\right)$$

incoming messages. Conversely, let us consider the number of outgoing messages needed to gather the information about the edges. Lemma 10 guarantees that the information about each edge only needs to be communicated to one vertex, which implies that each vertex u is the source of $\deg_{E^{\mathrm{in}}}(u)$ messages. Theorem 5 thus implies that the checking procedure can be implemented in $O(\mathrm{mix}(G^{\mathrm{in}}) \cdot n^{o(1)})$ rounds. The leader then checks if one of the vertices in V^{in} found a triangle, which can be done in $O(\mathrm{diam}(G^{\mathrm{in}})) = O(\mathrm{mix}(G^{\mathrm{in}}))$ rounds.

In order to reduce the complexity from $O(\mathrm{mix}(G^{\mathrm{in}}) \cdot n^{o(1)})$ to $\tilde{O}(\mathrm{mix}(G^{\mathrm{in}}))$ we simply need to modify slightly the routing scheme from [13], exactly as done in the classical case (see Section 3 of [7]).

References

1. Amir Abboud, Keren Censor-Hillel, Seri Khoury, and Christoph Lenzen. Fooling views: A new lower bound technique for distributed computations under congestion. *CoRR*, abs/1711.01623, 2017. arXiv:1711.01623.
2. Aleksandrs Belovs. Span programs for functions with constant-sized 1-certificates: extended abstract. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 77–84, 2012. doi:10.1145/2213977.2213985.
3. Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM Journal on Computing*, 18(4):766–776, 1989. doi:10.1137/0218053.
4. Harry Buhrman, Richard Cleve, and Avi Wigderson. Quantum vs. classical communication and computation. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 63–68, 1998. doi:10.1145/276698.276713.
5. Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. *Distributed Computing*, 32(6):461–478, 2019. doi:10.1007/s00446-016-0270-2.
6. Yi-Jun Chang, Seth Pettie, and Hengjie Zhang. Distributed triangle detection via expander decomposition. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 821–840, 2019. doi:10.1137/1.9781611975482.51.
7. Yi-Jun Chang and Thatchaphol Saranurak. Improved distributed expander decomposition and nearly optimal triangle enumeration. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 66–73, 2019.
8. Danny Dolev, Christoph Lenzen, and Shir Peled. "Tri, Tri Again": Finding triangles and small subgraphs in a distributed setting - (extended abstract). In *Proceedings of the International Symposium on Distributed Computing (DISC)*, pages 195–209, 2012. doi:10.1007/978-3-642-33651-5_14.
9. Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 367–376, 2014. doi:10.1145/2611462.2611493.

10 Michael Elkin, Hartmut Klauck, Danupon Nanongkai, and Gopal Pandurangan. Can quantum communication speed up distributed computation? In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 166–175, 2014. doi:10.1145/2611462.2611488.

11 Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. Networks cannot compute their diameter in sublinear time. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1150–1162, 2012. doi:10.1137/1.9781611973099.91.

12 Cyril Gavoille, Adrian Kosowski, and Marcin Markiewicz. What can be observed locally? In *Proceedings of the International Symposium on Distributed Computing (DISC)*, pages 243–257, 2009. doi:10.1007/978-3-642-04355-0_26.

13 Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. Distributed MST and routing in almost mixing time. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 131–140, 2017. doi:10.1145/3087801.3087827.

14 Mohsen Ghaffari and Jason Li. New distributed algorithms in almost mixing time via transformations from parallel algorithms. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, pages 31:1–31:16, 2018. doi:10.4230/LIPIcs.DISC.2018.31.

15 Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 212–219, 1996. doi:10.1145/237814.237866.

16 Juho Hirvonen, Joel Rybicki, Stefan Schmid, and Jukka Suomela. Large cuts with local algorithms on triangle-free graphs. *The Electronic Journal of Combinatorics*, 24(4):P4.21, 2017. URL: http://www.combinatorics.org/ojs/index.php/eljc/article/view/v24i4p21.

17 Taisuke Izumi and François Le Gall. Quantum distributed algorithm for the All-Pairs Shortest Path problem in the CONGEST-CLIQUE model. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 84–93, 2019.

18 Taisuke Izumi and François Le Gall. Triangle finding and listing in CONGEST networks. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 381–389, 2017. doi:10.1145/3087801.3087811.

19 François Le Gall. Improved quantum algorithm for triangle finding via combinatorial arguments. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 216–225, 2014. doi:10.1109/FOCS.2014.31.

20 François Le Gall and Frédéric Magniez. Sublinear-time quantum computation of the diameter in CONGEST networks. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 337–346, 2018. doi:10.1145/3212734.3212744.

21 François Le Gall, Harumichi Nishimura, and Ansis Rosmanis. Quantum advantage for the LOCAL model in distributed computing. In *Proceedings of the International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 49:1–49:14, 2019. doi:10.4230/LIPIcs.STACS.2019.49.

22 Troy Lee, Frédéric Magniez, and Miklos Santha. Improved quantum query algorithms for triangle finding and associativity testing. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1486–1502, 2013. doi:10.1137/1.9781611973105.107.

23 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.

24 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2011.

25 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. On the distributed complexity of large-scale graph computations. In *Proceedings of the Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 405–414, 2018. doi:10.1145/3210377.3210409.

26 Seth Pettie and Hsin-Hao Su. Distributed coloring algorithms for triangle-free graphs. *Information and Computation*, 243:263–280, 2015. doi:10.1016/j.ic.2014.12.018.

Lower Bounds for Arithmetic Circuits via the Hankel Matrix

Nathanaël Fijalkow
CNRS, LaBRI, Bordeaux, France
The Alan Turing Institute of data science, London, United Kingdom
Nathanael.Fijalkow@labri.fr

Guillaume Lagarde
LaBRI, Bordeaux, France
guillaume.lagarde@labri.fr

Pierre Ohlmann
Université de Paris, IRIF, CNRS, F-75013 Paris, France
Pierre.Ohlmann@irif.fr

Olivier Serre
Université de Paris, IRIF, CNRS, F-75013 Paris, France
Olivier.Serre@cnrs.fr

Abstract

We study the complexity of representing polynomials by arithmetic circuits in both the commutative and the non-commutative settings. To analyse circuits we count their number of parse trees, which describe the non-associative computations realised by the circuit.

In the non-commutative setting a circuit computing a polynomial of degree d has at most $2^{O(d)}$ parse trees. Previous superpolynomial lower bounds were known for circuits with up to $2^{d^{1/3-\epsilon}}$ parse trees, for any $\varepsilon > 0$. Our main result is to reduce the gap by showing a superpolynomial lower bound for circuits with just a small defect in the exponent for the total number of parse trees, that is $2^{d^{1-\varepsilon}}$, for any $\varepsilon > 0$.

In the commutative setting a circuit computing a polynomial of degree d has at most $2^{O(d \log d)}$ parse trees. We show a superpolynomial lower bound for circuits with up to $2^{d^{1/3-\varepsilon}}$ parse trees, for any $\varepsilon > 0$. When d is polylogarithmic in n, we push this further to up to $2^{d^{1-\varepsilon}}$ parse trees.

While these two main results hold in the associative setting, our approach goes through a precise understanding of the more restricted setting where multiplication is not associative, meaning that we distinguish the polynomials $(xy)z$ and $x(yz)$. Our first and main conceptual result is a characterization result: we show that the size of the smallest circuit computing a given non-associative polynomial is exactly the rank of a matrix constructed from the polynomial and called the Hankel matrix. This result applies to the class of all circuits in both commutative and non-commutative settings, and can be seen as an extension of the seminal result of Nisan giving a similar characterization for non-commutative algebraic branching programs. Our key technical contribution is to provide generic lower bound theorems based on analyzing and decomposing the Hankel matrix, from which we derive the results mentioned above.

The study of the Hankel matrix also provides a unifying approach for proving lower bounds for polynomials in the (classical) associative setting. We demonstrate this by giving alternative proofs of recent lower bounds as corollaries of our generic lower bound results.

2012 ACM Subject Classification Theory of computation → Circuit complexity; Theory of computation → Algebraic complexity theory

Keywords and phrases Arithmetic Circuit Complexity, Lower Bounds, Parse Trees, Hankel Matrix

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.24

Related Version A full version of the paper is available at `https://hal.archives-ouvertes.fr/hal-02440692`.

37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020).
Editors: Christophe Paul and Markus Bläser; Article No. 24; pp. 24:1–24:16

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

1 Introduction

The model of arithmetic circuits is the algebraic analogue of Boolean circuits: the latter computes Boolean functions and the former computes polynomials, replacing OR gates by addition and AND gates by multiplication. Computational complexity theory is concerned with understanding the expressive power of such models. A rich theory investigates the algebraic complexity classes **VP** and **VNP** introduced by Valiant [25]. A widely open problem in this area of research is to explicitly construct hard polynomials, meaning for which we can prove super polynomial lower bounds. To this day the best general lower bounds for arithmetic circuits were given by Baur and Strassen [4] for the polynomial $\sum_{i=1}^{n} x_i^d$, which requires $\Omega(n \log d)$ operations.

The seminal paper of Nisan [19] initiated the study of non-commutative computation: in this setting variables do not commute, and therefore xy and yx are considered as being two distinct monomials. Non-commutative computations arise in different scenarios, the most common mathematical examples being when working with algebras of matrices, group algebras of non-commutative groups or the quaternion algebra. A second motivation for studying the non-commutative setting is that it makes it easier to prove lower bounds which can then provide powerful ideas for the commutative case. Indeed, commutativity allows a circuit to rely on cancellations and to share calculations across different gates, making them more complicated to analyze.

1.1 Nisan's Characterization for ABP

The main result of Nisan [19] is to give a characterization of the smallest ABP computing a given polynomial. As a corollary of this characterization Nisan obtains exponential lower bounds for the non-commutative permanent against the subclass of circuits given by ABPs.

We sketch the main ideas behind Nisan's characterization, since our first contribution is to extend these ideas to the class of all non-associative circuits. An ABP is a layered graph with two distinguished vertices, a source and a target. The edges are labelled by affine functions in a given set of variables. An ABP computes a polynomial obtained by summing over all paths from the source to the target, with the value of a path being the multiplication of the affine functions along the traversed edges. Fix a polynomial f, and define following Nisan a matrix N_f whose rows and columns are indexed by monomials: for u, v two monomials, let $N_f(u, v)$ denote the coefficient of the monomial $u \cdot v$ in f.

The beautiful and surprisingly simple characterization of Nisan states that for a homogeneous (i.e., all monomials have the same degree) non-commutative polynomial f, the size of the smallest ABP computing f is exactly the rank of N_f. The key idea is that the computation of the polynomial in an ABP can be split into two parts: let r be a vertex in an ABP $\mathcal{C}$ computing the polynomial f, then we can split $\mathcal{C}$ into two ABPs, one with the original source and target r and the other one with source r and the original target. We let L_r and R_r denote the polynomials computed by these two ABPs. For u, v two monomials, we observe that the coefficient of uv in f is equal to $\sum_r L_r(u)R_r(v)$, where r ranges over all vertices of $\mathcal{C}$, $L_r(u)$ is the coefficient of u in L_r, and $R_r(v)$ is the coefficient of v in R_r. We see this as a matrix equality: $N_f = \sum_r L_r \cdot R_r$, where L_r is seen as a column vector, and R_r as a row vector. By subadditivity of the rank and since the product of a column vector by a row vector is a matrix of rank at most 1, this implies that $\operatorname{rank}(N_f)$ is bounded by the size of the ABP, yielding the lower bound in Nisan's result.

The crucial idea of splitting the computation of a monomial into two parts had been independently developed by Fliess when studying so-called *Hankel Matrices* in [9] to derive a very similar result in the field of *weighted automata*, which are finite state machines

recognising *words series*, i.e., functions from finite words into a field. Fliess' theorem [9, Th. 2.1.1] states that the size of the smallest weighted automaton recognising a word series f is exactly the rank of the Hankel matrix of f. The key insight to relate the two results is to see a non-commutative monomial as a finite word over the alphabet whose letters are the variables. Using this correspondence one can obtain Nisan's theorem from Fliess' theorem, observing that the Hankel matrix coincides with the matrix N_f defined by Nisan and that acyclic weighted automata correspond to ABPs. (We refer to an early technical report of this work for more details on this correspondence [8].)

1.2 Non-Associative Computations

Hrubeš, Wigderson and Yehudayoff in [12] drop the associativity rule and show how to define the complexity classes **VP** and **VNP** in the absence of either commutativity or associativity (or both) and prove that these definitions are sound in particular by obtaining the completeness of the permanent.

In the same way that a non-commutative monomial can be seen as a word, a non-commutative and non-associative monomial such as $(xy)(x(zy))$ can be seen as a tree, and more precisely as an ordered binary rooted tree whose leaves are labelled by variables. The starting point of our work was to exploit this connection. The work of Bozapalidis and Louscou-Bozapalidou [5] extends Fliess' result to trees; although we do not technically rely on their results they serve as a guide, in particular for understanding how to decompose trees.

Let us return to the key idea in Nisan's proof, which is to decompose the computation of an ABP into two parts. The way a monomial, e.g., $x_1x_2x_3\cdots x_d$, is evaluated in an ABP is very constrained, namely from left to right, or if we make the implicit non-associative structure explicit as $w = (\cdots(((x_1x_2)x_3)x_4)\cdots)x_d$. The decompositions of w into two monomials u, v are of the form $u = (\cdots((x_1x_2)x_3)\cdots)x_{i-1})$ and $v = (\cdots((\Box x_i)x_{i+1})\cdots)x_d$. Here $\Box$ is a new fresh variable (the *hole*) to be substituted by u. Moving to non-associative polynomials, a monomial is a tree whose leaves are labelled by variables. A *context* is a monomial over the set of variables extended with a new fresh one denoted $\Box$ and occurring exactly once. For instance the composition of the monomial $t = z((xx)y)$ with the context $c = (xy)((z\Box)y)$ is the monomial $c[t] = (xy)((z(z((xx)y)))y)$.

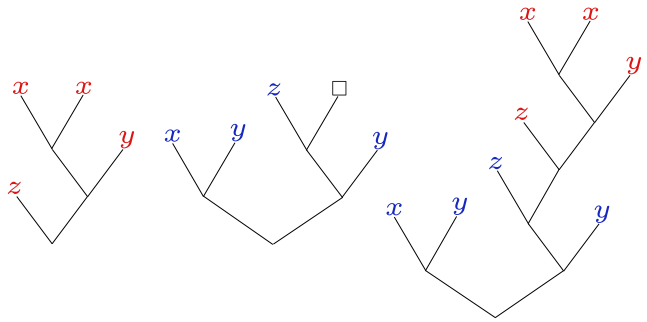


Figure 1 On the left hand side the monomial t, in the middle the context c, and on the right hand side the monomial $c[t]$.

Let f be a non-associative (possibly commutative) polynomial f, the *Hankel matrix* H_f of f is defined as follows: the rows of H_f are indexed by contexts and the columns by monomials, the value of $H_f(c, t)$ at row c and column t is the coefficient of the monomial $c[t]$ in f.

Extending Nisan's proof to computations in a *general circuit*, which are done along trees, we obtain a characterization in the non-associative setting.

▶ **Theorem 1.** *Let f be a non-associative homogeneous polynomial and let H_f be its Hankel matrix. Then, the size of the smallest circuit computing f is exactly rank(H_f).*

Note that this is a characterization result: the Hankel matrix exactly captures the size of the smallest circuit computing f (upper and lower bounds), exactly as in Nisan's result. Hence, understanding the rank of the Hankel matrix is equivalent to studying circuits for f. We recover and extend Nisan's characterization as a special case of our result.

Parse Trees

At an intuitive level, parse trees can be used to explain in what way a circuit uses the associativity rule. Consider the case of a circuit computing the (associative) monomial $2xyz$. Since this monomial corresponds to two non-associative monomials: $(xy)z$ and $x(yz)$, the circuit may sum different computations, for instance $3(xy)z - x(yz)$, which up to associativity is $2xyz$. We say that such a circuit contains two parse trees, corresponding to the two different ways of parenthesizing xyz.

The *shape* of a non-associative monomial is the tree obtained by forgetting the variables, e.g., the shape of $(z((xy)((xx)y)))$ is $(\_\ ((\_\ \_)((\_\ \_)\ \_)))$. The parse trees of a circuit $\mathcal{C}$ are the shapes induced by computations in $\mathcal{C}$.

Many interesting classes of circuits can be defined by restricting the set of allowed parse trees, both in the commutative and the non-commutative setting. The simplest such class is that of Algebraic Branching Programs (ABP) [19, 7, 21], whose only parse trees are left-combs, that is, the variables are multiplied sequentially. Lagarde, Malod and Perifel introduced in [16] the class of Unique Parse Tree circuits (UPT), which are circuits computing non-commutative homogeneous (but associative) polynomials such that all monomials are evaluated in the same non-associative way. The class of skew circuits [24, 2, 18, 17] and its extension small non-skew depth circuits [17], together with the class of unambiguous circuits [3] are all defined via parse tree restrictions. Last but not least, the class of k-PT circuits [3, 15, 23] is simply the class of circuits having at most k parse trees.

Contributions and Outline

In this paper we prove lower bounds for classes of circuits with parse tree restrictions, both in the commutative and non-commutative setting.

Our first and conceptually main contribution is the characterization result stated in Theorem 5 and proved in Section 2, which gives an algebraic approach to understanding circuits in the non-associative setting. All the subsequent results in this paper are based on this approach.

Our most technical developments are discussed in Section 3. We prove generic lower bound results by further analyzing and decomposing the Hankel matrix, with the following proof scheme. We consider a polynomial f in the associative setting. Let $\mathcal{C}$ be a circuit computing f. Forgetting about associativity we can see $\mathcal{C}$ as computing a non-associative polynomial $\tilde{f}$, which projects onto f, meaning is equal to f assuming associativity. This induces a set of linear constraints: for instance if the monomial xyz has coefficient 3 in f, then we know that $\tilde{f}((xy)z) + \tilde{f}(x(yz)) = 3$. We make use of the linear constraints to derive lower bounds on the rank of the Hankel matrix $H_{\tilde{f}}$, yielding a lower bound on the size of $\mathcal{C}$.

Sections 3.1 and 3.2 are devoted to the definition of parse trees and a classical tool for proving lower bounds, partial derivative matrices. We can already show at this point how Theorem 5 can be specialized to give a characterization result for UPT circuits, extending Nisan's result. (We note that a characterization result for UPT circuits was already known [16], we slightly improve on it.) As a corollary we obtain exponential lower bounds on the size of the smallest UPT circuit computing the permanent.

The final section is devoted to applications of our results, where we obtain superpolynomial and exponential lower bounds for various classes. In the results mentioned below, n is the number of variables, d is the degree of the polynomial, and k the number of parse trees. We note that the lower bounds hold for any (prime) n, any d, and any field.

We obtain alternative proofs of some known lower bounds: unambiguous circuits [3], skew circuits [17] and small non-skew depth circuits (obtaining a much shorter proof than [17]).

Our novel results are:

- *Slightly unbalanced circuits.* We extend the exponential lower bound from [17] on $\frac{1}{5}$-unbalanced circuits to $\left(\frac{1}{2}-\varepsilon\right)$-unbalanced circuits.
- *Slightly balanced circuits.* We derive a new exponential lower bound for ε-balanced circuits.
- *Circuits with k parse trees in the non-commutative setting.* We extend the superpolynomial lower bound of [15] from $k = 2^{d^{1/3-\varepsilon}}$ to $k = 2^{d^{1-\varepsilon}}$, the total number of possible non-commutative parse trees being $2^{O(d)}$.
- *Circuits with k parse trees in the commutative setting.* We substantially extend the superpolynomial lower bound from [3] from $k = d^{1/2-\varepsilon}$ to $k = 2^{d^{1/3-\varepsilon}}$, and even to $k = 2^{d^{1-\varepsilon}}$ when d is polylogarithmic in n.

Related Work

We argued that proving lower bounds in the non-commutative setting is easier, but this has not yet materialized since the best lower bound for general circuits in this setting is the same as in the commutative setting (by Baur and Strassen, already mentionned above). Indeed, recent impressive results suggest that this may be hard: Carmosino, Impagliazzo, Lovett, and Mihajlin [6] (essentially) proved that a lower bound in the non-commutative setting which would be slightly stronger than superlinear can be amplified to get strong lower bounds (even exponential, in some cases).

Most approaches for proving lower bounds rely on algebraic techniques and the rank of some matrix. A different and beautiful approach was investigated by Hrubeš, Wigderson and Yehudayoff [12] in the non-commutative setting through the study of the so-called *sum-of-squares problem*. Roughly speaking, the goal is to decompose $(x_1^2 + \cdots + x_k^2) \cdot (y_1^2 + \cdots + y_k^2)$ into a sum of n squared bilinear forms in the variables x_i and y_j. They show that almost any superlinear bound on n implies non-trivial lower bounds on the size of any non-commutative circuit computing the permanent.

The quest of finding lower bounds is deeply connected to another problem called polynomial identity testing (PIT) for which the goal is to decide whether a given circuit computes the formal zero polynomial. The connection was shown in [13], in which it is proved that providing an efficient deterministic algorithm to solve the problem implies strong lower bounds either in the arithmetic or boolean setting. PIT was widely investigated in the commutative and non-commutative settings for classes of circuits based on parse trees restrictions, see e.g., [22, 10, 1, 11, 23].

2 Characterizing Non-Associative Circuits

2.1 Basic Definitions

For an integer $d \in \mathbb{N}$, we let $[d]$ denote the integer interval $\{1, \dots, d\}$.

Polynomials

Let K be a field and let X be a set of ***variables***. Following [12] we consider that unless otherwise stated multiplication is neither commutative nor associative. We assume however that addition is commutative and associative, and that multiplication distributes over addition. A ***monomial*** is a product of variables in X and a polynomial f is a formal finite sum $\sum_i c_i m_i$ where m_i is a monomial and $c_i \in K$ is a non-zero element called the coefficient of m_i in f. We let $f(m_i)$ denote the coefficient of m_i in f, so that $f = \sum_i f(m_i) m_i$.

The ***degree*** of a monomial is defined in the usual way, i.e., $\deg(x) = 1$ when $x \in X$ and $\deg(m_1 m_2) = \deg(m_1) + \deg(m_2)$; the degree of a polynomial f is the maximal degree of a monomial in f. A polynomial is ***homogeneous*** if all its monomials have the same degree. Depending on whether we include the relations $u \cdot v = v \cdot u$ (commutativity) and $u \cdot (v \cdot w) = (u \cdot v) \cdot w$ (associativity) we obtain four classes of polynomials.

Unless otherwise specified, for a polynomial f we use n for the number of variables and d for the degree.

Trees and Contexts

The ***trees*** we consider have a single root and binary branching (every internal node has exactly two children). To account for the commutative and for the non-commutative setting we use either ***unordered trees*** or ***ordered trees***, the only difference being that in the case of ordered trees we distinguish the left child from the right child. We let ***Tree*** denote the set of trees (it will be clear from the context whether they are ordered or not). The size of a tree is defined as its number of leaves.

A non-associative monomial m is a tree with leaves labelled by variables. If m is non-commutative then it is an ordered tree, and if m is commutative then it is an unordered tree. We let $\mathbf{Tree}(X)$ denote the set of trees whose leaves are labelled by variables in X and $\mathbf{Tree_i}(X)$ denote the subset of such trees with i leaves, which are monomials of degree i.

In this paper we see a non-associative polynomial as a mapping from monomials to K, i.e., an element $f : \mathrm{Tree}(X) \to K$. To avoid possible confusion, let us insist that the notation $f(m)$ refers to the coefficient of the monomial m in the polynomial f, not to be confused with the evaluation of f at a given point. Similarly, a non-commutative associative homogeneous polynomial of degree d is seen as a mapping $f : X^d \to K$.

A (ordered or unordered) ***context*** is a tree with a distinguished leaf labelled by a special symbol called the ***hole*** and written $\Box$. We let $\mathbf{Context}(X)$ denote the set of contexts whose leaves are labelled by variables in X. Given a context c and a tree t we construct a new tree $c[t]$ by substituting the hole of c by t. This operation is defined in both ordered and unordered settings.

Hankel Matrices

Let f be a non-associative polynomial. The ***Hankel matrix*** H_f of f is the matrix whose rows are indexed by contexts and columns by monomials and such that the value of H_f at row c and column t is the coefficient of the monomial $c[t]$ in f.

Arithmetic Circuits

An (arithmetic) ***circuit*** is a directed acyclic graph such that the vertices are of three types:

- input gates: they have in-degree 0 and are labelled by variables in X,
- addition gates: they have arbitrary in-degree, an output value in K, and a weight $w(a) \in K$ on each incoming arc a,
- multiplication gates: they have in-degree 2, and we distinguish between the left child and the right child.

Each gate v in the circuit computes a polynomial f_v which we define by induction.

- An input gate labelled by a variable $x \in X$ computes the polynomial x.
- An addition gate v with n arcs incoming from gates $v_1, \dots, v_n$ and with weights $\alpha_1, \dots, \alpha_n$, computes the polynomial $\alpha_1 f_{v_1} + \cdots + \alpha_n f_{v_n}$.
- A multiplication gate with left child u and right child v computes the polynomial $f_u f_v$.

The circuit itself computes a polynomial given by the sum over all addition gates of the output value times the polynomial computed by the gate. Note that it is slightly unusual that all addition gates contribute to the circuit; one can easily reduce to the classical case where there is a unique output addition gate by adding an extra gate.

To define the size of a circuit we make a syntactic assumption: each arc is either coming from, or going to (but not both), an addition gate. This is a small assumption which can be lifted at the price of a linear blow-up. The ***size*** of a circuit $\mathcal{C}$ is denoted $|\mathcal{C}|$ and defined to be its number of addition gates. Note that this is how the size of ABPs is defined, it will be a convenient definition here since our characterization result captures the exact size of the smallest circuit computing a given polynomial.

Note that the definitions we gave above do not depend on which of the four settings we consider: commutative or non-commutative, associative or non-assocative.

Consider the circuit on the left hand side of Figure 2: it computes the polynomial $7y^2 + 2xy + yx$, which in the commutative setting is equal to $7y^2 + 3xy$.

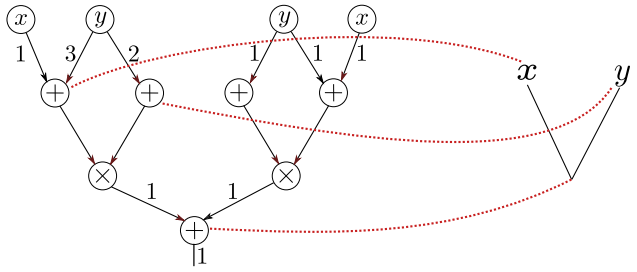


Figure 2 On the left hand side a circuit computing the polynomial $7y^2 + 2xy + yx$, which in the commutative setting is equal to $7y^2 + 3xy$. The only addition gate with a non-zero output value is at the bottom, its output value is 1. On the right hand side the monomial xy, seen as non-associative. The dashed red arrow show one run of the circuit over this monomial.

2.2 The Characterization

This section aims at proving the characterization stated in Theorem 5. It extends Nisan's characterization of non-commutative ABPs to general circuits in the non-associative setting. The result holds for both commutative and non-commutative settings, the proof being the same up to cosmetic changes.

The key step to go from ABPs to general circuits is the following: the polynomial computed by an ABP is the sum over the *paths* of the underlying graph, whereas in a general circuit the sum is over *trees*. We formalize this in the next definition by introducing *runs* of a circuit. The definition is given in the non-commutative setting but easily adapts to the commutative setting as explained in Remark 3.

▶ **Definition 2.** *Let $\mathcal{C}$ be a circuit and $V_\oplus$ denote its set of addition gates. Let $t \in Tree(X)$ be a monomial. A **run of $\mathcal{C}$ over t** is a map ρ from nodes of t to $V_\oplus$ such that*

- **(i)** *A leaf of t with label $x \in X$ is mapped to a gate with a non-zero edge incoming from an input gate labelled by x.*
- **(ii)** *If n is a node of t with left child n_1 and right child n_2, then $\rho(n)$ has a non-zero edge incoming from a multiplication gate with left child $\rho(n_1)$ and right child $\rho(n_2)$.*
- **(iii)** *The root of t is mapped to a gate with non-zero output value.*

*The **value** $val(\rho)$ of ρ is a non-zero element in K defined as the product of the weights of the edges mentioned in items (i) and (ii) together with the output value of $\rho(r)$, r being the root of t.*

We write by a small abuse of notation $\rho : t \to V_\oplus$ for runs of $\mathcal{C}$ over t.

We refer to Figure 2 for an example of a run over the monomial xy. The value of the run is 2.

▶ Remark 3. In the commutative setting we simply replace item (ii) by: "if n is a node of t with children n_1, n_2, then $\rho(n)$ has a non-zero edge incoming from a multiplication gate with children $\rho(n_1), \rho(n_2)$".

A run of $\mathcal{C}$ over a monomial t additively contributes to the coefficient of t in the polynomial computed by $\mathcal{C}$, leading to the following lemma.

▶ **Lemma 4.** *Let $\mathcal{C}$ be a circuit computing the non-associative polynomial $f : Tree(X) \to K$. Then the coefficient $f(t)$ of a monomial $t \in Tree(X)$ in f is equal to*

$$\sum_{\rho : t \to V_\oplus} val(\rho).$$

We may now state and prove our cornerstone result, which holds in both the commutative and non-commutative settings.

▶ **Theorem 5.** *Let $f : Tree(X) \to K$ be a non-associative polynomial, H_f be its Hankel matrix, and $\mathcal{C}$ be a circuit computing f. Then $|\mathcal{C}| \geq rank(H_f)$. Moreover, if f is homogeneous this bound is tight, meaning there exists a circuit $\mathcal{C}$ computing f of size $rank(H_f)$.*

An interesting feature of this theorem is that the upper bound is effective: given a homogenous polynomial one can construct a circuit computing this polynomial of size $\mathrm{rank}(H_f)$.

Here, we only prove the lower bound as the upper bound is not used in the rest of the paper. The proof of the lower bound follows the same lines as Nisan's original proof for non-commutative ABPs [19].

Proof. Let $\mathcal{C}$ be a circuit computing the non-associative polynomial $f : \mathrm{Tree}(X) \to K$. Let $V_\oplus$ denote the set of addition gates of $\mathcal{C}$. To bound the rank of the Hankel matrix H_f by $|\mathcal{C}| = |V_\oplus|$ we show that H_f can be written as the sum of $|V_\oplus|$ matrices each of rank at most 1.

For each $v \in V_\oplus$ we define two circuits which decompose the computations around v. Let $\mathcal{C}_1^v$ be the restriction of $\mathcal{C}$ to descendants of v, and $\mathcal{C}_2^v$ be a copy of $\mathcal{C}$ with just an extra input gate labelled by a fresh variable $\square \notin X$ with a single outgoing edge with weight 1 going to v.

We let $f^v : \mathrm{Tree}(X) \to K$ denote the polynomial computed by $\mathcal{C}_1^v$ and $g^v : \mathrm{Context}(X) \to K$ denote the restriction of the polynomial computed by $\mathcal{C}_2^v$ to $\mathrm{Context}(X) \subseteq \mathrm{Tree}(X \sqcup \{\square\})$.

We show the equality

$$H_f(c,t) = \sum_{v \in V_\oplus} f^v(t) g^v(c).$$

Fix a monomial $t \in \mathrm{Tree}(X)$ and a context $c \in \mathrm{Context}(X)$. We let $n_\square$ denote the leaf of c labelled by $\square$, which is also the root of t and the node to which t is substituted with in $c[t]$. Relying on Lemma 4, we calculate the coefficient $f(c[t])$ of $c[t]$ in f.

$$\begin{aligned} f(c[t]) &= \sum_{\rho : c[t] \to V_\oplus} \mathrm{val}(\rho) = \sum_{v \in V_\oplus} \sum_{\substack{\rho : c[t] \to V_\oplus \\ \rho(n_\square) = v}} \mathrm{val}(\rho) = \sum_{v \in V_\oplus} \sum_{\substack{\rho_1^v : t \to V_\oplus \\ \rho_1^v(n_\square) = v}} \sum_{\substack{\rho_2^v : c \to V_\oplus \\ \rho_2^v(n_\square) = v}} \mathrm{val}(\rho_1^v)\mathrm{val}(\rho_2^v) \\ &= \sum_{v \in V_\oplus} \sum_{\substack{\rho_1^v : t \to V_\oplus \\ \rho_1^v(n_\square) = v}} \mathrm{val}(\rho_1^v) \sum_{\substack{\rho_2^v : c \to V_\oplus \\ \rho_2^v(n_\square) = v}} \mathrm{val}(\rho_2^v) = \sum_{v \in V_\oplus} f^v(t) g^v(c). \end{aligned}$$

Let $M_v \in K^{\mathrm{Tree}(X) \times \mathrm{Context}(X)}$ be the matrix given by $M_v(t,c) = f^v(t)g^v(c)$: its rank is at most one as M_v is the product of a column vector by a row vector. The previous equality reads in matrix form $H_f = \sum_{v \in V_\oplus} M_v$. Hence, we obtain the announced lower bound using rank subadditivity:

$$\mathrm{rank}\,(H_f) = \mathrm{rank}\left(\sum_{v \in V_\oplus} M_v\right) \leq \sum_{v \in V_\oplus} \mathrm{rank}\,(M_v) \leq |V_\oplus| = |\mathcal{C}|. \qquad \blacktriangleleft$$

The remainder of this paper consists in applying Theorem 5 to obtain lower bounds in various cases. To this end we need a better understanding of the Hankel matrix: in Section 3 we introduce a few concepts and develop decomposition theorems for the Hankel matrix.

3 Decomposing the Hankel Matrix

Our decomposition of the Hankel matrix relies on the notion of parse trees and partial derivative matrices, which we formally introduce now.

3.1 Parse Trees

With any monomial $t \in \mathrm{Tree}(X)$ we associate its ***shape*** $\mathrm{shape}(t) \in \mathrm{Tree}$ as the tree obtained from t by removing the labels at the leaves.

▶ **Definition 6.** *Let $\mathcal{C}$ be a circuit computing a non-commutative non-associative polynomial f. A **parse tree** of $\mathcal{C}$ is any shape $s \in Tree$ for which there exists a monomial $t \in Tree(X)$ whose coefficient in f is non-zero and such that $s = shape(t)$. We let $PT(\mathcal{C}) = \{shape(t) \mid f(t)\ non\text{-}zero\}$.*

3.2 Partial Derivative Matrices

We now introduce a well known tool for proving circuit lower bounds, namely, partial derivative matrices. For $A \subseteq [d]$ of size i , $u \in X^{d-i}$, and $v \in X^i$, we define the monomial $u \otimes_A v \in X^d$: it is obtained by interleaving u and v with u taking the positions indexed by $[d] \setminus A$ and v the positions indexed by A. For instance $x_1x_2 \otimes_{\{2,4\}} y_1y_2 = x_1y_1x_2y_2$.

▶ **Definition 7.** *Let f be a homogeneous non-commutative associative polynomial. Let $A \subseteq [d]$ be a set of positions of size i.*

*The **partial derivative matrix** $M_A(f)$ of f with respect to A is defined as follows: the rows are indexed by $u \in X^{d-i}$ and the columns by $v \in X^i$, and the value of $M_A(f)(u,v)$ is the coefficient of the monomial $u \otimes_A v$ in f.*

▶ **Example 8.** Let $f = xyxy + 3xxyy + 2xxxy + 5yyyy$ and $A = \{2,4\}$. Then $M_A(f)$ is given below.

	$\_x\_x$	$\_x\_y$	$\_y\_x$	$\_y\_y$
$x\_x\_$	0	2	0	1
$y\_x\_$	0	0	0	0
$x\_y\_$	0	3	0	0
$y\_y\_$	0	0	0	5

We define a distance $\mathrm{dist} : \mathcal{P}([d]) \times \mathcal{P}([d]) \to \mathbb{N}$ on subsets of $[d]$ by letting $\mathrm{dist}(A,B)$ be the minimal number of additions and deletions of elements of $[d]$ to go from A to B, assuming that complementing is for free. Formally, $\mathrm{dist}(A,B) = \min\{|\Delta(A,B)|, |\Delta(A^{\mathrm{C}},B)|\}$, where $\Delta(A,B) = (A \setminus B) \cup (B \setminus A)$ is the symmetric difference between A and B.

The following lemma (see e.g., [17]) informally says that, if A and B are close to each other, then the ranks of the corresponding partial derivative matrices are close to each other as well.

▶ **Lemma 9.** *Let f be a homogeneous non-commutative associative polynomial of degree d with n variables. Then, for any subsets $A, B \subseteq [d]$, $\mathrm{rank}(M_A(f)) \leq n^{dist(A,B)} \mathrm{rank}(M_B(f))$.*

At this point, we have the material in hands to describe a precise characterization of the size of the smallest Unique Parse Tree circuit which computes a given polynomial. We take this short detour before moving on to our core lower bound results.

3.3 Characterization of smallest Unique Parse Tree Circuit

Unique Parse Tree (UPT) circuits are non-commutative associative circuits with a unique parse tree. They were first introduced in [16]. Our techniques allow a slight improvement and a better understanding of their results. We obtain a small improvement since the original result requires a normal form which can lead to an exponential blow-up.

Given a shape $s \in \mathrm{Tree}$ of size d, i.e., with d leaves and a node v of s, we let s_v denote the subtree of s rooted in v, and $I_v \subseteq [d]$ denote the interval of positions of the leaves of s_v in s. We say that $s' \in \mathrm{Tree}$ is a subshape of s if $s' = s_v$ for some v, and that I is spanned by s if $I = I_v$ for some v.

Let $f : X^d \to K$ be a homogeneous non-commutative associative polynomial of degree d, let $s \in \mathrm{Tree}$ be a shape of size d, and let s' be a subshape of s such that $v_1, \ldots, v_p$ are all the nodes v of s such that $s' = s_v$. We define

$$M_{s'} = \begin{bmatrix} M_{I_{v_1}}(f) \\ M_{I_{v_2}}(f) \\ \vdots \\ M_{I_{v_p}}(f) \end{bmatrix}.$$

▶ **Theorem 10.** *Let $f : X^d \to K$ be a homogeneous non-commutative associative polynomial and let $s \in Tree$ be a shape of size d. Then the smallest UPT circuit with shape s computing f has size exactly*

$$\sum_{s' \text{ subshape of } s} rank(M_{s'}) .$$

Proof. Let $\mathcal{C}$ be a UPT circuit with shape s computing f. We let $\tilde{f}$ denote the non-associative polynomial computed by $\mathcal{C}$. Since $\mathcal{C}$ is UPT with shape s, $\tilde{f}$ is the *unique* non-associative polynomial which is non-zero only on trees with shape s and projects to f, i.e., $\tilde{f}(t) = f(u)$ if $\text{shape}(t) = s$ and t is labelled by u, and $\tilde{f}(t) = 0$ otherwise.

In particular, the size of the smallest UPT circuit with shape s computing f is the same as the size of the smallest circuit computing $\tilde{f}$, which thanks to Theorem 5 is equal to the rank of the Hankel matrix $H_{\tilde{f}}$.

The Hankel matrix of $\tilde{f}$ may be non-zero only on columns indexed by trees whose shapes s' are subshapes of s, and on such columns, non-zero values are on rows corresponding to a context obtained from s by replacing an occurrence of s' by $\Box$. The corresponding blocks are precisely the matrices $M_{s'}$, and are placed in a diagonal fashion, hence the lower bound. ◀

Theorem 10 can be applied to concrete polynomials, for instance to the permanent of degree d.

▶ **Corollary 11.** *Let $s \in Tree$ be a shape. The smallest UPT circuit with shape s computing the permanent has size*

$$\sum_{v \text{ node of } s} \binom{d}{|I_v|},$$

where I_v is the set of leaves in the subtree rooted at v in s. In particular, this is always larger than $\binom{d}{d/3}$.

Applied to s being a left-comb, Corollary 11 yields that the smallest ABP computing the permanent has size $2^d + d$. Applied to s being a complete binary tree of depth $k = \log d$, the size of the smallest UPT is $\Theta\left(\frac{2^d}{d}\right)$, showing that this circuit is more efficient than any ABP.

3.4 General Roadmap

We now get to the technical core of the paper where we establish generic lower bounds theorems through a decomposition of the Hankel matrix, that we will later instantiate in Section 4 to concrete classes of circuits. We first restrict ourselves to the non-commutative setting. Our first decomposition, Theorem 12, seems to capture mostly previously known techniques. However, the second, more powerful decomposition, Theorem 13, takes advantage of the global shape of the Hankel matrix. Doing so allows to go beyond previous results only hinging around considering partial derivatives matrices which only turn out to be isolate slices of the Hankel matrix.

We later explain in Section 3.6 how to extend the study to the commutative case.

Let f be a (commutative or non-commutative) polynomial for which we want to prove lower bounds. Consider a circuit $\mathcal{C}$ which computes f, and let $\tilde{f}$ be the non-associative polynomial computed by $\mathcal{C}$. Our aim is, following Theorem 5, to give lower bounds on the rank of the Hankel matrix $H_{\tilde{f}}$. We know that the $\tilde{f}$ and f are equal up to associativity, which provides linear relations among the coefficients of $H_{\tilde{f}}$.

The bulk of the technical work is to reorganize the rows and columns of $H_{\tilde{f}}$ in order to decompose it into blocks which may be identified as partial derivative matrices with respect to some subsets $A_1, A_2, \cdots \subseteq [d]$, of some associative polynomials which depend on $\tilde{f}$ and sum to f. The number and choice of these subsets depend on the parse trees of the circuit $\mathcal{C}$.

Now, assume there exists a subset $A \subseteq [d]$ which is at distance at most δ to each A_i. Losing a factor of n^δ on the rank through the use of Lemma 9 we reduce the aforementioned blocks of $H_{\tilde{f}}$ to partial derivatives with respect to A. Such matrices can then be summed to recover the partial derivative matrix of f with respect to A, yielding in the lower bound a (dominating) factor of $\operatorname{rank}(M_A(f))$.

3.5 Generic Lower Bounds in the Non-commutative Setting

Following the general roadmap described above, we obtain a first generic lower bound result.

▶ **Theorem 12.** *Let $f : X^d \to K$ be a non-commutative homogeneous polynomial computed by a circuit $\mathcal{C}$. Let $A \subseteq [d]$ and $\delta \in \mathbb{N}$ such that all parse trees of $\mathcal{C}$ span an interval at distance at most δ from A. Then $\mathcal{C}$ has size at least $\operatorname{rank}(M_A(f))\, n^{-\delta} |PT(\mathcal{C})|^{-1}$.*

The crux to prove Theorem 12 is to identify for each parse tree s of $\mathcal{C}$ a block in $H_{\tilde{f}}$ containing the partial derivative matrix $M_{I(s)}(f_s)$ where f_s is the polynomial corresponding to the contribution of the parse tree s in the computation of f and $I(s)$ is an interval spanned by s.

However, we do not consider in this analysis how these blocks are located relative to each other. A more careful analysis of $H_{\tilde{f}}$ consists in grouping together all parse trees that lead to the same spanned interval. Aligning and then summing these blocks we remove the dependence in $|\mathrm{PT}(\mathcal{C})|$ and instead use d^2 which is the total number of possibly spanned intervals of $[d]$. This yields Theorem 13.

▶ **Theorem 13.** *Let f be a non-commutative homogeneous polynomial computed by a circuit $\mathcal{C}$. Let $A \subseteq [d]$ and $\delta \in \mathbb{N}$ such that all parse trees of $\mathcal{C}$ span an interval at distance at most δ from A. Then $\mathcal{C}$ has size at least $\operatorname{rank}(M_A(f))\, n^{-\delta} d^{-2}$.*

As we shall see in Section 4 the lower bounds we obtain using Theorem 12 match known results, while using Theorem 13 yields substantial improvements.

3.6 Extending to the Commutative Setting

We explain how to extend the notions of parse trees and the generic lower bound theorems to the commutative setting.

Let $X = X_1 \sqcup X_2 \sqcup \cdots \sqcup X_d$ be a partition of the variable set X. A monomial is ***set-multilinear*** with respect to the partition if it is the product of exactly one variable from each set X_i, and a polynomial is set-multilinear if all its monomials are.

The permanent and the determinant of degree d are set-multilinear with respect to the partition $X = X_1 \sqcup X_2 \sqcup \cdots \sqcup X_d$ where $X_i = \{x_{i,j}, j \in [d]\}$. The iterated matrix multiplication polynomial is another example of an important and well-studied set-multilinear polynomial. The partial derivative matrix also make sense in the realm of set-multilinear polynomials.

▶ **Definition 14.** *Let $X = X_1 \sqcup X_2 \sqcup \cdots \sqcup X_d$, f be a set-multilinear polynomial of degree d, and $A \subseteq [d]$ be a set of indices. The **partial derivative matrix $M_A(f)$** of f with respect to A is defined as follows: the rows are indexed by set-multilinear monomials g with respect to the partition $\bigsqcup_{i \notin A} X_i$ and the columns are indexed by set-multilinear monomials h with respect to the partition $\bigsqcup_{i \in A} X_i$. The value of $M_A(f)(g,h)$ is the coefficient of the monomial $g \cdot h$ in f.*

The notion of shape was defined by [3], and it slightly differs from the non-commutative case because we need to keep track of the indices of the variable sets given by the partition from which the variables belong. More precisely, given a partition of $X = X_1 \sqcup X_2 \sqcup \cdots \sqcup X_d$, we associate to any monomial $t \in \mathrm{Tree}(X)$ of degree d its ***shape*** $\mathrm{shape}(t) \in \mathrm{Tree}([d])$ defined as the tree obtained from t by replacing each label by its index in the partition. We let $\mathcal{T}_d \subseteq \mathrm{Tree}([d])$ denote the set of trees such that all elements of $[d]$ appear as a label of a leaf.

Let $\mathcal{C}$ be a commutative circuit. We let $\tilde{f}$ denote the commutative non-associative polynomial computed by C when it is seen as non-associative. A ***parse tree*** of $\mathcal{C}$ is any shape $s \in \mathcal{T}_d$ for which there exists a monomial $t \in \mathrm{Tree}(X)$ whose coefficient in $\tilde{f}$ is non-zero and such that $s = \mathrm{shape}(t)$. Hence, we let $\mathrm{PT}(\mathcal{C}) = \{\mathrm{shape}(t) \mid \tilde{f}(t) \text{ non-zero}\} \cap \mathcal{T}_d$.

Given a shape $s \in \mathrm{Tree}([d])$ with d leaves and a node v of s, we let s_v denote the subtree rooted at v and $A_v \subseteq [d]$ denote the set of labels appearing on the leaves of s_v.

Following the same roadmap as in the non-commutative setting we obtain the following counterpart of Theorem 12. We assume that the set of variables is partitioned into d parts of equal size n (this is a natural setting for polynomials such as the determinant, the permanent or the iterated matrix multiplication). In particular, it means that the polynomials we consider are of degree d and over nd variables.

▶ **Theorem 15.** *Let f be a set-multilinear polynomial computed by a circuit $\mathcal{C}$. Let $A \subseteq [d]$ and $\delta \in \mathbb{N}$ such that all parse trees of $\mathcal{C}$ span a subset at distance at most δ from A. Then $\mathcal{C}$ has size at least $\mathrm{rank}(M_A(f))\, n^{-\delta} |PT(\mathcal{C})|^{-1}$.*

A notable difference with the non-commutative setting is that now parse trees no longer span intervals of $[d]$ but subsets of $[d]$. As a consequence, the technique used to prove Theorem 13 groups together blocks corresponding to the same *subset* of $[d]$ and therefore the multiplicative factor is now 2^{-d} as there are 2^d such subsets.

▶ **Theorem 16.** *Let f be a set-multilinear polynomial computed by a circuit $\mathcal{C}$. Let $A \subseteq [d]$ and $\delta \in \mathbb{N}$ such that all parse trees of $\mathcal{C}$ span a subset at distance at most δ from A. Then $\mathcal{C}$ has size at least $\mathrm{rank}(M_A(f))\, n^{-\delta} 2^{-d}$.*

While in the non-commutative setting, Theorem 13 strengthens Theorem 12 (when d^2 is small), this is no longer the case in the commutative setting. Indeed, the maximal number of commutative parse trees being roughly $d!$ (the exact asymptotic is $\frac{\sqrt{2-\sqrt{2}}d^{d-1}}{e^d(\sqrt{2}-1)^{d+1}}$, see e.g., `https://oeis.org/A036774`), Theorem 15 and Theorem 16 are incomparable.

4 Applications

In this section we instantiate our generic lower bound theorems on concrete classes of circuits. We first show how the weaker version (Theorem 12) yields the best lower bounds to date for skew and small non-skew depth circuits. Extending these ideas we obtain exponential lower bounds for $\left(\frac{1}{2} - \varepsilon\right)$-unbalanced circuits, an extension of skew circuits which are just slightly unbalanced. We also adapt the proof to ε-balanced circuits, which are slightly balanced, then move on to our main results, which concern circuits with many parse trees.

High-ranked polynomials

The lower bounds we state below hold for any polynomial whose partial derivative matrices with respect to either a fixed subset A or all subsets have large rank. Such polynomials exist for all fields in both the commutative and non-commutative settings, and can be explicitly

constructed. For instance the so-called Nisan-Wigderson polynomial given in [14] (inspired by the notion of designs by Nisan and Wigderson [20]) has this property. It are given by

$$NW_{n,d} = \sum_{\substack{h \in \mathbb{F}_n[z] \\ \deg(h) \leq d/2}} \prod_{i=1}^{d} x_{i,h(i)},$$

where $\mathbb{F}_n[z]$ denotes univariate polynomials with coefficients in the finite field of prime order n. The fact that there exists a unique polynomial $h \in \mathbb{F}_n[z]$ of degree at most $d/2$ which takes $d/2$ given values at $d/2$ given positions exactly implies that the partial derivative matrix of $NW_{n,d}$ with respect to any $A \subseteq [d]$ of size $d/2$ is a permutation matrix. This is then easily extended to any $A \subseteq [d]$.

4.1 Skew, Slightly Unbalanced, Slightly Balanced and Small Non-Skew Depth Circuits

We show how using Theorem 12 yields exponential lower bounds for four classes of circuits in the non-commutative setting.

Skew Circuits

A circuit $\mathcal{C}$ is ***skew*** if all its parse trees are skew, meaning that each node has at least one of its children which is a leaf. As a direct application of Theorem 12, we obtain the following result.

▶ **Theorem 17.** *Let f be a homogeneous non-commutative polynomial such that $M_{[d/4+1,3d/4]}(f)$ has full rank $n^{d/2}$. Then any skew circuit computing f has size at least $2^{-d}n^{d/4}$.*

Slightly unbalanced circuits

A circuit $\mathcal{C}$ computing a homogeneous non-commutative polynomial of degree d is said to be ***α-unbalanced*** if every multiplication gate has at least one of its children which computes a polynomial of degree at most αd.

▶ **Theorem 18.** *Let f be a homogeneous non-commutative polynomial such that $M_{[d/4+1,3d/4]}(f)$ has full rank $n^{d/2}$. Then any $\left(\frac{1}{2}-\varepsilon\right)$-unbalanced circuit computing f has size at least $4^{-d}n^{\varepsilon d}$.*

This result improves over a previously known exponential lower bound on $\left(\frac{1}{5}\right)$-unbalanced circuits [17].

Slightly balanced circuits

A circuit $\mathcal{C}$ computing a homogeneous non-commutative polynomial of degree d is said to be ***α-balanced*** if every multiplication gate which computes a polynomial of degree k has both of its children which compute a polynomial of degree at least αk.

▶ **Theorem 19.** *Let f be a homogeneous non-commutative polynomial such that $M_{[1,d/2]}(f)$ has full rank $n^{d/2}$. Then any ε-balanced circuit computing f has size at least $4^{-d}n^{\varepsilon d}$.*

Small Non-skew Depth Circuits

A circuit $\mathcal{C}$ has ***non-skew depth k*** if all its parse trees are such that each path from the root to a leaf goes through at most k non-skew nodes, i.e., nodes for which the two children are inner nodes.

We obtain an alternative proof of the exponential lower bound of [17] on non-skew depth k circuits as an application of Theorem 12. In the statement below A refers to an explicit subset of $[d]$ that we do not define here.

▶ **Theorem 20.** *Let f be a homogeneous non-commutative polynomial of degree $d = 12kp$ such that $M_A(f)$ has full rank $n^{d/2}$. Then any circuit of non-skew depth k computing f has size at least $4^{-d}n^{p/3} = 4^{-d}n^{d/36k}$.*

4.2 Circuits with Many Parse Trees

We focus on ***k-PT circuits*** which are circuits with at most k different parse trees.

The Non-commutative Setting

Lagarde, Limaye, and Srinivasan [15] obtained a superpolynomial lower bound for superpolynomial k (up to $k = 2^{d^{\frac{1}{3}-\varepsilon}}$). We first show how to obtain the same result using Theorem 12.

For $s \in \mathrm{Tree}_d$ and $A \subseteq [d]$, we define $\mathrm{dist}(A, s) = \min\{\mathrm{dist}(A, I) \mid I \text{ spanned by } s\}$. The following lemma is a subtle probabilistic analysis ensuring the existence of a subset which is close enough to all k parse trees.

▶ **Lemma 21** (adapted from Claim 15 in [15])**.** *Let $s \in Tree_d$ be a shape with d leaves, and $\delta \leq \sqrt{d}$. Then*

$$\Pr_{A\sim\mathcal{U}\left(\binom{[d]}{d/2}\right)}\left[dist(A, s) > d/2 - \delta\right] \leq 2^{-\alpha d/\delta^2},$$

where α is some positive constant and $\mathcal{U}\left(\binom{[d]}{d/2}\right)$ the uniform distribution of subsets of d of size $d/2$.

Proof sketch. Following [15], we find a sequence of $r = \Omega(d/\delta^2)$ nodes of s which all span distant enough subtrees. We then obtain the bound by splitting the previous event into r essentially independent events. ◀

From there, the lower bound is obtained using Theorem 12 and a fine tuning of the parameters.

▶ **Theorem 22.** *Let f be a homogeneous non-commutative polynomial such that for all $A \subseteq [d]$ $M_A(f)$ has full rank. Let $k = 2^{d^{1/3-\varepsilon}}$ and $\varepsilon > 0$. Then for large enough d, any k-PT circuit computing f has size at least $2^{d^{1/3}(\log n - d^{-\varepsilon})}$.*

Proof. Let $\mathcal{C}$ be a k-PT circuit computing f, and $\delta = d^{1/3} \leq \sqrt{d}$. We first show that there exists a subset $A \subseteq [d]$ which is close to all parse trees in $\mathcal{C}$. Indeed, a union bound and Lemma 21 yield

$$\begin{aligned}\Pr_{A\sim\mathcal{U}\left(\binom{[d]}{d/2}\right)}\left[\exists s \in \mathrm{PT}(\mathcal{C}), \mathrm{dist}(A, s) > d/2 - \delta\right] &\leq \sum_{s\in\mathrm{PT}(\mathcal{C})} \Pr_{A\sim\mathcal{U}\left(\binom{[d]}{d/2}\right)}\left[\mathrm{dist}(A, s) > d/2 - \delta\right] \\ &\leq k2^{-\alpha d/\delta^2} = 2^{d^{1/3-\varepsilon} - \alpha d^{1/3}} < 1,\end{aligned}$$

for large enough d. We now pick a subset $A \subseteq [d]$ of size $d/2$ such that for all $s \in \mathrm{PT}(\mathcal{C}), \mathrm{dist}(A, s) \leq d/2 - \delta$, that is, any $s \in \mathrm{PT}(\mathcal{C})$ spans an interval $I(s)$ at distance at most $d/2 - \delta$ from A. Finally, we apply Theorem 12 to obtain

$$|\mathcal{C}| \geq \mathrm{rank}(M_A(f))\, n^{-(d/2-\delta)} k^{-1} = n^{d/2} n^{-(d/2-d^{1/3})} 2^{-d^{1/3-\varepsilon}} = 2^{d^{1/3}(\log n - d^{-\varepsilon})}.$$ ◀

We may improve the previous bound by applying Theorem 13 instead of Theorem 12. Indeed, since Theorem 13 gets rid of the factor k^{-1} in the lower bound, picking a much smaller δ ($\delta = d^{\varepsilon/3}$ instead of $d^{1/3}$) still leads to a superpolynomial lower bound, while allowing for more parse trees.

▶ **Theorem 23.** *Let f be a homogeneous non-commutative polynomial such that for all $A \subseteq [d]$ $M_A(f)$ has full rank. Let $k = 2^{d^{1-\varepsilon}}$ and $\varepsilon > 0$. Then for large enough d, any k-PT circuit computing f has size at least $n^{d^{\varepsilon/4}} d^{-2}$.*

The bound $2^{d^{1-\varepsilon}}$ on the number of parse trees is to be compared to the total number of shapes of size d which is $\frac{1}{d}\binom{2(d-1)}{d-1} \sim \frac{4^d}{d^{3/2}\sqrt{\pi}} \leq 2^{2d}$. As explained in the introduction this means that we obtain superpolynomial lower bounds for any class of circuits which has a small defect in the exponent of the total number of parse trees.

The Commutative Setting

Arvind and Raja [3] showed a superpolynomial lower bound for sublinear k (up to $k = d^{1/2-\varepsilon}$). We improve this to superpolynomial k (up to $k = 2^{d^{1-\varepsilon}}$).

Indeed, in the commutative setting, Lemma 21 holds as such (with a shape being an element of $\mathcal{T}_d$, that is, a commutative parse tree of size d). However, the generic lower bound theorems, namely Theorem 15 and Theorem 16, are not exactly the same, so we obtain slightly different results. In particular, the two results we obtain are incomparable. Applying Theorem 15 leads to Theorem 24, whereas Theorem 16 leads to Theorem 25.

▶ **Theorem 24.** *Let f be a set-multilinear commutative polynomial such that for all $A \subseteq [d]$, the matrix $M_A(f)$ has full rank. Let $k = 2^{d^{1/3-\varepsilon}}$ and $\varepsilon > 0$. Then for large enough d, any k-PT circuit computing f has size at least $2^{d^{1/3}(\log n - d^{-\varepsilon})}$.*

▶ **Theorem 25.** *Let f be a set-multilinear commutative polynomial such that for all $A \subseteq [d]$, the matrix $M_A(f)$ has full rank. Let $k = 2^{d^{1-\varepsilon}}$ and $\varepsilon > 0$. Then for large enough d, any k-PT circuit computing f has size at least $n^{d^{\varepsilon/4}} 2^{-d}$.*

References

1. Manindra Agrawal, Rohit Gurjar, Arpita Korwar, and Nitin Saxena. Hitting-sets for ROABP and sum of set-multilinear circuits. *SIAM Journal on Computing*, 44(3):669–697, 2015.
2. Eric Allender, Jia Jiao, Meena Mahajan, and V. Vinay. Non-commutative arithmetic circuits: Depth reduction and size lower bounds. *Theoretical Computer Science*, 209(1-2):47–86, 1998.
3. Vikraman Arvind and S. Raja. Some lower bound results for set-multilinear arithmetic computations. *Chicago Journal of Theoretical Computer Science*, 2016, 2016.
4. Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22:317–330, 1983.
5. Symeon Bozapalidis and Olympia Louscou-Bozapalidou. The rank of a formal tree power series. *Theoretical Computer Science*, 27:211–215, 1983.
6. Marco L. Carmosino, Russell Impagliazzo, Shachar Lovett, and Ivan Mihajlin. Hardness amplification for non-commutative arithmetic circuits. In *Proceedings of the 33rd Computational Complexity Conference (CCC 2018)*, volume 102 of *LIPIcs*, pages 12:1–12:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

7 Zeev Dvir, Guillaume Malod, Sylvain Perifel, and Amir Yehudayoff. Separating multilinear branching programs and formulas. In *Proceedings of the 44th Symposium on Theory of Computing Conference (STOC 2012)*, pages 615–624. ACM, 2012.

8 Nathanaël Fijalkow, Guillaume Lagarde, and Pierre Ohlmann. Tight bounds using hankel matrix for arithmetic circuits with unique parse trees. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:38, 2018. URL: `https://eccc.weizmann.ac.il/report/2018/038`.

9 Michel Fliess. Matrices de Hankel. *Journal de Mathématiques Pures et Appliquées*, 53:197–222, 1974.

10 Michael A. Forbes, Ramprasad Saptharishi, and Amir Shpilka. Hitting sets for multilinear read-once algebraic branching programs, in any order. In *Proceedings of the 46th Symposium on Theory of Computing, (STOC 2014)*, pages 867–875. ACM, 2014.

11 Rohit Gurjar, Arpita Korwar, Nitin Saxena, and Thomas Thierauf. Deterministic identity testing for sum of read-once oblivious arithmetic branching programs. *Computational Complexity*, 26(4):835–880, 2017.

12 Pavel Hrubeš, Avi Wigderson, and Amir Yehudayoff. Non-commutative circuits and the sum-of-squares problem. *Journal of the American Mathematical Society*, 24(3):871–898, 2011.

13 Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC 2003)*, pages 355–364. ACM, 2003.

14 Neeraj Kayal, Chandan Saha, and Ramprasad Saptharishi. A super-polynomial lower bound for regular arithmetic formulas. In *Proceedings of the 46th Symposium on Theory of Computing, (STOC 2014)*, pages 146–153. ACM, 2014.

15 Guillaume Lagarde, Nutan Limaye, and Srikanth Srinivasan. Lower bounds and PIT for non-commutative arithmetic circuits with restricted parse trees. *Computational Complexity*, pages 1–72, 2018. https://doi.org/10.1007/s00037-018-0171-9.

16 Guillaume Lagarde, Guillaume Malod, and Sylvain Perifel. Non-commutative computations: lower bounds and polynomial identity testing. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:94, 2016.

17 Nutan Limaye, Guillaume Malod, and Srikanth Srinivasan. Lower bounds for non-commutative skew circuits. *Theory of Computing*, 12(1):1–38, 2016.

18 Guillaume Malod and Natacha Portier. Characterizing Valiant's algebraic complexity classes. *Journal of Complexity*, 24(1):16–38, 2008.

19 Noam Nisan. Lower bounds for non-commutative computation (extended abstract). In *Proceedings of the 23rd Symposium on Theory of Computing (STOC 1991)*, pages 410–418. ACM, 1991.

20 Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.

21 C. Ramya and B. V. Raghavendra Rao. Lower bounds for special cases of syntactic multilinear abps. In *Proceedings of the 24th International Computing and Combinatorics Conference (COCOON 2018)*, volume 10976 of *Lecture Notes in Computer Science*, pages 701–712. Springer, 2018.

22 Ran Raz and Amir Shpilka. Deterministic polynomial identity testing in non-commutative models. *Computational Complexity*, 14(1):1–19, 2005.

23 Ramprasad Saptharishi and Anamay Tengse. Quasi-polynomial hitting sets for circuits with restricted parse trees. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:135, 2017.

24 Seinosuke Toda. Classes of arithmetic circuits capturing the complexity of computing the determinant. *IEICE Trans. Inf. Systems*, E75-D(1):116–124, 1992.

25 Leslie G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

Solving Vertex Cover in Polynomial Time on Hyperbolic Random Graphs

Thomas Bläsius
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany
thomas.blaesius@hpi.de

Philipp Fischbeck
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany
philipp.fischbeck@hpi.de

Tobias Friedrich
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany
tobias.friedrich@hpi.de

Maximilian Katzmann
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany
maximilian.katzmann@hpi.de

Abstract

The VertexCover problem is proven to be computationally hard in different ways: It is NP-complete to find an optimal solution and even NP-hard to find an approximation with reasonable factors. In contrast, recent experiments suggest that on many real-world networks the run time to solve VertexCover is way smaller than even the best known FPT-approaches can explain. Similarly, greedy algorithms deliver very good approximations to the optimal solution in practice.

We link these observations to two properties that are observed in many real-world networks, namely a heterogeneous degree distribution and high clustering. To formalize these properties and explain the observed behavior, we analyze how a branch-and-reduce algorithm performs on hyperbolic random graphs, which have become increasingly popular for modeling real-world networks. In fact, we are able to show that the VertexCover problem on hyperbolic random graphs can be solved in polynomial time, with high probability.

The proof relies on interesting structural properties of hyperbolic random graphs. Since these predictions of the model are interesting in their own right, we conducted experiments on real-world networks showing that these properties are also observed in practice. When utilizing the same structural properties in an adaptive greedy algorithm, further experiments suggest that, on real instances, this leads to better approximations than the standard greedy approach within reasonable time.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Random network models; Mathematics of computing → Random graphs

Keywords and phrases vertex cover, random graphs, hyperbolic geometry, efficient algorithm

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.25

Related Version A full version of the paper is available at `https://arxiv.org/abs/1904.12503`.

Funding This research was partially funded by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) – project number 390859508.

1 Introduction

VertexCover is a fundamental NP-complete graph problem. For a given undirected graph G on n vertices the goal is to find the smallest vertex subset S, such that each edge in G is incident to at least one vertex in S. Since, by definition, there can be no edge between two vertices outside of S, these remaining vertices form an independent set. Therefore, one can easily derive a maximal independent set from a minimal vertex cover and vice versa.

Due to its NP-completeness there is probably no polynomial time algorithm for solving VertexCover. The best known algorithm for IndependentSet runs in $1.1996^n \operatorname{poly}(n)$ [22]. To analyze the complexity of VertexCover on a finer scale, several parameterized solutions have been proposed. One can determine whether a graph G has a vertex cover of size k by applying a *branch-and-reduce* algorithm. The idea is to build a search tree by recursively considering two possible extensions of the current vertex cover (*branching*), until a vertex cover is found or the size of the current cover exceeds k. Each branching step is followed by a *reduce* step in which *reduction rules* are applied to make the considered graph smaller. This branch-and-reduce technique yields a simple $\mathcal{O}(2^k \operatorname{poly}(n))$ algorithm, where the exponential portion comes from the branching. The best known FPT (fixed-parameter tractable) algorithm runs in $\mathcal{O}(1.2738^k + kn)$ time [7], and unless ETH (exponential time hypothesis) fails, there can be no $2^{o(k)} \operatorname{poly}(n)$ algorithm [6].

While these FPT approaches promise relatively small running times if the considered network has a small vertex cover, the cover is large for many real-world networks. Nevertheless, it was recently observed that applying a branch-and-reduce technique on real instances is very efficient [1]. Some of the considered networks had millions of vertices, yet an optimal solution (also containing millions of vertices) was computed within seconds. Most instances were solved so quickly since the expensive branching was not necessary at all. In fact, the application of the reduction rules alone already yielded an optimal solution. Most notably, applying the *dominance reduction rule*, which eliminates vertices whose neighborhood contains a vertex together with its neighborhood, reduces the graph to a very small remainder on which the branching, if necessary, can be done quickly. We trace the effectiveness of the dominance rule back to two properties that are often observed in real-world networks: a *heterogeneous degree distribution* (the network contains many vertices of small degree and few vertices of high degree) and *high clustering* (the neighbors of a vertex are likely to be neighbors themselves).

We formalize these key properties using *hyperbolic random graphs* to analyze the performance of the dominance rule. Introduced by Krioukov et al. [17], hyperbolic random graphs are obtained by randomly distributing nodes in the hyperbolic plane and connecting any two that are geometrically close. The resulting graphs feature a power-law degree distribution and high clustering [14, 17] (the two desired properties) which can be tuned using parameters of the model. Additionally, the generated networks have a small diameter [13]. All of these properties have been observed in many real-world networks such as the internet, social networks, as well as biological networks like protein-protein interaction networks. Furthermore, Boguná, Papadopoulos, and Krioukov showed that the internet can be embedded into the hyperbolic plane such that routing packages between network participants greedily works very well [5], indicating that this network naturally fits into the hyperbolic space.

By making use of the underlying geometry, we show that VertexCover can be solved in polynomial time on hyperbolic random graphs, with high probability. This is done by showing that even a single application of the dominance reduction rule reduces a hyperbolic random graph to a remainder with small pathwidth on which VertexCover can then be solved efficiently. Our analysis provides an explanation for why VertexCover can be

solved efficiently on practical instances. We note that, while our analysis makes use of the underlying hyperbolic geometry, the algorithm itself is oblivious to it. Besides the running time the model predicts certain structural properties that also point us to an adapted greedy algorithm that is still very efficient and achieves better approximation ratios. We conducted experiments indicating that these predictions (concerning the structural properties and improved approximation) actually match the real world for a significant fraction of networks.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph. We denote the number of vertices in G with n. The *neighborhood* of a vertex v is defined as $N(v) = \{w \in V \mid \{v, w\} \in E\}$ and the size of the neighborhood, called the *degree* of v, is denoted by $\deg(v)$. For a subset $S \subseteq V$, we use $G[S]$ to denote the induced subgraph of G obtained by removing all vertices in $V \setminus S$. Furthermore, we use the shorthand notation $G_{\leq d}$ to denote $G[\{v \in V \mid \deg(v) \leq d\}]$.

The Hyperbolic Plane. After choosing a designated origin O in the two-dimensional hyperbolic plane, together with a reference ray starting at O, a point p is uniquely identified by its *radius* $r(p)$, denoting the hyperbolic distance to O, and its *angle* (or *angular coordinate*) $\varphi(p)$, denoting the angular distance between the reference ray and the line through p and O. The hyperbolic distance between two points p and q is given by

$$\operatorname{dist}(p, q) = \operatorname{acosh}(\cosh(r(p))\cosh(r(q)) - \sinh(r(p))\sinh(r(q))\cos(\Delta_\varphi(\varphi(p), \varphi(q)))),$$

where $\cosh(x) = (e^x + e^{-x})/2$, $\sinh(x) = (e^x - e^{-x})/2$ (both growing as $e^x/2 \pm o(1)$), and $\Delta_\varphi(p, q) = \pi - |\pi - |\varphi(p) - \varphi(q)||$ denotes the angular distance between p and q. If not stated otherwise, we assume that computations on angles are performed modulo 2π.

We use $B_p(r)$ to denote a disk of radius r centered at p, i.e., the set of points with hyperbolic distance at most r to p. Such a disk has an area of $2\pi(\cosh(r)-1)$ and circumference $2\pi\sinh(r)$. Thus, the area and the circumference of a disk in the hyperbolic plane grow exponentially with its radius. In contrast, this growth is polynomial in Euclidean space. Therefore, representing hyperbolic shapes in the Euclidean geometry results in a distortion. In the *native representation*, used in our figures, circles can appear teardrop-shaped (see Figure 2).

Hyperbolic Random Graphs. Hyperbolic random graphs are obtained by distributing n points uniformly at random within the disk $B_O(R)$ and connecting any two of them if and only if their hyperbolic distance is at most R; see Figure 1. The disk radius R (which matches the connection threshold) is defined as $R = 2\log(8n/(\pi\bar{\kappa}))$, where $\bar{\kappa}$ is a constant describing the desired average degree of the generated network. The coordinates for the vertices are drawn as follows. For vertex v the angular coordinate, denoted by $\varphi(v)$, is drawn uniformly at random from $[0, 2\pi]$ and the radius of v, denoted by $r(v)$, is sampled according to the probability density function $\alpha\sinh(\alpha r)/(\cosh(\alpha R) - 1)$ for $r \in [0, R]$ and $\alpha \in (1/2, 1)$. Thus,

$$f(r) = \frac{1}{2\pi}\frac{\alpha\sinh(\alpha r)}{\cosh(\alpha R) - 1} = \frac{\alpha}{2\pi}e^{-\alpha(R-r)}(1 + \Theta(e^{-\alpha R} - e^{-2\alpha r})), \tag{1}$$

is their joint distribution function for $r \in [0, R]$. For $r > R$, $f(r) = 0$. The constant $\alpha \in (1/2, 1)$ is used to tune the power-law exponent $\beta = 2\alpha + 1$ of the degree distribution of the generated network. Note that we obtain power-law exponents $\beta \in (2, 3)$. Exponents outside of this range are atypical for hyperbolic random graphs. On the one hand, for

$\beta < 2$ the average degree of the generated networks is divergent. On the other hand, for $\beta > 3$ hyperbolic random graphs degenerate: They decompose into smaller components, none having a size linear in n. The obtained graphs have logarithmic tree width [4], meaning the VertexCover problem can be solved efficiently in that case.

The probability for a given vertex to lie in a certain area A of the disk is given by its probability measure $\mu(A) = \int_A f(r)\mathrm{d}r$. The hyperbolic distance between two vertices u and v increases with increasing angular distance between them. The maximum angular distance such that they are still connected by an edge is bounded by [14, Lemma 6]

$$\begin{aligned}\theta(r(u), r(v)) &= \arccos\left(\frac{\cosh(r(u))\cosh(r(v)) - \cosh(R)}{\sinh(r(u))\sinh(r(v))}\right) \\ &= 2e^{(R-r(u)-r(v))/2}(1 + \Theta(e^{R-r(u)-r(v)})). \end{aligned} \tag{2}$$

Interval Graphs and Circular Arc Graphs. In an interval graph each vertex v is identified with an interval on the real line and two vertices are adjacent if and only if their intervals intersect. The *interval width* of an interval graph G, denoted by $\mathrm{iw}(G)$, is its maximum clique size, i.e., the maximum number of intervals that intersect in one point. For any graph the interval width is defined as the minimum interval width over all of its interval supergraphs. Circular arc graphs are a superclass of interval graphs, where each vertex is identified with a subinterval of the circle called *circular arc* or simply *arc*. The interval width of a circular arc graph G is at most twice the size of its maximum clique, since one obtains an interval supergraph of G by mapping the circular arcs into the interval $[0, 2\pi]$ on the real line and replacing all intervals that were split by this mapping with the whole interval $[0, 2\pi]$. Consequently, for any graph G, if k denotes the minimum over the maximum clique number of all circular arc supergraphs G' of G, then the interval width of G is at most $2k$.

Treewidth and Pathwidth. A *tree decomposition* of a graph G is a tree T where each tree node represents a subset of the vertices of G called *bag*, and the following requirements have to be satisfied: Each vertex in G is contained in at least one bag, all bags containing a given vertex in G form a connected subtree of T, and for each edge in G, there exists a bag containing both endpoints. The *width* of a tree decomposition is the size of its largest bag minus one. The *treewidth* of G is the minimum width over all tree decompositions of G. The *path decomposition* of a graph is defined analogously to the tree decomposition, with the constraint that the tree has to be a path. Additionally, as for the treewidth, the *pathwidth* of a graph G, denoted by $\mathrm{pw}(G)$, is the minimum width over all path decompositions of G. Clearly the pathwidth is an upper bound on the treewidth. It is known that for any graph G and any $k \geq 0$, the interval width of G is at most $k + 1$ if and only if its pathwidth is at most k [8, Theorem 7.14]. Consequently, if k' is the maximum clique size of a circular arc supergraph of G, then $2k' - 1$ is an upper bound on the pathwidth of G.

Probabilities. Since we are analyzing a random graph model, our results are of probabilistic nature. To obtain meaningful statements, we show that they hold *with high probability* (for short *whp.*), i.e., with probability $1 - \mathcal{O}(n^{-1})$. The following Chernoff bound is a useful tool for showing that certain events occur with high probability.

▶ **Theorem 1** (Chernoff Bound [11, A.1]). *Let $X_1, \ldots, X_n$ be independent random variables with $X_i \in \{0, 1\}$ and let X be their sum. Let $f(n) = \Omega(\log(n))$. If $f(n)$ is an upper bound for $\mathbb{E}[X]$, then for each constant c there exists a constant c' such that $X \leq c' f(n)$ holds with probability $1 - \mathcal{O}(n^{-c})$.*

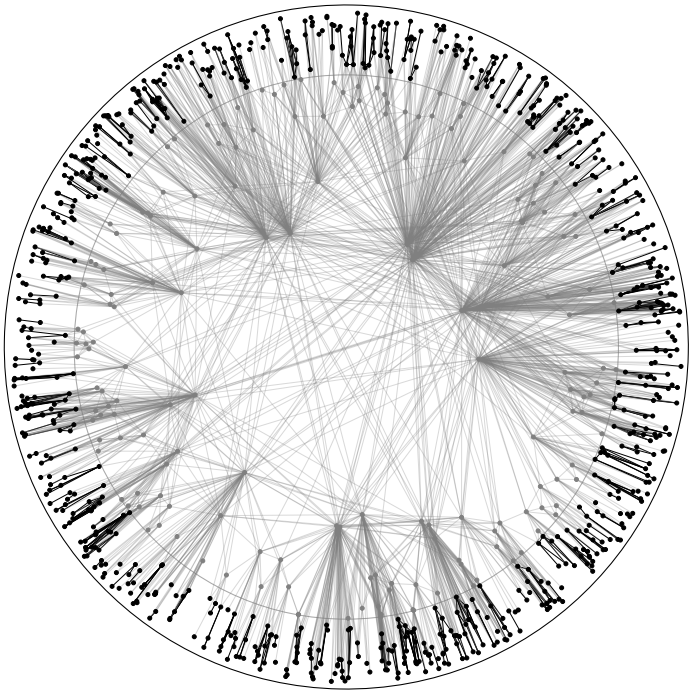


Figure 1 A hyperbolic random graph with 979 nodes, average degree 8.3, and a power-law exponent of 2.5. With high probability, the gray vertices and edges are removed by the dominance reduction rule. Additionally, the remaining subgraph in the outer band (consisting of the black vertices and edges) has a small path width, with high probability.

3 Vertex Cover on Hyperbolic Random Graphs

Reduction rules are often applied as a preprocessing step, before using a brute force search or branching in a search tree. They simplify the input by removing parts that are easy to solve. For example, an isolated vertex does not cover any edges and can thus never be part of a minimum vertex cover. Consequently, in a preprocessing step all isolated vertices can be removed, which leads to a reduced input size without impeding the search for a minimum.

The dominance reduction rule was previously defined for the IndependentSet problem [12], and later used for VertexCover in the experiments by Akiba and Iwata [1]. Formally, vertex u *dominates* a neighbor $v \in N(u)$ if $(N(v) \setminus \{u\}) \subseteq N(u)$, i.e., all neighbors of v are also neighbors of u. We say u is *dominant* if it dominates at least one vertex. The dominance rule states that u can be added to the vertex cover (and afterwards removed from the graph), without impeding the search for a minimum vertex cover. To see that this is correct, assume that u dominates v and let S be a minimum vertex cover that does not contain u. Since S has to cover all edges, it contains all neighbors of u. These neighbors include v and all of v's neighbors, since u dominates v. Therefore, removing v from S leaves only the edge $\{u, v\}$ uncovered which can be fixed by adding u instead. The resulting vertex cover has the same size as S. When searching for a minimum vertex cover of G, it is thus safe to assume that u is part of the solution and to reduce the search to $G[V \setminus \{u\}]$.

In the remainder of this section, we study the effectiveness of the dominance reduction rule on hyperbolic random graphs and conclude that VertexCover can be solved efficiently on these graphs. Our results are summarized in the following main theorem.

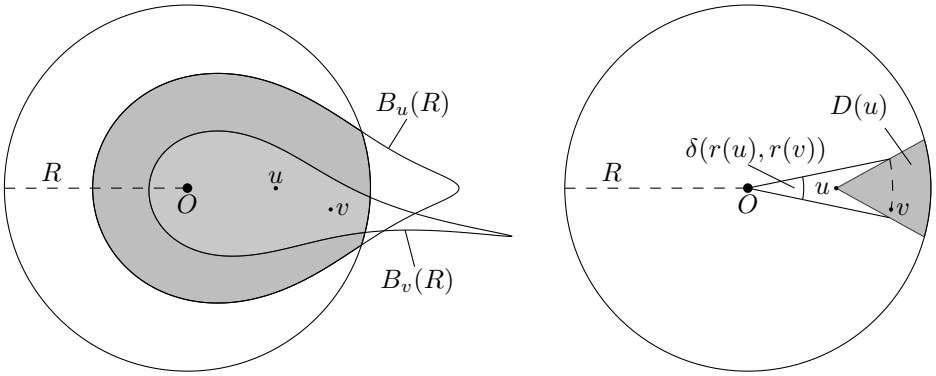


Figure 2 Left: Vertex u dominates vertex v, as $B_v(R) \cap B_O(R)$ (light gray) is completely contained in $B_u(R) \cap B_O(R)$ (gray). Right: All vertices that lie in $D(u)$ are dominated by u.

▶ **Theorem 2.** *Let G be a hyperbolic random graph on n vertices. Then the* VertexCover *problem on G can be solved in* $\text{poly}(n)$ *time, with high probability.*

The proof of Theorem 2 consists of two parts that make use of the underlying hyperbolic geometry. In the first part, we show that applying the dominance reduction rule once removes all vertices in the inner part of the hyperbolic disk with high probability, as depicted in Figure 1. We note that this is independent of the order in which the reduction rule is applied, as dominant vertices remain dominant after removing other dominant vertices. In the second part, we consider the induced subgraph containing the remaining vertices near the boundary of the disk (black vertices in Figure 1). We prove that this subgraph has a small pathwidth, by showing that there is a circular arc supergraph with a small interval width. Consequently, a tree decomposition of this subgraph can be computed efficiently. Finally, we obtain a polynomial time algorithm for VertexCover by first applying the reduction rules and afterwards solving VertexCover on the remaining subgraph using dynamic programming on the tree decomposition of small width.

3.1 Dominance on Hyperbolic Random Graphs

Recall that a hyperbolic random graph is obtained by distributing n vertices in a hyperbolic disk $B_O(R)$ and that any two are connected if their distance is at most R. Consequently, one can imagine the neighborhood of a vertex u as another disk $B_u(R)$. Vertex u dominates another vertex v if its neighborhood disk completely contains that of v (both constrained to $B_O(R)$), as depicted in Figure 2 left. We define the *dominance area* $D(u)$ of u to be the area containing all such vertices v. That is, $D(u) = \{p \in B_O(R) \mid B_p(R) \cap B_O(R) \subseteq B_u(R) \cap B_O(R)\}$. The result is illustrated in Figure 2 right. We note that it is sufficient for a vertex v to lie in $D(u)$ in order to be dominated by u, however, it is not necessary.

Given the radius $r(u)$ of vertex u we can now compute a lower bound on the probability that u dominates another vertex, i.e., the probability that at least one vertex lies in $D(u)$, by determining the measure $\mu(D(u))$. To this end, we first define $\delta(r(u), r(v))$ to be the maximum angular distance between two nodes u and v such that v lies in $D(u)$.

▶ **Lemma 3.** *Let u, v be vertices with $r(u) \leq r(v)$. Then, $v \in D(u)$ if $\Delta_\varphi(u, v)$ is at most*

$$\delta(r(u), r(v)) = 2(e^{-r(u)/2} - e^{-r(v)/2}) + \Theta(e^{-3/2r(u)}) - \Theta(e^{-3/2r(v)}).$$

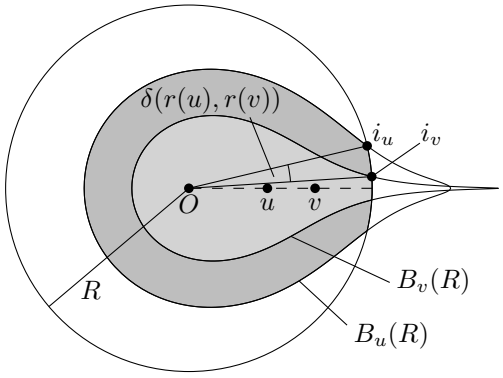


Figure 3 Vertex u dominates vertex v, with $r(u) \leq r(v)$, if $\Delta_\varphi(u, v) \leq \Delta_\varphi(i_u, i_v)$.

Proof. Without loss of generality we assume that $\varphi(u) = 0$. For now assume that $\varphi(v) = \varphi(u)$. Since $r(v) \geq r(u)$ we know that the intersections of the boundaries of $B_v(R)$ with $B_O(R)$ lie between those of $B_u(R)$ with $B_O(R)$, as is depicted in Figure 3. Now let i_u denote one of these intersections for $B_u(R)$ and $B_O(R)$, and let i_v denote the intersection for $B_v(R)$ and $B_O(R)$ that is on the same side of the ray through O and u as i_u. It is easy to see that the maximum angular distance between u and v such that $B_v(R) \cap B_O(R)$ is contained within $B_u(R) \cap B_O(R)$ is given by the angular distance between i_u and i_v. Therefore, v lies in the dominance area of u if $\Delta_\varphi(u, v) \leq \Delta_\varphi(i_u, i_v)$.

Recall that $\theta(r(p), r(q))$ denotes the maximum angular distance such that $\text{dist}(p, q) \leq R$, as defined in Equation (2). Since i_u and i_v have radius R and hyperbolic distance R to u and v, respectively, we know that their angular coordinates are $\theta(r(u), R)$ and $\theta(r(v), R)$, respectively. Consequently, the angular distance between i_u and i_v is given by

$$\begin{aligned}\delta(r(u), r(v)) &= \theta(r(u), R) - \theta(r(v), R) \\ &= 2(e^{-r(u)/2} - e^{-r(v)/2}) + \Theta(e^{-3/2r(u)}) - \Theta(e^{-3/2r(v)}).\end{aligned}$$ ◀

Using Lemma 3 we can now compute the probability for a given vertex to lie in the dominance area of u. We note that this probability grows roughly like $2/\pi e^{-r(u)/2}$, which is a constant fraction of the measure of the neighborhood disk of u which grows as $\alpha/(\alpha - 1/2) \cdot 2/\pi e^{-r(u)/2}$ [14, Lemma 3.2]. Consequently, the expected number of nodes that u dominates is a constant fraction of the expected number of its neighbors.

▶ **Lemma 4.** *Let u be a node with radius $r(u) \geq R/2$. The probability for a given node to lie in $D(u)$ is given by*

$$\mu(D(u)) = \frac{2}{\pi} e^{-r(u)/2}(1 - \Theta(e^{-\alpha(R - r(u))})) \pm \mathcal{O}(1/n).$$

Proof. The probability for a given vertex v to lie in $D(u)$ is obtained by integrating the probability density (given by Equation (1)) over $D(u)$.

$$\begin{aligned}\mu(D(u)) &= 2 \int_{r(u)}^{R} \int_{0}^{\delta(r(u), r)} f(r) \, \mathrm{d}\varphi \, \mathrm{d}r \\ &= 2 \int_{r(u)}^{R} \Big(2(e^{-r(u)/2} - e^{-r/2}) + \Theta(e^{-3/2r(u)}) - \Theta(e^{-3/2r})\Big) \\ &\qquad \cdot \frac{\alpha}{2\pi} e^{-\alpha(R-r)}(1 + \Theta(e^{-\alpha R} - e^{-2\alpha r})) \, \mathrm{d}r\end{aligned}$$

Since $r(u) \geq R/2$ and $r \in [r(u), R]$ we have $\Theta(e^{-3/2r(u)}) - \Theta(e^{-3/2r}) = \pm\mathcal{O}(e^{-3/4R})$ and $(1 + \Theta(e^{-\alpha R} - e^{-2\alpha r})) = (1 + \Theta(e^{-\alpha R}))$. Due to the linearity of integration, constant factors within the integrand can be moved out of the integral, which yields

$$\begin{aligned}\mu(D(u)) &= \frac{\alpha}{\pi} e^{-\alpha R}(1 + \Theta(e^{-\alpha R})) \int_{r(u)}^{R} \left(2(e^{-r(u)/2} - e^{-r/2}) \pm \mathcal{O}(e^{-3/4R})\right) \cdot e^{\alpha r}\,\mathrm{d}r \\ &= \frac{2\alpha}{\pi} e^{-r(u)/2} e^{-\alpha R}(1 + \Theta(e^{-\alpha R})) \int_{r(u)}^{R} e^{\alpha r}\mathrm{d}r \\ &\quad - \frac{2\alpha}{\pi} e^{-\alpha R}(1 + \Theta(e^{-\alpha R})) \int_{r(u)}^{R} e^{(\alpha - 1/2)r}\mathrm{d}r \pm \mathcal{O}\left(e^{-(3/4+\alpha)R} \int_{r(u)}^{R} e^{\alpha r}\mathrm{d}r\right).\end{aligned}$$

The remaining integrals can be computed easily and we obtain

$$\begin{aligned}\mu(D(u)) &= \frac{2}{\pi} e^{-r(u)/2}(1 + \Theta(e^{-\alpha R}))(1 - e^{-\alpha(R - r(u))}) \\ &\quad - \frac{2\alpha}{(\alpha - 1/2)\pi} e^{-R/2}(1 + \Theta(e^{-\alpha R}))(1 - e^{-(\alpha - 1/2)(R - r(u))}) \\ &\quad \pm \mathcal{O}\left(e^{-3/4R}(1 - e^{-\alpha(R - r(u))})\right).\end{aligned}$$

As $e^{-R/2} = \Theta(n^{-1})$ and $e^{-3/4R} = \Theta(n^{-3/2})$, simplifying the error terms yields the claim. ◀

The following lemma shows that, with high probability, all vertices that are not too close to the boundary of the disk dominate at least one vertex.

▶ **Lemma 5.** *Let G be a hyperbolic random graph with average degree $\bar{\kappa}$. Then there is a constant $c > 4/\bar{\kappa}$, such that all vertices u with $r(u) \leq \rho = R - 2\log\log(n^c)$ are dominant, with high probability.*

Proof. Vertex u is dominant if at least one vertex lies in $D(u)$. To show this for any u with $r(u) \leq \rho$, it suffices to show it for $r(u) = \rho$, since $D(u)$ increases with decreasing radius. To determine the probability that at least one vertex lies in $D(u)$, we use Lemma 4 and obtain

$$\begin{aligned}\mu(D(u)) &= \frac{2}{\pi} e^{-\rho/2}(1 - \Theta(e^{-\alpha(R-\rho)})) \pm \mathcal{O}(1/n) \\ &= \frac{2}{\pi} e^{-R/2 + \log\log(n^c)}(1 - \Theta(e^{-2\alpha\log\log(n^c)})) \pm \mathcal{O}(1/n).\end{aligned}$$

By substituting $R = 2\log(8n/(\pi\bar{\kappa}))$, we obtain $\mu(D(u)) = \bar{\kappa}/(4n)(c\log(n)(1 - o(1)) \pm \mathcal{O}(1))$. The probability of at least one node falling into $D(u)$ is now given by

$$\Pr[\{v \in D(u)\} \neq \emptyset] = 1 - (1 - \mu(D(u)))^n \geq 1 - e^{-n\mu(D(u))} = 1 - \Theta(n^{-c\bar{\kappa}/4(1-o(1))}).$$

Consequently, for large enough n we can choose $c > 4/\bar{\kappa}$ such that the probability of a vertex at radius ρ being dominant is at least $1 - \Theta(n^{-2})$, allowing us to apply union bound. ◀

▶ **Corollary 6.** *Let G be a hyperbolic random graph and $c > 4/\bar{\kappa}$. With high probability, all vertices with radius at most $\rho = R - 2\log\log(n^c)$ are removed by the dominance rule.*

By Corollary 6 the dominance rule removes all vertices of radius at most ρ. Consequently, all remaining vertices have radius at least ρ. We refer to this part of the disk as *outer band*. More precisely, the outer band is defined as $B_O(R) \setminus B_O(\rho)$. It remains to show that the pathwidth of the subgraph induced by the vertices in the outer band is small.

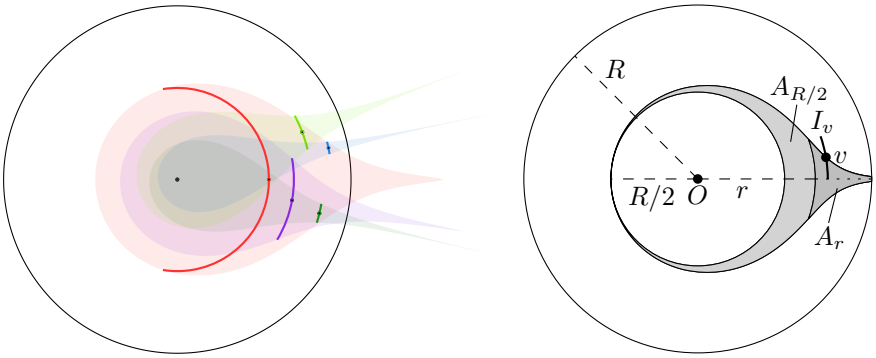


Figure 4 Left: The circular arcs representing the neighborhood of a vertex. For vertex v the area containing the whole neighborhood of v, as well as the circular arc I_v are drawn in the same color. Right: The area that contains the vertices whose arcs intersect angle 0. Area A_r contains all such vertices with radius at least r. Vertex v lies on the boundary of A_r and its interval I_v extends to 0.

3.2 Pathwidth in the Outer Band

In the following, we use $G_r = G[\{v \in V\} \mid r(v) \geq r]$ to denote the induced subgraph of G that contains all vertices with radius at least r. To show that the pathwidth of G_ρ (the induced subgraph in the outer band) is small, we first show that there is a circular arc supergraph G_ρ^S of G_ρ with a small maximum clique. We use G^S to denote a circular arc supergraph of a hyperbolic random graph G, which is obtained by assigning each vertex v an angular interval I_v on the circle, such that the intervals of two adjacent vertices intersect. More precisely, for a vertex v, we set $I_v = [\varphi(v) - \theta(r(v), r(v)), \varphi(v) + \theta(r(v), r(v))]$. Intuitively, this means that the interval of a vertex contains a superset of all its neighbors that have a larger radius, as can be seen in Figure 4 left. The following lemma shows that G^S is actually a supergraph of G.

▶ **Lemma 7.** *Let $G = (V, E)$ be a hyperbolic random graph. Then G^S is a supergraph of G.*

Proof. Let $\{u, v\} \in E$ be any edge in G. To show that G^S is a supergraph of G we need to show that u and v are also adjacent in G^S, i.e., $I_u \cap I_v \neq \emptyset$. Without loss of generality assume $r(u) \leq r(v)$. Since u and v are adjacent in G, the hyperbolic distance between them is at most R. It follows, that their angular distance $\Delta_\varphi(u, v)$ is bounded by $\theta(r(u), r(v))$. Since $\theta(r(u), r(v)) \leq \theta(r(u), r(u))$ for $r(u) \leq r(v)$, we have $\Delta_\varphi(u, v) \leq \theta(r(u), r(u))$. As I_u extends by $\theta(r(u), r(u))$ from $\varphi(u)$ in both directions, it follows that $\varphi(v) \in I_u$. ◀

It is easy to see that, after removing a vertex from G and G^S, G^S is still a supergraph of G. Consequently, G_ρ^S is a supergraph of G_ρ. It remains to show that G_ρ^S has a small maximum clique number, which is given by the maximum number of arcs that intersect at any angle. To this end, we first compute the number of arcs that intersect a given angle which we set to 0 without loss of generality. Let A_r denote the area of the disk containing all vertices v with radius $r(v) \geq r$ whose interval I_v intersects 0, as illustrated in Figure 4 right. The following lemma describes the probability for a given vertex to lie in A_r.

▶ **Lemma 8.** *Let G be a hyperbolic random graph and let $r \geq R/2$. The probability for a given vertex to lie in A_r is bounded by*

$$\mu(A_r) \leq \frac{2\alpha}{(1-\alpha)\pi} e^{-(\alpha-1/2)R-(1-\alpha)r} \cdot \left(1 + \Theta(e^{-\alpha R} + e^{-(2r-R)} - e^{-(1-\alpha)(R-r)})\right).$$

Proof. We obtain the measure of A_r by integrating the probability density function over A_r. Due to the definition of I_v we can conclude that A_r includes all vertices v with radius $r(v) \geq r$ whose angular distance to 0 is at most $\theta(r(v), r(v))$, defined in Equation (2). We obtain,

$$\begin{aligned}\mu(A_r) &= \int_r^R 2 \int_0^{\theta(x,x)} f(x) \,\mathrm{d}\varphi \,\mathrm{d}x \\ &= 2\int_r^R 2e^{(R-2x)/2}(1 \pm \Theta(e^{R-2x})) \cdot \frac{\alpha}{2\pi} e^{-\alpha(R-x)}(1 + \Theta(e^{-\alpha R} - e^{-2\alpha x})) \,\mathrm{d}x.\end{aligned}$$

As before, we can conclude that $(1 + \Theta(e^{-\alpha R} - e^{-2\alpha r})) = (1 + \Theta(e^{-\alpha R}))$, since $r \geq R/2$. By moving constant factors out of the integral, the expression can be simplified to

$$\mu(A_r) \leq \frac{2\alpha}{\pi} e^{-(\alpha-1/2)R}(1 + \Theta(e^{-\alpha R})) \int_r^R e^{-(1-\alpha)x}(1 + \Theta(e^{R-2x})) \,\mathrm{d}x.$$

We split the sum in the integral and deal with the two resulting integrals separately.

$$\begin{aligned}\mu(A_r) &\leq \frac{2\alpha}{\pi} e^{-(\alpha-1/2)R}(1 + \Theta(e^{-\alpha R})) \left(\int_r^R e^{-(1-\alpha)x} \,\mathrm{d}x + \Theta\left(\int_r^R e^{-(1-\alpha)x+R-2x} \,\mathrm{d}x \right) \right) \\ &= \frac{2\alpha}{\pi} e^{-(\alpha-1/2)R}(1 + \Theta(e^{-\alpha R})) \\ &\quad \cdot \left(\frac{1}{1-\alpha} e^{-(1-\alpha)r}(1 - e^{-(1-\alpha)(R-r)}) + \Theta\left(e^R e^{-(3-\alpha)r}(1 - e^{-(3-\alpha)(R-r)}) \right) \right).\end{aligned}$$

By placing $1/(1-\alpha)e^{-(1-\alpha)r}$ outside of the brackets we obtain

$$\begin{aligned}\mu(A_r) &\leq \frac{2\alpha}{(1-\alpha)\pi} e^{-(\alpha-1/2)R-(1-\alpha)r}(1 + \Theta(e^{-\alpha R})) \\ &\quad \cdot \left((1 - e^{-(1-\alpha)(R-r)}) + \Theta\left(e^{R-2r}(1 - e^{-(3-\alpha)(R-r)}) \right) \right).\end{aligned}$$

Simplifying the remaining error terms then yields the claim. ◀

We can now bound the maximum clique number in G_ρ^S and thus its interval width $\mathrm{iw}(G_\rho^S)$.

▶ **Theorem 9.** *Let G be a hyperbolic random graph and $r \geq R/2$. Then there exists a constant c such that, whp., $\mathrm{iw}(G_r^S) = \mathcal{O}(\log(n))$ if $r \geq R - \frac{1}{(1-\alpha)} \log\log(n^c)$, and otherwise*

$$\mathrm{iw}(G_r^S) \leq \frac{4\alpha}{(1-\alpha)\pi} n e^{-(\alpha-1/2)R-(1-\alpha)r} \left(1 + \Theta(e^{-\alpha R} + e^{-(2r-R)} - e^{-(1-\alpha)(R-r)})\right).$$

Proof. We start by determining the expected number of arcs that intersect at a given angle, which can be done by computing the expected number of vertices in A_r, using Lemma 8:

$$\mathbb{E}[|\{v \in A_r\}|] \leq \frac{2\alpha}{(1-\alpha)\pi} n e^{-(\alpha-1/2)R-(1-\alpha)r}(1 + \Theta(e^{-\alpha R} + e^{-(2r-R)} - e^{-(1-\alpha)(R-r)})).$$

It remains to show that this bound holds with high probability at every angle. To this end, we make use of a Chernoff bound (Theorem 1), by first showing that the bound on $\mathbb{E}[|\{v \in A_r\}|]$ is $\Omega(\log(n))$. We start with the case where $r < R - \frac{1}{1-\alpha} \log\log(n^c)$.

$$\begin{aligned}\mathbb{E}[|\{v \in A_r\}|] &< \frac{2\alpha}{(1-\alpha)\pi} n e^{-(\alpha-1/2)R-(1-\alpha)(R-1/(1-\alpha)\log\log(n^c))} \\ &\quad \cdot \Big(1 + \Theta(e^{-\alpha R} + e^{-(2(R-1/(1-\alpha)\log\log(n^c))-R)} \\ &\quad - e^{-(1-\alpha)(R-(R-1/(1-\alpha)\log\log(n^c)))})\Big) \\ &= \frac{2\alpha}{(1-\alpha)\pi} n e^{-R/2+\log\log(n^c))} \\ &\quad \cdot \Big(1 + \Theta(e^{-\alpha R} + e^{-(R-2/(1-\alpha)\log\log(n^c))} - e^{-\log\log(n^c)})\Big)\end{aligned}$$

Substituting $R = 2\log(8n/(\pi\bar{\kappa}))$ we obtain

$$\mathbb{E}[|\{v \in A_r\}|] < \frac{\alpha\bar{\kappa}c}{4(1-\alpha)}\log(n)(1+o(1)).$$

Thus, for all radii smaller than $R - \frac{1}{(1-\alpha)}\log\log(n^c)$, the resulting upper bound is lower bounded by $\Omega(\log(n))$, which lets us apply Theorem 1. Moreover, as $\mathbb{E}[|\{v \in A_r\}|]$ decreases with increasing r, $\mathcal{O}(\log(n))$ is a pessimistic but valid upper bound for the case $r \geq R - \frac{1}{(1-\alpha)}\log\log(n^c)$. Thus, we can also apply Theorem 1 to this case, using the $\mathcal{O}(\log(n))$ bound.

By Theorem 1, we can choose c such that in both cases the bound holds with probability $1 - \mathcal{O}(n^{-c'})$ for any c' at a given angle. In order to see that it holds at every angle, note that it suffices to show that it holds at all arc endings as the number of intersecting arcs does not change in between arc endings. Since there are exactly $2n$ arc endings, we can apply union bound and obtain that the bound holds with probability $1 - \mathcal{O}(n^{-c'+1})$ for any c' at every angle. Since our bound on $\mathbb{E}[|\{v \in A_r\}|]$ is an upper bound on the maximum clique size of G_r^S, the interval width of G_r^S is at most twice as large, as argued in Section 2. ◀

Since the interval width of a circular arc supergraph of G is an upper bound on the pathwidth of G [8, Theorem 7.14] and since $\rho \geq R - 1/(1-\alpha)\log\log(n^c)$ for $\alpha \in (1/2, 1)$, we immediately obtain the following corollary.

▶ **Corollary 10.** *Let G be a hyperbolic random graph and let G_ρ be the subgraph obtained by removing all vertices with radius at most $\rho = R - 2\log\log(n^c)$. Then,* $\mathrm{pw}(G_\rho) = \mathcal{O}(\log(n))$.

We are now ready to prove our main theorem, which we restate for the sake of readability.

▶ **Theorem 2.** *Let G be a hyperbolic random graph on n vertices. Then the* VertexCover *problem in G can be solved in* $\mathrm{poly}(n)$ *time, with high probability.*

Proof. Consider the following algorithm that finds the minimum vertex cover of G. We start with an empty vertex cover S. Initially, all dominant vertices are added to S, which is correct due to the dominance rule. By Lemma 5, this includes all vertices of radius at most $\rho = R - 2\log\log(n^c)$, for some constant c, with high probability. Obviously, finding all vertices that are dominant can be done in $\mathrm{poly}(n)$ time. It remains to determine a vertex cover of G_ρ. By Corollary 10, the pathwidth of G_ρ is $\mathcal{O}(\log(n))$, with high probability. Since the pathwidth is an upper bound on the treewidth, we can find a tree decomposition of G_ρ and solve the VertexCover problem in G_ρ in $\mathrm{poly}(n)$ time [8, Theorems 7.18 and 7.9]. ◀

Moreover, linking the radius of a vertex in Theorem 9 with its expected degree leads to the following corollary, which is interesting in its own right. It links the pathwidth to the degree d in the graph $G_{\leq d}$. Recall that $G_{\leq d}$ denotes the subgraph of G induced by the vertices of degree at most d.

▶ **Corollary 11.** *Let G be a hyperbolic random graph and let $d \leq \sqrt{n}$. Then, with high probability,* $\mathrm{pw}(G_{\leq d}) = \mathcal{O}(d^{2-2\alpha} + \log(n))$.

Proof. Consider the radius $r = R - 2\log(\varepsilon d)$ for some constant $\varepsilon > 0$, and the graph G_r which is obtained by removing all vertices of radius at most r. By substituting $R = 2\log(8n/(\pi\bar{\kappa}))$ and using [14, Lemma 3.2] we can compute the expected degree of a vertex with radius r as

$$\mathbb{E}[\deg(v) \mid r(v) = r] = \frac{2\alpha}{(\alpha - 1/2)\pi} n e^{-r/2}(1 \pm \mathcal{O}(e^{-(\alpha-1/2)r})) = \frac{\alpha\bar{\kappa}\varepsilon}{4(\alpha - 1/2)} d(1 \pm o(1)).$$

First assume that $d \geq \log(n)^{1/(2-2\alpha)}$. We handle the other case later. Since $d \in \Omega(\log(n))$ we can choose ε large enough to apply Theorem 1 and conclude that this holds with high probability. Furthermore, since a smaller radius implies a larger degree, we know that, with high probability, all nodes v with radius at most r, have

$$\deg(v) \geq \frac{\alpha\bar{\kappa}\varepsilon}{4(\alpha - 1/2)} d(1 \pm o(1)).$$

For large enough n we can choose ε such that, with high probability, G_r is a supergraph of $G_{\leq d}$. To prove the claim, it remains to bound the pathwidth of G_r. If $r > R - 1/(1-\alpha)\log\log(n^c)$, we can apply the first part of Theorem 9 to obtain $\mathrm{iw}(G_r^S) = \mathcal{O}(\log(n))$. Otherwise, we use part two to conclude that the interval width of G_r is at most

$$\begin{aligned}\mathrm{iw}(G_r^S) &\leq \frac{4\alpha}{(1-\alpha)\pi} n e^{-(\alpha-1/2)R-(1-\alpha)r}\left(1 + \Theta(e^{-\alpha R} + e^{-(2r-R)} - e^{-(1-\alpha)(R-r)})\right) \\ &= \frac{\alpha\bar{\kappa}\varepsilon^{2-2\alpha}}{(2-2\alpha)} d^{2-2\alpha}\left(1 + \Theta(n^{-2\alpha} + ((\varepsilon d)^2/n)^2 - (\varepsilon d)^{-(2-2\alpha)})\right) = \mathcal{O}(d^{2-2\alpha}).\end{aligned}$$

As argued in Section 2 the interval width of a graph is an upper bound on the pathwidth.

For $d < \log(n)^{1/(2-2\alpha)}$ (which we excluded above), consider $G_{\leq d'}$ for $d' = \log(n)^{1/(2-2\alpha)} > d$. As we already proved the corollary for d', we obtain $\mathrm{pw}(G_{\leq d'}) = \mathcal{O}(d'^{2-2\alpha} + \log(n)) = \mathcal{O}(\log(n))$. As $G_{\leq d}$ is a subgraph of $G_{\leq d'}$, the same bound holds for $G_{\leq d}$. ◀

4 Discussion

Our results show that a heterogeneous degree distribution as well as high clustering make the dominance rule very effective. This matches the behavior for real-world networks, which typically exhibit these two properties. However, our analysis actually makes more specific predictions: (I) vertices with sufficiently high degree usually have at least one neighbor they dominate and can thus safely be included in the vertex cover; and (II) the graph remaining after deleting the high degree vertices has simple structure, i.e., small pathwidth.

To see whether this matches the real world, we run experiments on 59 networks from several network datasets [2, 3, 18, 19, 20]. Although the focus of this paper is the theoretical analysis on hyperbolic random graphs, we briefly report on our experimental results. (Detailed results are in the full version of the paper.) Out of the 59 instances, we can solve VertexCover for 47 networks in reasonable time. We refer to these as *easy*, while the remaining 12 are called *hard*. Note that our theoretical analysis aims at explaining why the easy instances are easy.

Recall from Lemma 5 that all vertices with radius at most $R - 2\log\log(n^{4/\bar{\kappa}})$ probably dominate, which corresponds to an expected degree of $\alpha/(\alpha - 1/2) \cdot \log n$. For more than half of the 59 networks, more than 78 % of the vertices above this degree were in fact dominant. For more than a quarter of the networks, more than 96 % were dominant. Restricted to the 47 easy instances, these number increase to 82 % and 99 %, respectively.

Experiments concerning the pathwidth of the resulting graph are much more difficult, due to the lack of efficient tools. Therefore, we used the tool by Tamaki et al. [21] to heuristically compute upper bounds on the treewidth instead. As in our analysis, we only removed vertices that dominate in the original graph instead of applying the reduction rule exhaustively. On the resulting subgraphs, the treewidth heuristic ran with a 15 min timeout. The resulting treewidth is at most 50 for 44 % of the networks, at most 15 for 34 %, and at most 5 for 25 %. Restricted to easy instances, the values increase to 55 %, 43 %, and 32 %, respectively.

Hyperbolic random graphs are of course an idealized representation of real-world networks. However, these experiments indicate that the predictions derived from the model match the real world, at least for a significant fraction of networks.

Approximation. Concerning approximation algorithms for VertexCover, there is a similar theory-practice gap as for exact solutions. In theory, there is a simple 2-approximation and the best known polynomial time approximation reduces the factor to $2 - \Theta(\log(n)^{-1/2})$ [15]. However, it is NP-hard to approximate VertexCover within a factor of 1.3606 [10], and presumably it is even NP-hard to approximate within a factor of $2 - \varepsilon$ for all $\varepsilon > 0$ [16]. Moreover, the greedy strategy that iteratively adds the vertex with maximum degree to the vertex cover and deletes it, is only a $\log n$ approximation. However, on scale-free networks this strategy performs exceptionally well with approximation ratios very close to 1 [9].

Our results for hyperbolic random graphs at least partially explain this good approximation ratio. Lemma 5 states that, with high probability, we do not make any mistake by taking all vertices below a certain radius ρ, which corresponds to vertices of at least logarithmic degree. The same computation for larger values of ρ does no longer give such strong guarantees. However, it still gives bounds on the probability for making a mistake. In fact, this error probability is sub-constant as long as the corresponding expected degree is super-constant.

Although this is not a formal argument, it still explains to a degree why greedy works so well on networks with a heterogeneous degree distribution and high clustering. Moreover, it indicates how the greedy algorithm should be adapted to obtain better approximation ratios: As the probability to make a mistake grows with growing radius and thus with shrinking vertex degree, the majority of mistakes are done when all vertices have already low degree. However, for hyperbolic random graphs, the subgraphs induced by vertices below a certain constant degree decompose into small components for $n \to \infty$. It thus seems to be a good idea to run the greedy algorithm only until all remaining vertices have low degree, say k. The remaining small connected components of maximum-degree k can then be solved with brute force in reasonable time. In the following we call the resulting algorithm *k-adaptive greedy*.

We ran experiments on the 47 easy real networks mentioned above (for the hard instances, we cannot measure approximation ratios). For these networks, we compare the normal greedy algorithm with 2- and 4-adaptive greedy. Note that 2-adaptive greedy is special, as VertexCover can be solved efficiently on graphs with maximum degree 2 (no brute-forcing is necessary). For 4-adaptive greedy, the size of the largest connected component is relevant.

The median approximation ratio for greedy over all 47 networks is 1.008. This goes down to 1.005 for 2-adaptive and to 1.002 for 4-adaptive greedy. Thus, the number of too many selected vertices goes down by a factor of 1.6 and 4, respectively. As mentioned above, the size of the largest connected component is relevant for 4-adaptive greedy. For 49 % of the networks, this was below 100 (which is still a reasonable size for a brute-force algorithm). Restricted to these networks, normal greedy has a median approximation ratio of 1.004, while 4-adaptive again improves by a factor of 4 to 1.001. Moreover, the number of networks for which we actually obtain the optimal solution increases from 4 to 7.

References

1 Takuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/FPT algorithms in practice: A case study of vertex cover. *Theor. Comput. Sci.*, 609:211–225, 2016. doi:10.1016/j.tcs.2015.09.023.

2 Alexandre Arenas, Albert-László Barabási, Vladimir Batagelj, Andrej Mrvar, Mark Newman, and Tore Opsahl. Gephi datasets. https://github.com/gephi/gephi/wiki/Datasets.

3 Vladimir Batagelj and Andrej Mrvar. Pajek datasets. http://vlado.fmf.uni-lj.si/pub/networks/data/, 2006.

4 Thomas Bläsius, Tobias Friedrich, and Anton Krohmer. Hyperbolic Random Graphs: Separators and Treewidth. In *24th Annual European Symposium on Algorithms (ESA 2016)*, pages 15:1–15:16, 2016. doi:10.4230/LIPIcs.ESA.2016.15.

5 Marián Boguná, Fragkiskos Papadopoulos, and Dmitri Krioukov. Sustaining the internet with hyperbolic mapping. *Nat. Commun.*, 1:62, 2010. doi:10.1038/ncomms1063.

6 Liming Cai and David Juedes. On the existence of subexponential parameterized algorithms. *J. Comput. Syst. Sci.*, 67:789–807, 2003. doi:10.1016/S0022-0000(03)00074-6.

7 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40):3736–3756, 2010. doi:10.1016/j.tcs.2010.06.026.

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

9 Mariana O. Da Silva, Gustavo A. Gimenez-Lugo, and Murilo V. G. Da Silva. Vertex cover in complex networks. *Int. J. Mod. Phys. C*, 24(11):1350078, 2013. doi:10.1142/S0129183113500782.

10 Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Ann. Math.*, 162(1):439–485, 2005. doi:10.4007/annals.2005.162.439.

11 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2012.

12 Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5):25:1–25:32, 2009. doi:10.1145/1552285.1552286.

13 Tobias Friedrich and Anton Krohmer. On the diameter of hyperbolic random graphs. In *Automata, Languages, and Programming*, pages 614–625. Springer Berlin Heidelberg, 2015. doi:10.1007/978-3-662-47666-6_49.

14 Luca Gugelmann, Konstantinos Panagiotou, and Ueli Peter. Random hyperbolic graphs: Degree sequence and clustering. In *Automata, Languages, and Programming*, pages 573–585. Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-31585-5_51.

15 George Karakostas. A better approximation ratio for the vertex cover problem. *ACM Trans. Algorithms*, 5(4):41:1–41:8, 2009. doi:10.1145/1597036.1597045.

16 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2-\varepsilon$. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008. doi:10.1016/j.jcss.2007.06.019.

17 Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *Phys. Rev. E*, 82:036106, 2010. doi:10.1103/PhysRevE.82.036106.

18 Jérôme Kunegis. KONECT: The koblenz network collection. In *International Conference on World Wide Web (WWW)*, pages 1343–1350, 2013. doi:10.1145/2487788.2488173.

19 Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, 2014.

20 Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015. URL: http://networkrepository.com.

21 Hisao Tamaki, Hiromu Ohtsuka, Takuto Sato, and Keitaro Makii. TCS-Meiji PACE2017-TrackA. https://github.com/TCS-Meiji/PACE2017-TrackA, 2017.

22 Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. *Inf. Comput.*, 255:126–146, 2017. doi:10.1016/j.ic.2017.06.001.

Domino Problem Under Horizontal Constraints

Nathalie Aubrun
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1,
LIP, F-69342, LYON Cedex 07, France
https://perso.ens-lyon.fr/nathalie.aubrun/
nathalie.aubrun@ens-lyon.fr

Julien Esnay
Institut de Mathématiques de Toulouse, Université Paul Sabatier,
118 route de Narbonne, F-31062 Toulouse Cedex 9, France
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1,
LIP, F-69342, Lyon Cedex 07, France
julien.esnay@ens-lyon.fr

Mathieu Sablik
Institut de Mathématiques de Toulouse, Université Paul Sabatier,
118 route de Narbonne, F-31062 Toulouse Cedex 9, France
http://www.math.univ-toulouse.fr/~msablik/index.html
Mathieu.Sablik@math.univ-toulouse.fr

Abstract

The Domino Problem on $\mathbb{Z}^2$ asks if it is possible to tile the plane with a given set of Wang tiles; it is a classical decision problem which is known to be undecidable. The purpose of this article is to parameterize this problem to explore the frontier between decidability and undecidability. To do so we fix some horizontal constraints H on the tiles and consider a new Domino Problem DP_H: given a vertical constraint, is it possible to tile the plane? We characterize the nearest-neighbor horizontal constraints where DP_H is decidable using graphs combinatorics.

2012 ACM Subject Classification Theory of computation → Models of computation; Mathematics of computing → Combinatoric problems

Keywords and phrases Dynamical Systems, Symbolic Dynamics, Subshifts, Wang tiles, Undecidability, Domino Problem, Combinatorics, Tilings, Subshifts of Finite Type

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.26

Funding *Nathalie Aubrun*: Supported by the ANR project CoCoGro (ANR-16-CE40-0005).
Julien Esnay: Supported by the ANR project CoCoGro (ANR-16-CE40-0005) and the Labex CIMI (project "Computational Complexity of Dynamical Systems").
Mathieu Sablik: Supported by the Labex CIMI (project "Computational Complexity of Dynamical Systems").

1 Introduction

The Domino Problem is a classical decision problem introduced by Wang [20] to study satisfaction procedures for some fragments of first-order logic. Considering a finite set of tiles that are squares with colored edges, called Wang tiles, we ask if it is possible to tile the plane with shifted copies of these tiles so that contiguous edges have the same color. This question is also central in symbolic dynamics. A $\mathbb{Z}^d$-subshift of finite type is a set of colorings of $\mathbb{Z}^d$ by a finite alphabet, called configurations, and a finite set of patterns that are forbidden to appear in those configurations. The set of tilings obtained when we tile the plane with a Wang tile set is an example of $\mathbb{Z}^2$-subshift of finite type. In this setting, the Domino Problem becomes: given a finite set of forbidden patterns, is the associated subshift of finite type empty?

On $\mathbb{Z}$-subshifts of finite type, the Domino Problem is easily shown to be decidable. On those over $\mathbb{Z}^2$, Wang conjectured the Domino Problem was decidable too, and produced an algorithm of decision relying on the hypothetical fact that all subshifts of finite type contained some periodic configuration. However, his claim was disproved by Berger [5] who proved that the Domino Problem over any $\mathbb{Z}^d, d \geq 2$ is algorithmically undecidable. The key of the proof is the existence of a $\mathbb{Z}^d$-subshift of finite type containing only aperiodic configurations, on which computations are implemented. In the decades that followed, many alternative proofs of this fact were provided [19, 17, 14].

The exact conditions to cross this frontier between decidability and undecidability have been intensively studied under different points of view during the last decade. To explore the difference of behavior between $\mathbb{Z}$ and $\mathbb{Z}^2$, the Domino Problem has been extended on discrete groups [7, 11, 6, 1, 2] and fractal structures [4] in order to determine which types of structures can implement computation. The frontier is also studied by restraining complexity (number of patterns of a given size) [15] or bounding the difference between numbers of colors and tiles [13]. Additional dynamical constraints are also considered, such as block gluing property [18, Lemma 3.1] or minimality [9].

In this article we propose a new approach. We fix horizontal constraints H which define a $\mathbb{Z}$-subshift of finite type and consider the decision problem DP_H that given vertical contraints V asks if it is possible to tile the plane. In other words, is the $\mathbb{Z}$-subshift of finite type defined by V compatible with the one defined by H? The purpose is to determine for which horizontal constraints H the decision problem DP_H is decidable.

This point of view has various motivations. First, one could eliminate horizontal constraints that necessarily yield periodic configurations to perform a more efficient computer proof of the smallest aperiodic Wang tile set (reached with 11 Wang tiles in [12]). Second, a classical result is that every effective $\mathbb{Z}$-subshift can be realized as an horizontal projection on $\mathbb{Z}$ of a $\mathbb{Z}^2$-subshift of finite type [10, 3, 8]. However, in these constructions, the subshift in the vertical direction is trivial. We can ask which other vertical subshift can be compatible with a given horizontal restriction: the result presented here is the first step to understand this. Finally, looking for the undecidable frontier under horizontal constraints helps us understand how to transfer information in order to implement computation.

Section 2 recalls the notions needed and formalizes the problem. Section 3 presents three categories of horizontal constraints that yield a decidable Domino Problem. Finally Section 4 proves that all others have an undecidable Domino Problem. This is shown by reduction: we prove that for any Wang tile set W, it is possible to add vertical constraints to these H so that the resulting subshift simulates W. These vertical constraints can control the horizontal transfer of information and implement any Wang tile set, and by extension computation. The proof faces combinatorial explosion into subcases and is based on a careful dichotomy.

2 Definitions

As a preliminary note, *any interval mentioned in this article will be an interval of integers*, unless explicitly stated otherwise.

2.1 Symbolic Dynamics

For a given finite set $\mathcal{A}$ called the alphabet, $\mathcal{A}^{\mathbb{Z}^d}$ is called the *d-dimensional full shift* over $\mathcal{A}$. Any $x \in \mathcal{A}^{\mathbb{Z}^d}$, called a *configuration*, can be seen as a function from $\mathbb{Z}^d$ to $\mathcal{A}$ and we write $x_{\vec{v}} := x(\vec{v})$. For any $\vec{k} \in \mathbb{Z}^d$ define the *shift map* $\sigma^{\vec{k}} : \mathcal{A}^{\mathbb{Z}^d} \to \mathcal{A}^{\mathbb{Z}^d}$ such that $\sigma^{\vec{k}}(x)_{\vec{v}} = x_{\vec{v}+\vec{k}}$. The product topology on $\mathcal{A}^{\mathbb{Z}^d}$ is generated by the metric $d(x, y) = 2^{-\inf\{|\vec{v}||\vec{v}\in\mathbb{Z}^d, x_{\vec{v}}\neq y_{\vec{v}}\}}$, and

makes $\mathcal{A}^{\mathbb{Z}^d}$ a compact space. A *pattern* p is a finite configuration $p \in \mathcal{A}^{P_p}$ where $P_p \subset \mathbb{Z}^d$ is finite. We say that a pattern $p \in \mathcal{A}^{P_p}$ *appears* in a configuration $x \in \mathcal{A}^{\mathbb{Z}^d}$ – or that x *contains* p – if there exists $\vec{k} \in \mathbb{Z}^d$ such that for every $\vec{\ell} \in P_p$, $\sigma^{\vec{k}}(x)_{\vec{\ell}} = p_{\vec{\ell}}$.

A $\mathbb{Z}^d$*-subshift* associated to a set of patterns $\mathcal{F}$, called set of *forbidden patterns*, is defined by

$$X_{\mathcal{F}} = \{x \in \mathcal{A}^{\mathbb{Z}^d} \mid \forall p \in \mathcal{F}, \forall \vec{k} \in \mathbb{Z}^d, \exists \vec{\ell} \in P_p, \sigma^{\vec{k}}(x)_{\vec{\ell}} \neq p_{\vec{\ell}}\}$$

that is, $X_{\mathcal{F}}$ is the set of all configurations that do not contain any pattern from $\mathcal{F}$. Note that there can be several sets of forbidden patterns defining the same subshift X. A subshift can equivalently be defined as a closed set under both the topology and the shift map. If $X = X_{\mathcal{F}}$ with $\mathcal{F}$ finite, then X is called a *Subshift of Finite Type*, SFT for short.

For $\mathbb{Z}$-subshifts, we will talk about *nearest-neighbor* SFTs if $\mathcal{F} \subset \mathcal{A}^{\{0,1\}}$. For $\mathbb{Z}^2$-subshifts, the most well-known nearest-neighbor SFTs are the *Wang shifts*, defined by a finite number of squared tiles with colored edges that must be placed matching colors called Wang tiles. Formally, these tiles are quadruplets of symbols (t_e, t_w, t_n, t_s). A Wang shift is described by a finite Wang tile set, and local rules $x(i,j)_e = x(i+1,j)_w$ and $x(i,j)_n = x(i,j+1)_s$ for all integers i, j.

Note that Wang shifts are enough to encode any $\mathbb{Z}^2$-SFT, albeit changing the underlying local rules and alphabet, so that a $\mathbb{Z}^2$-SFT is empty if and only if the corresponding Wang shift is.

2.2 One-dimensional SFTs as graphs

As explained in [16], a nearest-neighbor SFT $X = X_{\mathcal{F}} \subset \mathcal{A}^{\mathbb{Z}}$, with $\mathcal{F} \subset \mathcal{A}^2$, can be described as an oriented graph $\mathcal{G} = (\mathcal{V}, \vec{E})$ with $\mathcal{V} = \mathcal{A}$ and $(a,b) \in \vec{E} \Leftrightarrow ab \notin \mathcal{F}$. This graph, that encodes the allowed patterns, depends on $\mathcal{A}$ and $\mathcal{F}$, thus two different descriptions of the same SFT will yield different graphs.

However one graph can be canonically associated to a nearest neighbor SFT from any other description of said SFT. It is the only one obtained by iterated suppression of all vertices with no incoming edge or no outgoing edge, and so there only remain biinfinite paths in the graph, that correspond to proper tilings of the line. This graph, denoted by $\mathcal{G}(X)$, is called the *Rauzy graph* (of order 1) of X. Note that a Rauzy graph can be made of one or several *strongly connected components*, *SCC* for short. In case it has several SCCs it can also contain *transient vertices*, that are vertices with no path from themselves to themselves.

▶ **Example 1.** The two subshifts that are $X = X_{\{10,20,21,11,30,31,32,33\}} \subset \{0,1,2,3\}^{\mathbb{Z}}$ and $Y = Y_{\{10,20,21,11\}} \subset \{0,1,2\}^{\mathbb{Z}}$ are the same SFT. They have the same Rauzy graph made of two SCCs $\{0\}$ and $\{2\}$, and one transient vertex 1 (vertex 3 has been deleted from $\mathcal{G}(X)$ else it would be of out-degree 0).

This technique that algorithmically associates a graph to an SFT will be of great use in Section 4, because it means that our proof can mostly focus on combinatorics over graphs to describe all nearest-neighbor one-dimensional SFTs.

2.3 The Domino Problem

Define $DP(\mathbb{Z}^d) = \{< H > | \ H \text{ is a nonempty } \mathbb{Z}^d\text{-SFT}\}$ where $< H >$ is the encoding of the SFT H using a finite alphabet and a finite set of forbidden patterns. $DP(\mathbb{Z}^d)$ is a language called the *Domino Problem* on $\mathbb{Z}^d$. As for any language, we can ask if it is algorithmically

decidable, i.e. recognizable by a Turing Machine. Said otherwise, is it possible to find a Turing Machine that takes as input any *finite* set of patterns $\mathcal{F} \subset \mathcal{A}^{\mathbb{Z}^d}$ of rules and answers `YES` if $X_{\mathcal{F}}$ contains at least one configuration, and `NO` if it is empty?

It is widely known that $DP(\mathbb{Z})$ is decidable, because the problem can be reduced to the emptiness of nearest-neighbor $\mathbb{Z}$-SFTs, and finding a valid configuration in a nearest-neighbor SFT is equivalent, via what precedes, to finding a biinfinite path – hence a cycle – in a finite oriented graph. On the contrary, $DP(\mathbb{Z}^2)$ is undecidable [5, 19, 17, 14], and so is any $DP(\mathbb{Z}^d)$ for $d \geq 2$ by reduction to the undecidability of $DP(\mathbb{Z}^2)$.

2.4 Framework

▶ **Definition 2.** *Let $H, V \subset \mathcal{A}^{\mathbb{Z}}$ be subshifts. The two-dimensional subshift*

$$X_{H,V} := \{x \in \mathcal{A}^{\mathbb{Z}^2} \mid \forall i,j \in \mathbb{Z}, (x_{k,j})_{k\in\mathbb{Z}} \in H \text{ and } (x_{i,\ell})_{\ell\in\mathbb{Z}} \in V\}$$

is called the combined subshift *of H and V, and uses H as horizontal rules and V as vertical rules.*

▶ Remark. The projection of the horizontal configurations that appear in $X_{H,V}$ does not necessarily recover all of H; we may simply have a subset of it. Indeed, all configurations in H will not necessarily appear because some of them may not be legally extended vertically.

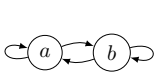
An easy instance of this is choosing $\mathcal{A} = \{0,1\}$, H nearest-neighbor and forbidding 00 and 11, and V non-nearest-neighbor and forcing to alternate a 0 and two 1s: the resulting $X_{H,V}$ is empty, although neither H nor V are. In some sense, said H and V are incompatible.

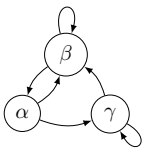
The previous remark motivates the main problem we will study in the rest of this article: understanding when two one-dimensional SFTs are compatible to build a two-dimensional SFT, and by extension where the frontier between decidability and undecidability lies. This question is notably reflected by the following adapted version of the Domino Problem:

▶ **Definition 3.** *Let $H \subset \mathcal{A}^{\mathbb{Z}}$ be an SFT. The* Domino Problem depending on H *is the language*

$$DP_H := \{< V > \mid V \subset \mathcal{A}^{\mathbb{Z}} \text{ is an SFT and } X_{H,V} \neq \emptyset\}.$$

▶ Remark. It is important to understand that this Domino Problem is defined for a given H, and its decidability depends on such a H we choose beforehand. Subshifts can be conjugate, this being defined as a continuous bijection that commutes with the shift maps. Although it allows to identify structurally identical subshifts from a dynamical point of view, these subshifts remain different in how they can encode information. Indeed, some SFT H_1 and H_2 may be conjugate with DP_{H_1} decidable but DP_{H_2} undecidable. Consider for instance the following Rauzy graphs and applications on finite words (extensible to biinfinite words):





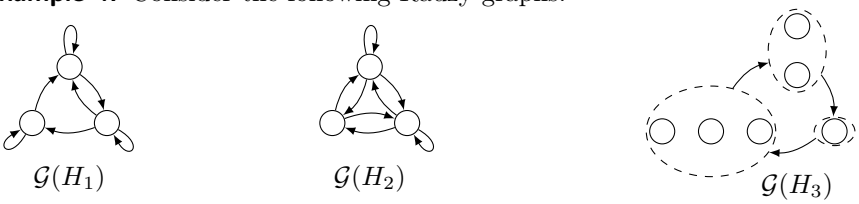
$$\phi : \begin{cases} aa \mapsto \gamma \\ ab \mapsto \beta \\ ba \mapsto \alpha \\ bb \mapsto \beta \end{cases} \qquad \psi : \begin{cases} \alpha \mapsto a \\ \beta \mapsto b \\ \gamma \mapsto a \end{cases}$$

These graphs describe conjugate SFTs through these applications, with $\phi(x)_{i+1} = \phi(x_i x_{i+1})$. However, as we will see in Section 3, the first graph has decidable DP_H and the second has not.

3 Main result: frontier between decidability and undecidability

To understand where our future decidability conditions stem from, we study three examples:

▶ **Example 4.** Consider the following Rauzy graphs:



$\mathcal{G}(H_1)$ $\mathcal{G}(H_2)$ $\mathcal{G}(H_3)$

Let us consider a "vertical" $\mathbb{Z}$-SFT V. As long as one can build a single column respecting the rules of V, this column can legally be juxtaposed with itself in $X_{H_1,V}$ since any element of H_1 can be horizontally juxtaposed with itself due to the self-loop at each vertex. Conversely, if $X_{H_1,V}$ contains a configuration, then in particular it contains a single column respecting the rules of V. Hence checking if $X_{H_1,V}$ is empty is tantamount to checking if V is empty. Since $DP(\mathbb{Z})$ is decidable, DP_{H_1} is easily decidable in this case.

The same reasoning can be applied to the two other cases: for H_2 any pair of columns, and for H_3 any triplet of columns, can be juxtaposed with itself. This finite number of columns makes the decidability of our Domino Problem at stake depend on the decidability of $DP(\mathbb{Z})$. The extensive proof of this fact is located below Theorem 6.

With these three examples, we briefly saw how these three kinds of graphs yielded a decidable DP_H. The rest of this article will produce a proper proof of this and, more importantly, will show that these three rather natural categories are in fact the only ones where decidability appears.

▶ **Definition 5.** *We say that an oriented graph $\mathcal{G} = (\mathcal{V}, \vec{E})$ verifies condition D (for "Decidable") if all its SCCs have a type in common among the following list. A SCC S can be of none, one or several of these types:*

- *for all vertices $v \in S$, we have $(v, v) \in \vec{E}$) (we say that S is of* reflexive type*);*
- *for all vertices $v \neq w \in S$ such that $(v, w) \in \vec{E}$, we have $(w, v) \in \vec{E}$ (we say that S is of* symmetric type*; note that $S = \{v\}$ a single vertex with a loop is also symmetric);*
- *$S = \bigsqcup V_i$ such that $\forall v \in V_i, [(v, w) \in \vec{E} \Leftrightarrow w \in V_{i+1}]$ with i meant modulo the number of classes (we say that S is of* state-split cycle type *in reference to a term used in [16]; note that a partition with one unique class V_0 causes S to be a single vertex with self-loop).*

▶ **Theorem 6.** *Let H be a nearest-neighbor $\mathbb{Z}$-SFT.*

$$DP_H \text{ is decidable} \Leftrightarrow \mathcal{G}(H) \text{ verifies condition } D.$$

Proof. Proof of $\Leftarrow$: assume $\mathcal{G}(H)$ verifies condition D. Then its SCCs share a common type, be it reflexive, symmetric, or state-split cycle. For each of these three cases, we produce an algorithm that takes as input a $\mathbb{Z}$-SFT $V \subset \mathcal{A}^{\mathbb{Z}}$, and that returns `YES` if $X_{H,V}$ is nonempty, and `NO` otherwise.

Let M be the maximal size of forbidden patterns in $\mathcal{F}_V$ (since V is an SFT, such an integer exists).

- If $\mathcal{G}(H)$ has state-split cycle type SCCs: let L be the LCM of the number of V_is in each component. If there is no rectangle of size $L \times M(|\mathcal{A}|^{LM} + 1)$ (width $\times$ height) respecting local rules of $X_{H,V}$ and containing no transient element, then answer `NO`. Indeed, any configuration in $X_{H,V}$ contains valid rectangles as large as we want that do not contain

transient elements. If there is such a rectangle R, then by the pigeonhole principle it contains at least twice the same rectangle R' of size $L \times M$. To simplify the writing, we assume that the rectangle that repeats is the one of coordinates $[1, L] \times [1, M]$ inside R where $[1, L]$ and $[1, M]$ are intervals of integers, and that it can be found again with coordinates $[1, L] \times [k, k + M - 1]$. Else, we simply truncate a part of R so that it becomes true.

Define $P := R|_{[1,L]\times[1,k+M-1]}$. Since V has forbidden patterns of size at most M, and since R respects our local rules and begins and ends with R', P can be vertically juxtaposed with itself (overlapping on R').

P can also be horizontally juxtaposed with itself (without overlap). Indeed, one line of P uses only elements of one SCC of H (since elements of two different SCCs cannot be juxtaposed horizontally, and we banned transient elements). Since L is a multiple of the length of all cycle classes, the first element in a given line can follow the last element in the same line. Hence all lines of P can be juxtaposed with themselves.

As a conclusion, P is a valid patch that can tile $\mathbb{Z}^2$ periodically. Therefore, $X_{H,V}$ is nonempty; return `YES`.

- If $\mathcal{G}(H)$ has symmetric type SCCs the construction is similar, but this time build a rectangle R of size $2 \times M(|\mathcal{A}|^{2M} + 1)$. Either we cannot find one and return `NO`; or we can find one and from it extract a patch that tiles the plane periodically and return `YES`.
- Finally, if $\mathcal{G}(H)$ has reflexive type SCCs, the construction is even simpler than before. Build a rectangle R of size $1 \times M(|\mathcal{A}|^{M} + 1)$; the rest of the reasoning is identical.

Proof of $\Rightarrow$ is postponed to Section 4, and is done by contraposition. If $\mathcal{G}(H)$ does not verify condition D, then for any Wang shift W we can algorithmically build some $\mathbb{Z}$-SFT V_W such that X_{H,V_W} reproduces all configurations in W. If we were able to solve DP_H, then there would exist a Turing Machine $\mathcal{M}$ able to tell us if $X_{H,V}$ is empty for any $\mathbb{Z}$-SFT V. But then we could build a Turing Machine $\mathcal{N}$ taking as input any Wang shift W, building the corresponding V_W after the following construction, and by running $\mathcal{M}$, $\mathcal{N}$ would be able to tell us if X_{H,V_W} is empty or not. Then it could answer if W is empty or not; but determining the emptiness or nonemptiness of every Wang shift is equivalent to $DP(\mathbb{Z}^2)$ being decidable, which is false. Hence, since $DP(\mathbb{Z}^2)$ is undecidable, DP_H is too. ◀

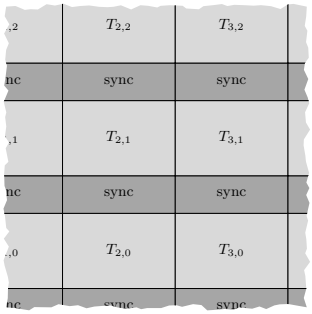
4 Encoding a Wang shift under horizontal constraints

We begin this section by presenting the core idea of our algorithmic construction to prove the direct implication by contraposition. Then, we introduce the vertical patterns needed to achieve it in a generic case, said generic case being based on a set of conditions C. Finally, we show that most graphs that do not verify condition D do verify condition C, and those which don't only need a slight adaptation of our generic construction.

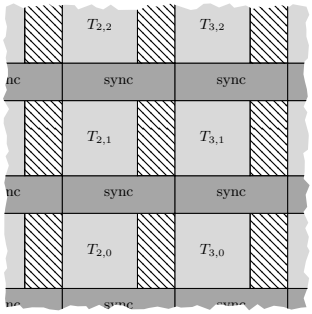
4.1 Core idea

We have a one-dimensional nearest-neighbor SFT $H \subset \mathcal{A}^{\mathbb{Z}}$ ("horizontal") that does not verify condition D, and we fix a Wang shift W with a set of N tiles $\tau = \{\tau_1, ..., \tau_N\}$.

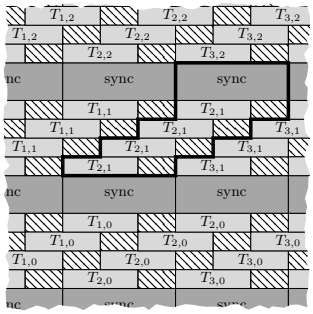
The idea is to introduce a well-chosen one-dimensional SFT $V \subset \mathcal{A}^{\mathbb{Z}}$ ("vertical") depending on W so that $X_{H,V}$ encodes the full shift on N elements. Then, we refine V by adding conditions on forbidden patterns, thus encoding exactly the configurations in W. Such a construction is done with the use of two main parts, that we will obtain by some carefully chosen forbidden patterns in V.



(a) Basic depiction of sync parts and coding parts that represent tiles of W (not to scale; actually unrealizable).



(b) Same construction adding "buffers" between codings of tiles to be realizable.



(c) Improved construction: we encode vertically the horizontal restrictions between tiles of W. A tile of W (here $T_{2,1}$) is represented in bold.

Figure 1 Steps of the core idea to reach the generic construction. Not to scale: the sync part will be much bigger than the code part.

First, there are parts of synchronization, also called *sync parts*, that give some rigidity to our tilings. They precise where the actual coding parts can be, which letters of the alphabet can be used and where in these coding parts, and they ensure that you cannot glue patches together in an unexpected way. They are the frame of our construction. Second comes the filling: the *coding parts*. A given coding part simply codes a number between 1 and N, possibly several times.

In Figure 1a (our first rough attempt to encode a full shift on an alphabet of size N), we suppose that our sync parts properly maintain this global structure. We notice that it offers an interesting opportunity to transmit information vertically. Since our coding parts are exactly aligned, once we have encoded the full shift over an alphabet of size N, it will suffice to add vertical conditions to our V to precise whether a coding part can be above another one.

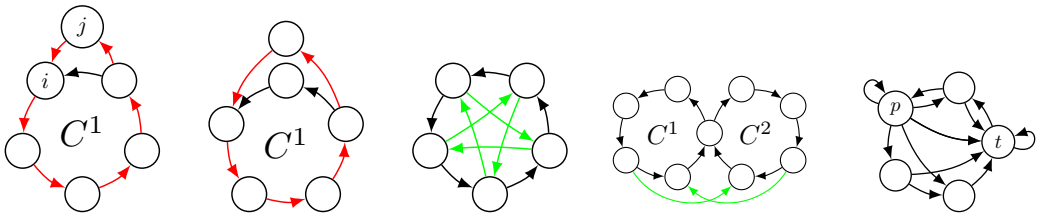
However, horizontally Figure 1a overlooks two problems:

- Since we must respect the horizontal conditions given by H, we cannot put any coding part next to any other one if we do not put some kind of buffer between the two;
- Even with this, we have no control on the horizontal transfer of information. The idea is to transmit this horizontal information vertically, since we can add vertical constraints.

We can fix the first problem by setting a buffer (see Figure 1b) between two coding parts, a portion of column that contains no coding and which can be next to any coding part. Of course, we must ensure that this buffer cannot be anywhere in a configuration but obediently remains between two coding parts. The sync parts will be designed to handle this.

However, this does not solve our need for horizontal transmission of information. Hence a new idea: altering our coding parts so that they transmit information diagonally. We put several consecutive lines of them, shifted little by little, as illustrated in Figure 1c. That way, we can encode horizontal forbidden patterns vertically, because we can see vertically which coding part is on the right of the one we are considering. For instance, by looking vertically we can know that the encoding of $T_{1,1}$ is next to $T_{2,1}$ and above $T_{1,0}$, and thus restrict the content of these codings.

In what follows, we will build Figure 1c in details, although some technicalities will be needed to preserve the integrity of our sync parts and to ensure that the coding of a tile of W is well transmitted. This construction will indeed encode the full shift over τ, the tile set



(a) C^1 and C^2 (in red) have a good pair (i, j).

(b) C^1 and C^2 (in red) have no good pair.

(c) A cycle with uniform shortcuts (in green).

(d) A cross-bridge (in green) between cycles C^1 and C^2.

(e) An attractive vertex t and a repulsive vertex p.

Figure 2 Cases of compliance or not with elements of Condition C.

of W. Then, it can easily be refined by adding vertical rules so that the local rules of W are ensured. Consequently, our newly built $X_{H,V}$ will properly simulate all configurations of W, allowing us to perform the rest of the proof of Theorem 6.

4.2 Generic construction

In this section, we describe a set of conditions on a nearest-neighbor SFT H that allows to build formally what we described informally in Section 4.1. In all that follows, *we denote elements of cycles with an index that is written modulo the length of the corresponding cycle.* The following definitions are illustrated in Figure 2.

▶ **Definition 7.** *Let C^1 and C^2 be two cycles in an oriented graph $\mathcal{G}$, with elements denoted respectively c_i^1 and c_j^2. Let $M := \mathrm{lcm}(|C^1|, |C^2|)$.*

We say that the cycles C^1 and C^2 contain a good pair *if there is a pair (i, j) and an integer $1 < l < M - 1$ such that $c_i^1 \neq c_j^2, c_{i+1}^1 \neq c_{j+1}^2, \dots, c_{i+l}^1 \neq c_{j+l}^2$ and $c_{i+(l+1)}^1 = c_{j+(l+1)}^2, \dots, c_{i+(M-1)}^1 = c_{j+(M-1)}^2$.*

▶ **Definition 8.** *Let $H \subset \mathcal{A}^{\mathbb{Z}}$ be a one-dimensional nearest-neighbor SFT. We say that H* verifies condition C *if $\mathcal{G}(H) = (\mathcal{V}, \vec{E})$ contains two cycles C^1 and C^2, of elements denoted respectively c_i^1 and c_j^2, with the following properties:*

- **(i)** *$|C^1| \geq 3$;*
- **(ii)** *C^1 and C^2 contain a good pair;*
- **(iii)** *(there is no uniform shortcut neither in C^1 nor in C^2) There does not exist any $k \in \{0, 2, ..., |C^1| - 1\}$ such that for any $c_i^1 \in C^1, (c_i^1, c_{i+k}^1) \in \vec{E}$; and there does not exist any $k \in \{0, 2, ..., |C^2| - 1\}$ such that for any $c_j^2 \in C^2, (c_j^2, c_{j+k}^2) \in \vec{E}$;*
- **(iv)** *(there is no cross-bridge between C^1 and C^2) There are no $i \in \{0, ..., |C^1| - 1\}$ and $j \in \{0, ..., |C^2| - 1\}$ with $c_i^1 \neq c_j^2$ and $c_{i+1}^1 \neq c_{j+1}^2$ such that $(c_i^1, c_{j+1}^2) \in \vec{E}$ and $(c_j^2, c_{i+1}^1) \in \vec{E}$;*
- **(v)** *(there cannot be both an attractive vertex and a repulsive vertex for C^1) Either there is no $r \in C^1 \cup C^2$ such that for all $c \in C^1, (c, r) \in \vec{E}$, or there is no $r \in C^1 \cup C^2$ such that for all $c \in C^1, (r, c) \in \vec{E}$.*

▶ **Proposition 9.** *If $\mathcal{G}(H)$ verifies condition C, then DP_H is undecidable.*

The rest of the subsection is devoted to proving this result.

Let H with $\mathcal{G}(H)$ verifying condition C. We focus on encoding a full shift on an alphabet τ of cardinality N. Then, the possibility to add vertical rules will allow us to encode any Wang shift W using this alphabet, that is, to simulate the configurations of W as described in Section 4.1.

For the rest of the construction, we will name $M := \text{lcm}(|C^1|, |C^2|)$ and $K := 2|C^1| + |C^2| + 3$. We suppose that $N \geq 2$ and do not focus on the trivial instance of W being a monotile Wang shift.

We refer to Figure 3a in all that follows. We use the term *slice* as a truncation of a column: it is a part of width 1 and of finite height. We use the following more specific denominations for the various scales of our construction:

- A *macro-slice* is of height KMN. Any column is merely made of a succession of some specific macro-slices called ordered macro-slices (see below).
- A *meso-slice* is of height MN. An ordered macro-slice is made of various meso-slices that ensure it carries information and is correctly aligned with neighboring ordered macro-slices (of neighboring columns).
- A *micro-slice* is of height N. This subdivision is used inside code meso-slices (see below).

Although any scale of slice could denote any truncation of column of the right size, we will only focus on specific slices that are meaningful because of what they contain, so that we can assemble them precisely. They are:

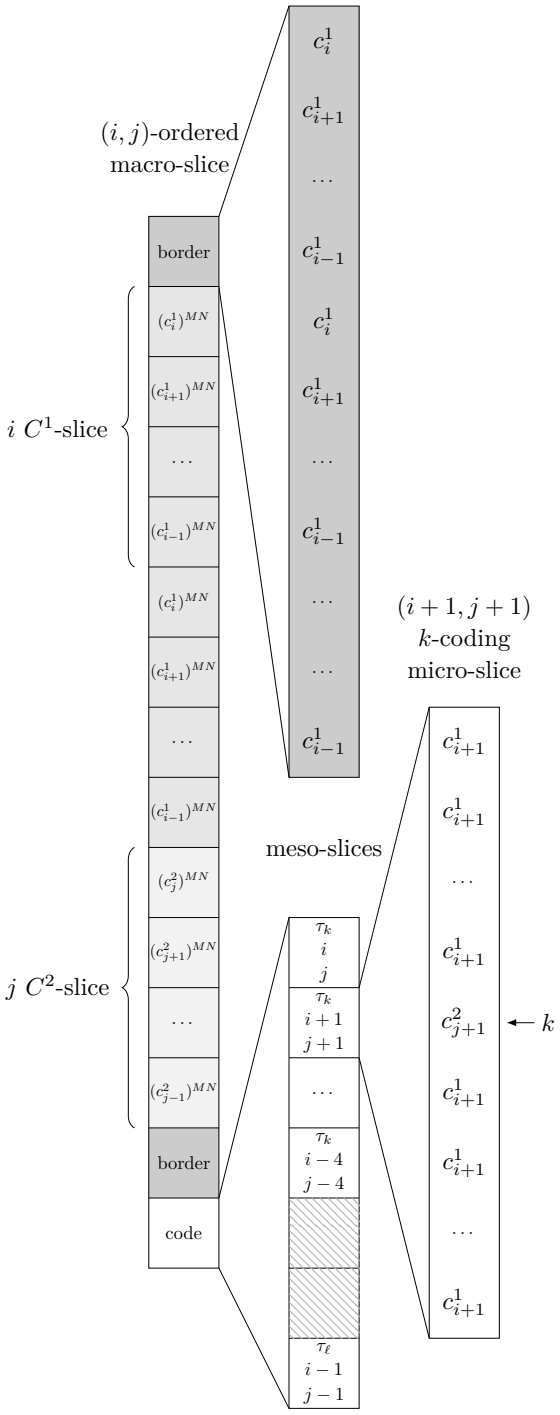
- An (i, j) k-coding micro-slice is a micro-slice composed of $N - 1$ symbols c_i^1 and one symbol c_j^2 at position k. It encodes the kth tile of alphabet τ, unless $c_i^1 = c_j^2$: it is then called a *buffer* and encodes nothing.
- An (i_0, j_0)-code meso-slice is made of M successive coding micro-slices starting with a (i_0, j_0) coding micro-slice (that can encode anything). We add the vertical constraint that if $c_i^1 \neq c_j^2$, then the (i, j) k-coding micro-slice is vertically followed by the $(i + 1, j + 1)$ k-coding micro-slice. If $c_i^1 = c_j^2$, the (i, j) k-coding micro-slice can be vertically followed by any $(i+1, j+1)$ l-coding micro-slice. Note that there can be at most one such rupture in the coding since C^1 and C^2 contain a good pair; k is then called the *main-coded tile*, and l the *side-coded tile*.
- An i-border meso-slice is made of $\frac{M}{|C^1|}N$ times the vertical repetition of elements of the cycle C^1, starting at c_i^1.
- A c_i^1 meso-slice is made of MN times the vertical repetition of element c_i^1, denoted $(c_i^1)^{MN}$ in Figure 3a. Same for a c_j^2 meso-slice.
- The succession of a c_i^1 meso-slice, then a c_{i+1}^1 meso-slice, ..., then a c_{i-1}^1 meso-slice is called a i C^1-slice (of height $MN|C^1|$). Similarly, we define a j C^2-slice (of height $MN|C^2|$).
- Finally, a (i, j)-ordered macro-slice is the succession of a i-border meso-slice, a i C^1-slice, a second i C^1-slice, a j C^2-slice, a i-border meso-slice, and finally a (i, j)-code meso-slice.

Now, the patterns we authorize in V are exactly the cyclic permutations of $(i_0 + k, j_0 + k)$-ordered macro-slices with some good pair (i_0, j_0) and $k \in \{0, \dots, M - 1\}$. As a consequence, a given column is simply the repetition of a given (i, j)-ordered macro-slice. We prove below that it is enough for our resulting $X_{H,V}$ to simulate a full shift on τ.

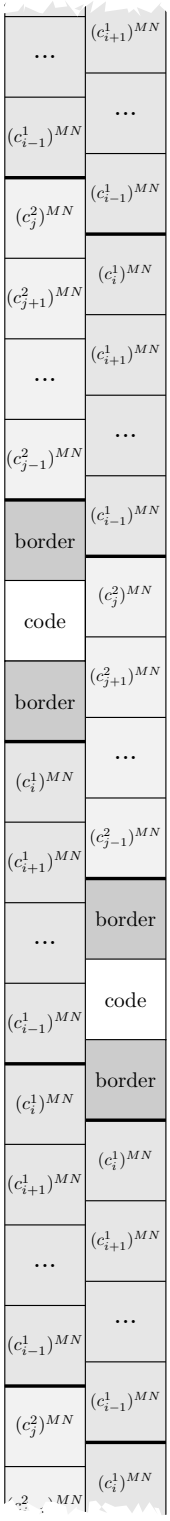
We say that two legally adjacent columns are *aligned* if they are subdivided into ordered macro-slices exactly on the same lines. We say that two adjacent and aligned columns are *synchronized* if any (i, j)-ordered macro-slice of the first one is followed by a $(i + 1, j + 1)$-ordered macro-slice in the second one.

▶ **Proposition 10.** *In this construction, two legally adjacent columns are aligned up to a vertical translation of size at most* $2|C^1| - 1$ *of one of the columns.*

Proof. If two columns, call them K_1 and K_2, can be legally juxtaposed such that they are not aligned even when vertically shifted by $2|C^1| - 1$ elements, it means that one of the border meso-slices of K_1 has at least $2|C^1|$ vertically consecutive elements that are horizontally followed by something that is not a border meso-slice in K_2 (see Figure 3b).



(a) Columns allowed, for (i, j) in the orbit of a good pair. Here $c^1_{i-3} = c^2_{j-3}$ and $c^1_{i-2} = c^2_{j-2}$, forming buffers in the code meso-tile.



(b) Faulty alignment of adjacent columns.

Figure 3 The generic construction.

Since $2|C^1| < MN$, at least $|C^1|$ successive elements among the ones of the border meso-slice are horizontally followed by elements that are part of the same meso-slice. If this is a code meso-slice, simply consider the other border meso-slice of K_1 (the first you can find, above or below, before repeating the pattern cyclically): this one must be in contact with a c_i^1 or c_j^2 meso-slice instead. Either way, we obtain that a border meso-slice has at least $|C^1|$ successive elements that are horizontally followed by some t meso-slice made of a single element t. Hence if we suppose that juxtaposing K_1 and K_2 this way is legal, it means that in H all the elements of C^1 lead to t, i.e t is an attractive vertex. Either this is forbidden, or the "reverse" reasoning where we focus on the borders of K_2 proves that there is also an element p used in a C^j slice of K_1 that leads to every element of C^1; that is, a repulsive vertex. We forbade any graph that had both, hence we reach a contradiction here. We obtain the proposition we announced. ◀

▶ **Proposition 11.** *In this construction, two legally adjacent columns are always aligned and synchronized.*

Proof. Proposition 10 states that two adjacent columns K_1 and K_2 are always, in some sense, approximately aligned (up to a vertical translation of size at most $2|C^1| - 1$). If the two columns are indeed slightly shifted, then any meso-slice of the C^1 slice of K_1 (consisting only of the repetition of some c_i^1) is horizontally followed by two different meso-slices in K_2. Being different, at least one of them is not c_{i+1}^1 but some $c_{i+k}^1, k \in \{2, ..., |C^1|\}$. This is true with the same k for all values of i because all meso-slices representing c_i^1 are repeated twice so that there is no "border effect". We obtain something that contradicts our assumption that C^1 has no uniform shortcut. Hence there is no vertical shift at all between two consecutive columns. Thus our construction ensures that two adjacent columns are always aligned.

It is easy to see that a meso-slice made only of c_i^1 in column K_1 is horizontally followed, because the columns are aligned, by a meso-slice made only of c_{i+k}^1 in column K_2. This k is once again independent of the i because inside a macro-slice, meso-slices respect the order of cycle C^1. But because C^1 has no uniform shortcut, we must have $k = 1$. The reasoning is the same for the C^2 slice, and we use the fact that C^2 has no shortcut either. Hence our columns are synchronized. ◀

With these properties, we have ensured that our structure is rigid: our ordered macro-slices are aligned just as we expected. The last fact to check is the transmission of information between horizontally aligned code meso-slices (since, by the structure of a code meso-slice, the vertical transmission is guaranteed).

We have to ensure that, every M horizontally aligned coding micro-slices, we have a succession of buffers (a (i, j)-coding micro-slice that does not encode information because $c_i^1 = c_j^2$) then of non-buffers. Furthermore, we ask that all the buffers follow each other so that we get some distinct "coding zone" and "buffer zone". This is actually simply deduced from the fact that we only authorized as ordered macro-slices the ones based on the orbit of a good pair (see Definition 7).

Now, in which situation can there be a problem of horizontal transmission of the encoded tile between two micro-slices? Suppose we have a (i, j) micro-slice then a $(i + 1, j + 1)$ micro-slice. A problem of transmission would mean that c_i^1 can be followed by c_{j+1}^2, and c_j^2 by c_{i+1}^1. A problem of transmission also assumes that we transmit something, hence we don't consider buffer micro-slices: necessarily $c_i^1 \neq c_j^2$ and $c_{i+1}^1 \neq c_{j+1}^2$. Then we would contradict assumption (iv) "no cross-bridge" (which was assumed precisely to prevent this case).

As a consequence of all this, every M micro-slices starting with a buffer micro-slice, horizontally successive coding micro-slices encode exactly one element of τ, since there is one single coding zone and the coded tile is correctly transmitted.

Figure 4 Some Rauzy graph and several associated code meso-slices for $|\tau| = 3$. Here are horizontally successively encoded τ_3, τ_1, τ_2 and τ_2, the number being indicated by the location of the line of c's.

In the end, we proved that if we were able to find such C^1 and C^2 complying with condition C, they would be enough to build the construction we desire: a full shift on N elements. Then, to encode only configurations that are valid in W, we forbid the following additional vertical patterns:

- the code meso-slices that would contain both the main-coded kth tile and the side-coded lth tile if tile k cannot be horizontally followed by tile l in W;
- and the vertical succession of two ordered macro-slices that would contain code meso-slices with two main-coded tiles that cannot be vertically successive in W.

With this, we proved Proposition 9.

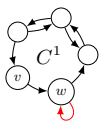
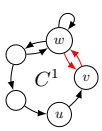
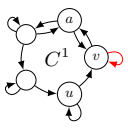
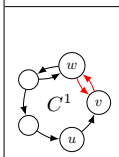
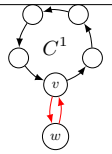
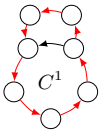
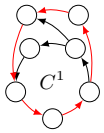
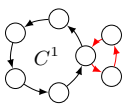
4.3 Proof of Theorem 6 for one strongly connected component

We suppose that $H \subset \mathcal{A}^{\mathbb{Z}}$ is a one-dimensional nearest-neighbor SFT such that its Rauzy graph does not verify condition D and is made of only one SCC.

Note that $\mathcal{G}(H)$, since it does not verify condition D, contains at least one loopless vertex, and one unidirectional edge.

The idea is to divide the possible graphs into various cases. This way, one has a standard procedure to find convenient C^1 and C^2 inside any graph to perform the generic construction. Of course, for some specific cases, we won't meet condition C even if H does not verify condition D. However, we will punctually adapt the generic construction to these specificities.

Table 1 Table of the main cases, each of them illustrated with an example (the C^2 on which we perform the generic construction is in red).

Loops			No loop				
			Bidirectional edges		No bidirectional edge		
							
Case 1.1	Case 1.2	Case 1.3	Case 2.1	Case 2.2	Case 3.1	Case 3.2	Case 3.3

The division into cases is presented in a disjunctive fashion, see Table 1:
Is there a loop on a vertex?

- If YES: Is there a unidirectional edge $(v, w) \in \vec{E}$ so that v is loopless and w has a loop (or the reverse, which is similar)?
  - If YES: This is Case 1.1. We can find C^1 and C^2 that check condition C with the exception of the possible presence of both an attractive and a repulsive vertices. However, Proposition 10 is verified anyway because choosing the smallest possible cycle containing such v and w, v has in-degree 1, a property that allows for an easy synchronization.
  - If NO: Do unidirectional edges have loopless vertices?
    - If YES: This is Case 1.2. We can find C^1 and C^2 that check condition C, possibly by reducing to a situation encountered in case 2.2.
    - If NO: This is Case 1.3. This one generates some exceptional graphs with 4 or 5 vertices that do not check condition C and must be treated separately. However, technical considerations prove that our generic construction still works.
- If NO: Is there a bidirectional edge?
  - If YES: Is there a cycle of size at least 3 that contains a bidirectional edge?
    - If YES: This is Case 2.1. We can find C^1 and C^2 that verify condition C rather easily.
    - If NO: This is Case 2.2, in which checking condition C is also easy.
  - If NO: Is there a minimal cycle with a path between two *different* elements of it, say c_0^1 and c_k^1, that does not belong to the cycle?
    - If YES: Can we find such a path of length different from k?
      - If YES: This is Case 3.1, a rather tedious case, but we can find cycles C^1 and C^2 that verify condition C nonetheless.
      - If NO: This is Case 3.2, which relies heavily on the fact that $\mathcal{G}(H)$ is not of state-split cycle type to find cycles that verify condition C.
    - If NO: This is Case 3.3, an easy case to find cycles that verify condition C.

4.4 Proof of Theorem 6 for multiple strongly connected components

The idea if H has several SCCs is to build one, by products of SCCs, that is none of the three types that constitute condition D. We can then apply what we did in the previous subsections.

The direct product $S_1 \times S_2$ of two SCCs S_1 and S_2 is made of pairs (s_1, s_2), where an edge exists between two pairs if and only if edges exist in both S_1 and S_2 between the corresponding vertices. It can be used in our construction by forcing pairs of elements $(s_1, s_2) \in S_1 \times S_2$ to be vertically one on top of the other.

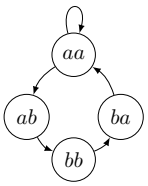
Since H does not verify condition D, it has a non-reflexive SCC S_1, a non-symmetric SCC S_2 and a non-state-split SCC S_3 (two of them being possibly the same). But then:

- Since S_1 is non-reflexive, no SCC of $S_1 \times S_2 \times S_3$ is reflexive. Indeed, since S_1 is strongly connected, all vertices of S_1 are represented in any SCC C of that graph product, meaning that for any $s_1 \in S_1$ there is at least one vertex of the form $(s_1, *, *)$ in C. But if C had loop on all its vertices, then in particular S_1 would be reflexive.
- Similarly, since S_2 is non-symmetric, no SCC of $S_1 \times S_2 \times S_3$ is symmetric.
- Finally, since S_3 is non-state-split, no SCC of $S_1 \times S_2 \times S_3$ is a state-split cycle. Indeed, suppose S is such a state-split SCC of the direct product. It can be written as a collection of classes $(V_i)_{i \in I}$ of elements from $S_1 \times S_2 \times S_3$ that we can project onto S_3, getting

new classes $(W_i)_{i\in I}$, with some elements of S_3 that possibly appear in several of these. Let c be any vertex in S_3 that appears at least twice *with the least difference of indices between two classes where it appears*; say $c \in W_i$ and $c \in W_{i+k}$. Since S is state-split, all elements in W_{i+1} are exactly the elements of S_3 to which c leads. But it is the same for W_{i+k+1}. Hence $W_{i+1} = W_{i+k+1}$. From this we deduce that $W_i = W_{i+k}$ for any i, using the fact that indices are modulo $|I|$. Since k is the smallest possible distance between classes having a common element, classes from $(W_i)_{i\in\{0,\dots,k-1\}}$ are all disjoint; and they obviously contain all vertices from S_3. Now simply consider these classes W_0 to W_{k-1}: you get the proof that S_3 is state-split.

Perspectives

Nearest-neighbor conditions are strong constraints, hence it is rather coherent that apart from some very simple graphs the undecidability of DP_H is systematic. Investigation has begun about non-nearest-neighbor constraints, for which there seem to be more graphs with a decidable Domino Problem, this set of graphs possibly being non-recursive. For instance, the graph below is of decidable Domino Problem.



Another perspective would be to generalize these results to $\mathbb{Z}^d$: we fix restrictions on a $\mathbb{Z}^k$-SFT with $k < d$ and look at the consequent decidability for $\mathbb{Z}^d$-SFTs. It is immediate that if $d - k \geq 2$ then we can reduce to $DP(\mathbb{Z}^2)$ so this new problem is always undecidable. However, the $d - k = 1$ case – that is, fixing a hyperplane – is still open.

References

1 Nathalie Aubrun, Sebastián Barbieri, and Emmanuel Jeandel. *About the Domino Problem for Subshifts on Groups*, chapter 9, pages 331–389. Springer International Publishing, 2019. doi:10.1007/978-3-319-69152-7_9.

2 Nathalie Aubrun, Sebastián Barbieri, and Etienne Moutot. The domino problem is undecidable on surface groups. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, pages 46:1–46:14, 2019. doi:10.4230/LIPIcs.MFCS.2019.46.

3 Nathalie Aubrun and Mathieu Sablik. Simulation of Effective Subshifts by Two-dimensional Subshifts of Finite Type. *Acta Appl. Math.*, 126:35–63, 2013. doi:10.1007/s10440-013-9808-5.

4 Sebastián Barbieri and Mathieu Sablik. The domino problem for self-similar structures. In *Pursuit of the universal*, volume 9709 of *Lecture Notes in Comput. Sci.*, pages 205–214. Springer, [Cham], 2016. doi:10.1007/978-3-319-40189-8_21.

5 Robert Berger. *Undecidability of the Domino Problem*, volume 66 of *Memoirs AMS*. American Mathematical Society, 1966.

6 David B. Cohen. The large scale geometry of strongly aperiodic subshifts of finite type. *ArXiv e-prints*, December 2014. arXiv:1412.4572.

7 David B. Cohen and Chaim Goodman-Strauss. Strongly aperiodic subshifts on surface groups. *ArXiv e-prints*, October 2015. arXiv:1510.06439.

8 Bruno Durand, Andrei E. Romashchenko, and Alexander Shen. Fixed-point tile sets and their applications. *J. Comput. Syst. Sci.*, 78(3):731–764, 2012. doi:10.1016/j.jcss.2011.11.001.

9 Silvère Gangloff and Mathieu Sablik. Simulation of minimal effective dynamical systems on the cantor sets by minimal tridimensional subshifts of finite type. *Prépublication (http://arxiv.org/abs/1806.07799)*, 2018.

10 Michael Hochman. On the dynamics and recursive properties of multidimensional symbolic systems. *Inventiones mathematicae*, 176(1):131, December 2008. doi:10.1007/s00222-008-0161-7.

11 Emanuel Jeandel. Aperiodic subshifts of finite type on groups. *https://hal.inria.fr/hal-01110211*, 2015.

12 Emmanuel Jeandel and Michaël Rao. An aperiodic set of 11 wang tiles. *CoRR*, abs/1506.06492, 2015. arXiv:1506.06492.

13 Emmanuel Jeandel and Nicolas Rolin. Fixed parameter undecidability for wang tilesets. In *Proceedings 18th international workshop on Cellular Automata and Discrete Complex Systems and 3rd international symposium Journées Automates Cellulaires, AUTOMATA & JAC 2012, La Marana, Corsica, September 19-21, 2012.*, pages 69–85, 2012. doi:10.4204/EPTCS.90.6.

14 Jarkko Kari. The tiling problem revisited (extended abstract). In Jérôme Durand-Lose and Maurice Margenstern, editors, *Machines, Computations, and Universality*, pages 72–79, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi:10.1007/978-3-540-74593-8_6.

15 Jarkko Kari and Etienne Moutot. Decidability and periodicity of low complexity tilings. *CoRR*, abs/1904.01267, 2019. arXiv:1904.01267.

16 Douglas Lind and Brian Marcus. *Symbolic Dynamics and Coding*. Cambridge University Press, 1995.

17 Shahar Mozes. Tilings, substitution systems and dynamical systems generated by them. *Journal d'Analyse Mathématique*, 53(1):139–186, December 1989. doi:10.1007/BF02793412.

18 Ronnie Pavlov and Michael Schraudner. Entropies realizable by block gluing $\mathbb{Z}^d$ shifts of finite type. *Journal d'Analyse Mathématique*, 126(1):113–174, April 2015. doi:10.1007/s11854-015-0014-4.

19 Raphael M. Robinson. Undecidability and nonperiodicity for tilings of the plane. *Inventiones mathematicae*, 12(3):177–209, September 1971. doi:10.1007/BF01418780.

20 Hao Wang. Proving theorems by pattern recognition – II. *The Bell System Technical Journal*, 40(1):1–41, 1961. doi:10.1002/j.1538-7305.1961.tb03975.x.

Computing Maximum Matchings in Temporal Graphs

George B. Mertzios
Department of Computer Science, Durham University, UK
george.mertzios@durham.ac.uk

Hendrik Molter
TU Berlin, Faculty IV, Algorithmics and Computational Complexity, Berlin, Germany
h.molter@tu-berlin.de

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

Viktor Zamaraev
Department of Computer Science, University of Liverpool, UK
viktor.zamaraev@liverpool.ac.uk

Philipp Zschoche
TU Berlin, Faculty IV, Algorithmics and Computational Complexity, Berlin, Germany
zschoche@tu-berlin.de

Abstract

Temporal graphs are graphs whose topology is subject to discrete changes over time. Given a static underlying graph G, a temporal graph is represented by assigning a set of integer time-labels to every edge e of G, indicating the discrete time steps at which e is active. We introduce and study the complexity of a natural temporal extension of the classical graph problem Maximum Matching, taking into account the dynamic nature of temporal graphs. In our problem, Maximum Temporal Matching, we are looking for the largest possible number of time-labeled edges (simply *time-edges*) (e, t) such that no vertex is matched more than once within any time window of Δ consecutive time slots, where $\Delta \in \mathbb{N}$ is given. The requirement that a vertex cannot be matched twice in any Δ-window models some necessary "recovery" period that needs to pass for an entity (vertex) after being paired up for some activity with another entity. We prove strong computational hardness results for Maximum Temporal Matching, even for elementary cases. To cope with this computational hardness, we mainly focus on fixed-parameter algorithms with respect to natural parameters, as well as on polynomial-time approximation algorithms.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Fixed parameter tractability; Theory of computation → Approximation algorithms analysis

Keywords and phrases Temporal Graph, Link Stream, Temporal Line Graph, NP-hardness, APX-hardness, Approximation Algorithm, Fixed-parameter Tractability, Independent Set

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.27

Related Version A full version of the paper is available at `https://arxiv.org/abs/1905.05304`.

Funding *George B. Mertzios*: Supported by the EPSRC grant EP/P020372/1.
Hendrik Molter: Supported by the DFG, project MATE (NI369/17).
Viktor Zamaraev: Supported by the EPSRC grant EP/P020372/1. The main part of this paper was prepared while affiliated with the Department of Computer Science, Durham University, UK.

37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020).
Editors: Christophe Paul and Markus Bläser; Article No. 27; pp. 27:1–27:14

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

1 Introduction

Computing a maximum matching in an undirected graph (a maximum-cardinality set of "independent edges", i.e., edges which do not share any endpoint) is one of the most fundamental graph-algorithmic primitives. In this work, we lift the study of the algorithmic complexity of computing maximum matchings from static graphs to the – recently strongly growing – field of *temporal graphs* [15, 18]. In a nutshell, a temporal graph is a graph whose topology is subject to discrete changes over time. We adopt a simple and natural model for temporal graphs which originates in the foundational work of Kempe et al. [16]. According to this model, every edge of a static graph is given along with a set of time labels, while the vertex set remains unchanged.

▶ **Definition 1** (Temporal Graph). *A* temporal graph $\mathcal{G} = (G, \lambda)$ *is a pair* (G, λ)*, where* $G = (V, E)$ *is an underlying (static) graph and* $\lambda : E \to 2^{\mathbb{N}} \setminus \{\emptyset\}$ *is a* time-labeling *function that specifies which edge is* active *at what time.*

An alternative way to view a temporal graph is to see it as an ordered set (according to the discrete time slots) of graph instances (called *snapshots*) on a fixed vertex set. Due to their vast applicability in many areas, temporal graphs have been studied from different perspectives under various names such as *time-varying*, *evolving*, *dynamic*, and *graphs over time*.

In this paper we introduce and study the complexity of a natural temporal extension of the classical problem Maximum Matching, which takes into account the dynamic nature of temporal graphs. To this end, we extend the notion of "edge independence" to the temporal setting: two time-labeled edges (simply *time-edges*) (e, t) and (e', t') are Δ-*independent* whenever (i) the edges e, e' do not share an endpoint or (ii) their time labels t, t' are at least Δ time units apart from each other.[1] Then, for any given Δ, the problem Maximum Temporal Matching asks for the largest possible set of pairwise Δ-independent edges in a temporal graph. That is, in a feasible solution, no vertex can be matched more than once within any time window of length Δ. The concept of Δ-windows has been employed in many different temporal graph problem settings [1, 7, 14, 19]. It is particularly important to understand the complexity of the problem in the case where Δ is a constant, since this models short "recovery" periods.

Our main motivation for studying Maximum Temporal Matching is of theoretical nature, namely to lift one of the most classical optimization problems, Maximum Matching, to the temporal setting. As it turns out, Maximum Temporal Matching is computationally hard to approximate: we prove that the problem is APX-hard, even when $\Delta = 2$ and the lifetime T of the temporal graph (i.e., the maximum edge label) is 3 (see Section 3.1). That is, unless P=NP, there is no Polynomial-Time Approximation Scheme (PTAS) for any $\Delta \geq 2$ and $T \geq 3$. In addition, we show that the problem remains NP-hard even if the underlying graph G is just a path (see Section 3.2). Consequently, we mainly turn our attention to approximation and to fixed-parameter algorithms (see Section 4).

In order to prove our hardness results (see Section 3), we introduce the notion of a *temporal line graph*[2] which is a class of (static) graphs of independent interest and may prove useful in other contexts, too. This notion enables us to reduce Maximum Temporal

[1] Throughout the paper, Δ always refers to that number, and never to the maximum degree of a static graph (which is another common use of Δ).

[2] We remark that a different notion of temporal line graphs was introduced in a survey by Latapy et al. [18], which is somewhat similar to our definition for the case of $\Delta = 1$.

Matching to the problem of computing a large independent set in a static graph (i.e., in the temporal line graph that is defined from the input temporal graph). Moreover, as an intermediate result, we show (see Theorem 11) that the classic problem Independent Set (on static graphs) remains NP-hard on induced subgraphs of *diagonal grid* graphs, thus strengthening an old result of Clark et al. [9] for unit disk graphs.

During the last few decades it has been repeatedly observed that for many variations of Maximum Matching it is straightforward to obtain online (resp. greedy offline approximation) algorithms which achieve a competitive (resp. an approximation) ratio of $\frac{1}{2}$, while great research efforts have been made to increase the ratio to $\frac{1}{2}+\varepsilon$, for *any* constant $\varepsilon > 0$. It is well known that an arbitrary greedy algorithm for matching gives approximation ratio at least $\frac{1}{2}$ [13, 17], while it remains a long-standing open problem to determine how well a randomized greedy algorithm can perform. Aronson et al. [3] provided the so-called Modified Randomized Greedy (MRG) algorithm which approximates the maximum matching within a factor of at least $\frac{1}{2}+\frac{1}{400{,}000}$. Recently, Poloczek and Szegedy [20] proved that MRG actually provides an approximation ratio of $\frac{1}{2}+\frac{1}{256}$. Similarly to the above problems, it is straightforward[3] to approximate Maximum Temporal Matching in polynomial time within a factor of $\frac{1}{2}$. However, we manage to provide a simple approximation algorithm which, for any constant Δ, achieves an approximation ratio $\frac{1}{2}+\varepsilon$ for a constant ε. For $\Delta = 2$ this ratio is $\frac{2}{3}$, while for an arbitrary constant Δ it becomes $\frac{\Delta}{2\Delta-1} = \frac{1}{2}+\frac{1}{2(2\Delta-1)}$ (see Section 4.1).

Given that Maximum Temporal Matching is NP-hard, we show fixed-parameter tractability with respect to the desired solution size parameter. From a parameterized classification standpoint, this improves a result of Baste et al. [6] who needed additionally Δ as a second parameter for fixed-parameter tractability.

Finally, we show fixed-parameter tractability with respect to the combined parameter Δ and size of a maximum matching of the underlying graph (which may be significantly smaller than the cardinality of a maximum temporal matching of the temporal graph). Our algorithmic techniques are essentially based on kernelization and matroid theory (see Section 4).

It is worth mentioning that another temporal variation of Maximum Matching, which is related to ours, was recently proposed by Baste et al. [6]. The main difference is that their model requires edges to exist in at least Δ *consecutive* snapshots in order for them to be eligible for a matching. Thus, their matchings need to consist of time-consecutive edge blocks, which requires some data cleaning on real-word instances in order to perform meaningful experiments [6].

It turns out that the model of Baste et al. is a special case of our model, as there is an easy reduction from their model to ours, and thus their positive results are also implied by ours. Baste et al. [6] showed that solving (using their definition) Maximum Temporal Matching is NP-hard for $\Delta \geq 2$. In terms of parameterized complexity, they provided a polynomial-sized kernel for the combined parameter (k, Δ), where k is the size of the desired solution.

We see the concept of multistage (perfect) matchings, introduced by Gupta et al. [12], as the main alternative model for temporal matchings in temporal graphs. This model, which is inspired by reconfiguration or reoptimization problems, is not directly related to ours:

[3] To achieve the straightforward $\frac{1}{2}$-approximation it suffices to just greedily compute at every time slot a maximal matching among the edges that are Δ-independent with the edges that were matched in the previous time slots.

roughly speaking, their goal is to find perfect matchings for every snapshot of a temporal graph such that the matchings only slowly change over time. In this setting one mostly encounters computational intractability, which leads to several results on approximation hardness and algorithms [5, 12].

Several details and proofs (marked with $\star$) are omitted due to space constraints.

2 Preliminaries

We use standard mathematical and graph-theoretic notation. In the full version of this paper there is an overview of the most important classical notation and terminology we use.

Temporal graphs. Throughout the paper we consider temporal graphs $\mathcal{G}$ with *finite lifetime* $T(\mathcal{G}) = \max\{t \in \lambda(e) \mid e \in E\}$, that is, there is a maximum label assigned by λ to an edge of G. When it is clear from the context, we denote the lifetime of $\mathcal{G}$ simply by T. The *snapshot* (or *instance*) of $\mathcal{G}$ *at time* t is the static graph $G_t = (V, E_t)$, where $E_t = \{e \in E \mid t \in \lambda(e)\}$. We refer to each integer $t \in [T]$ as a *time slot* of $\mathcal{G}$. For every $e \in E$ and every time slot $t \in \lambda(e)$, we denote the *appearance of edge* e *at time* t by the pair (e, t), which we also call a *time-edge*. We denote the set of edge appearances of a temporal graph $\mathcal{G} = (G = (V, E), \lambda)$ by $\mathcal{E}(\mathcal{G}) := \{(e, t) \mid e \in E \text{ and } t \in \lambda(e)\}$. For every $v \in V$ and every time slot t, we denote the *appearance of vertex* v *at time* t by the pair (v, t). That is, every vertex v has T different appearances (one for each time slot) during the lifetime of $\mathcal{G}$. For every time slot $t \in [T]$, we denote by $V_t = \{(v, t) : v \in V\}$ the set of all vertex appearances of $\mathcal{G}$ at time slot t. Note that the set of all vertex appearances in $\mathcal{G}$ is $V \times [T] = \bigcup_{1 \le t \le T} V_t$. Two vertex appearances (v, t) and (w, t) are *adjacent* if the temporal graph has the time-edge $(\{v, w\}, t)$. For a temporal graph $\mathcal{G} = (G, \lambda)$ and a set of time-edges M, we denote by $\mathcal{G} \setminus M := (G', \lambda')$ the temporal graph $\mathcal{G}$ without the time-edges in M, where $G' := (V, E')$ with $E' := \{e \in E \mid \lambda(e) \setminus \{t \mid (e, t) \in M\} \neq \emptyset\}$ and for all $e \in E'$, $\lambda'(e) := \lambda(e) \setminus \{t \mid (e, t) \in M\}$. For a subset $S \subseteq [T]$ of time slots and a time-edge set M, we denote by $M|_S := \{(e, t) \in M \mid t \in S\}$ the set of time-edges in M with a label in S. For a temporal graph $\mathcal{G}$, we denote by $\mathcal{G}|_S := \mathcal{G} \setminus (\mathcal{E}(\mathcal{G})|_{[T] \setminus S})$ the temporal graph where only time-edges with label in S are present.

In the remainder of the paper we denote by n and m the number of vertices and edges of the underlying graph G, respectively, unless otherwise stated. We assume that there is no compact representation of the labeling λ, that is, $\mathcal{G}$ is given with an explicit list of labels for every edge, and hence the *size* of a temporal graph $\mathcal{G}$ is $|\mathcal{G}| := |V| + \sum_{t=1}^{T} |E_t| \in O(n + mT)$. Furthermore, in accordance with the literature [23, 24] we assume that the lists of labels are given in ascending order.

Temporal matchings. A *matching* in a (static) graph $G = (V, E)$ is a set $M \subseteq E$ of edges such that for all $e, e' \in M$ we have that $e \cap e' = \emptyset$. In the following, we transfer this concept to temporal graphs.

For a natural number Δ, two time-edges (e, t), (e', t') are Δ-*independent* if $e \cap e' = \emptyset$ or $|t - t'| \ge \Delta$. If two time-edges are not Δ-independent, then we say that they are *in conflict*. A time-edge (e, t) Δ-*blocks* a vertex appearance (v, t') (or (v, t') is Δ-*blocked* by (e, t)) if $v \in e$ and $|t - t'| \le \Delta - 1$. A Δ-*temporal matching* M of a temporal graph $\mathcal{G}$ is a set of time-edges of $\mathcal{G}$ which are pairwise Δ-independent. Formally, it is defined as follows.

▶ **Definition 2** (Δ-Temporal Matching). *A* Δ-temporal matching *of a temporal graph* $\mathcal{G}$ *is a set* M *of time-edges of* $\mathcal{G}$ *such that for every pair of distinct time-edges* $(e, t), (e', t')$ *in* M *we have that* $e \cap e' = \emptyset$ *or* $|t - t'| \ge \Delta$.

We remark that this definition is similar to the definition of γ-*matchings* by Baste et al. [6].

A Δ-temporal matching is called *maximal* if it is not properly contained in any other Δ-temporal matching. A Δ-temporal matching is called *maximum* if there is no Δ-temporal matching of larger cardinality. We denote by $\mu_\Delta(\mathcal{G})$ the size of a maximum Δ-temporal matching in $\mathcal{G}$.

Having defined temporal matchings, we naturally arrive at the following central problem.

Maximum Temporal Matching	
Input:	A temporal graph $\mathcal{G} = (G, \lambda)$ and an integer $\Delta \in \mathbb{N}$.
Output:	A Δ-temporal matching in $\mathcal{G}$ of maximum cardinality.

We refer to the problem of deciding whether a given temporal graph admits a Δ-temporal matching of given size k by Temporal Matching.

For some basic observations about our problem settings and more details about the relation between our model and the model of Baste et al. [6] we refer to the full version of this paper.

Temporal line graphs. In the following, we transfer the concept of line graphs to temporal graphs and temporal matchings. In particular, we make use of temporal line graphs in the NP-hardness result of Section 3.2.

The Δ-*temporal line graph* of a temporal graph $\mathcal{G}$ is a static graph that has a vertex for every time-edge of $\mathcal{G}$ and two vertices are connected by an edge if the corresponding time-edges are in conflict, i.e., they cannot be both part of a Δ-temporal matching of $\mathcal{G}$. We say that a graph H is a *temporal line graph* if there exist a Δ and a temporal graph $\mathcal{G}$ such that H is isomorphic to the Δ-temporal line graph of $\mathcal{G}$. Formally, temporal line graphs and Δ-temporal line graphs are defined as follows.

▶ **Definition 3** (Temporal Line Graph). *Given a temporal graph $\mathcal{G} = (G = (V, E), \lambda)$ and a natural number Δ, the* Δ-temporal line graph $L_\Delta(\mathcal{G})$ *of $\mathcal{G}$ has vertex set $V(L_\Delta(\mathcal{G})) = \{e_t \mid e \in E \wedge t \in \lambda(e)\}$ and edge set $E(L_\Delta(\mathcal{G})) = \{\{e_t, e'_{t'}\} \mid e \cap e' \neq \emptyset \wedge |t - t'| < \Delta\}$. We say that a graph H is a* temporal line graph *if there is a temporal graph $\mathcal{G}$ and an integer Δ such that $H = L_\Delta(\mathcal{G})$.*

By definition, Δ-temporal line graphs have the following property.

▶ **Observation 4.** *Let $\mathcal{G}$ be a temporal graph and let $L_\Delta(\mathcal{G})$ be its Δ-temporal line graph. The cardinality of a maximum independent set in $L_\Delta(\mathcal{G})$ equals the size of a maximum Δ-temporal matching of $\mathcal{G}$.*

It follows that solving Temporal Matching on a temporal graph $\mathcal{G}$ is equivalent to solving Independent Set on $L_\Delta(\mathcal{G})$.

3 Hardness Results

In this section we show that Maximum Temporal Matching is APX-hard and that Temporal Matching is NP-complete when the underlying graph is a path.

3.1 APX-completeness of Maximum Temporal Matching

In this subsection, we look at Maximum Temporal Matching where we want to maximize the cardinality of the temporal matching. We prove that Maximum Temporal Matching is APX-complete even if $\Delta = 2$ and $T = 3$. For this we provide a so-called *L-reduction* [4] from

the APX-complete Maximum Independent Set problem on cubic graphs [2] to Maximum Temporal Matching. Together with the constant-factor approximation algorithm that we present in Section 4.1 this implies APX-completeness for Maximum Temporal Matching. The reduction also implies NP-completeness of Temporal Matching. Formally, we show the following result.

▶ **Theorem 5** (⋆). *Temporal Matching is NP-complete and Maximum Temporal Matching is APX-complete even if $\Delta = 2$, $T = 3$, and every edge of the underlying graph appears only once. Furthermore, for any $\delta \geq \frac{664}{665}$, there is no polynomial-time δ-approximation algorithm for Maximum Temporal Matching, unless P = NP, and Temporal Matching does not admit a $2^{o(k)} \cdot |\mathcal{G}|^{f(T)}$-time algorithm for any function f, unless the Exponential Time Hypothesis fails.*

We provide the following construction for a reduction from Maximum Independent Set on cubic graphs. It is easy to check that it uses only three time steps and every edge appears in exactly one time step.

▶ **Construction 1.** Let $G = (V, E)$ be an n-vertex cubic graph. We construct in polynomial time a corresponding temporal graph (H, λ) of lifetime three as follows. First, we find a proper 4-edge coloring $c : E \to \{1, 2, 3, 4\}$ of G. Such a coloring exists by Vizing's theorem and can be found in $O(|E|)$ time [21]. Now the underlying graph $H = (U, F)$ contains two vertices v_0 and v_1 for every vertex v of G, and one vertex w_e for every edge e of G. The set F of the edges of H contains $\{v_0, v_1\}$ for every $v \in V$, and for every edge $e = \{u, v\} \in E$ it contains $\{w_e, u_\alpha\}, \{w_e, v_\alpha\}$, where $c(e) \equiv \alpha \pmod 2$. In temporal graph (H, λ) every edge of the underlying graph appears in exactly one of the three time slots:

1. $\lambda(\{w_e, u_\alpha\}) = \lambda(\{w_e, v_\alpha\}) = 1$, where $c(e) \equiv \alpha \pmod 2$, for every edge $e = \{u, v\} \in E$ such that $c(e) \in \{1, 2\}$;
2. $\lambda(\{v_0, v_1\}) = 2$ for every $v \in V$;
3. $\lambda(\{w_e, u_\alpha\}) = \lambda(\{w_e, v_\alpha\}) = 3$, where $c(e) \equiv \alpha \pmod 2$, for every edge $e = \{u, v\} \in E$ such that $c(e) \in \{3, 4\}$.

It is easy to check that the reduction also implies NP-completeness of Temporal Matching. The full proof of Theorem 5 can be found in the full version of this paper.

▶ **Observation 6** (⋆). *Temporal Matching is NP-complete, even if $\Delta = 2$, $T = 5$, and the underlying graph of the input temporal graph is complete.*

The importance of this observation is due to the following parameterized complexity implication. Parameterizing Temporal Matching by structural graph parameters of the underlying graph that are constant on complete graphs cannot yield fixed-parameter tractability unless P = NP, even if combined with the lifetime T. Note that many structural parameters fall into this category, such as domination number, distance to cluster graph, clique cover number, etc. We discuss how our reduction can be adapted to the model of Baste et al. [6] in the full version of this paper.

3.2 NP-completeness of Temporal Matching with Underlying Paths

In this subsection we show NP-completeness of Temporal Matching even for a very restricted class of temporal graphs.

▶ **Theorem 7.** *Temporal Matching is NP-complete even if $\Delta = 2$ and the underlying graph of the input temporal graph is a path.*

We show this result by a reduction from Independent Set on connected cubic planar graphs, which is known to be NP-complete [11]. More specifically, we show that Independent Set is NP-complete on the temporal line graphs of temporal graphs that have a path as underlying graph. Recall that by Observation 4, solving Independent Set on a temporal line graph is equivalent to solving Temporal Matching on the corresponding temporal graph. We proceed as follows.

1. We show that 2-temporal line graphs of temporal graphs that have a path as underlying graph have a grid-like structure. More specifically, we show that they are induced subgraphs of so-called *diagonal grid graphs* or *king's graphs*.
2. We show that Independent Set is NP-complete on induced subgraphs of diagonal grid graphs which together with Observation 4 yields Theorem 7. More specifically:
   - We exploit that cubic planar graphs are induced topological minors of grid graphs and extend this result by showing that they are also induced topological minors of diagonal grid graphs.
   - We show how to modify the subdivision of a cubic planar graph that is an induced subgraph of a diagonal grid graph such that NP-hardness of finding independent sets of certain size is preserved.

▶ **Definition 8** (Diagonal Grid Graph). *A* diagonal grid graph $\widehat{Z}_{n,m}$ *has a vertex* $v_{i,j}$ *for all* $i \in [n]$ *and* $j \in [m]$ *and there is an edge* $\{v_{i,j}, v_{i',j'}\}$ *if and only if* $|i-i'|^2 + |j-j'|^2 \leq 2$.

It is easy to check that for a temporal graph with a path as underlying graph and where each edge is active at every time step, the 2-temporal line graph is a diagonal grid graph.

▶ **Observation 9.** *Let* $\mathcal{G} = (P_n, \lambda)$ *with* $\lambda(e) = [T]$ *for all* $e \in E(P_n)$*, then* $L_2(\mathcal{G}) = \widehat{Z}_{n-1,T}$.

Further, it is easy to see that deactivating an edge at a certain point in time results in removing the corresponding vertex from the diagonal grid graph. See Figure 1 for an example. Hence, we have that every induced subgraph of a diagonal grid graph is a 2-temporal line graph.

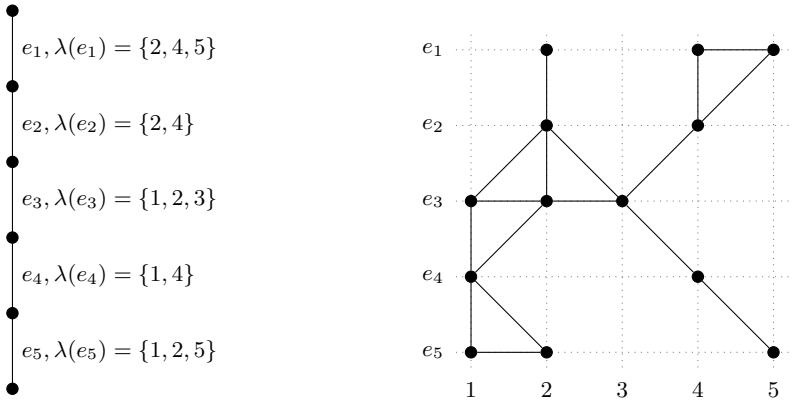
▶ **Corollary 10.** *Let* Z' *be a connected* induced *subgraph of* $\widehat{Z}_{n-1,T}$*. Then there is a* λ *and an* $n' \leq n$ *such that* $Z' = L_2((P_{n'}, \lambda))$.

Having these results at hand, it suffices to show that Independent Set is NP-complete on induced subgraphs of diagonal grid graphs. By Observation 4, this directly implies that Temporal Matching is NP-complete on temporal graphs that have a path as underlying graph. Hence, in the remainder of this section, we discuss the following result.

▶ **Theorem 11** (⋆). *Independent Set on induced subgraphs of diagonal grid graphs is NP-complete.*

This result may be of independent interest and strengthens a result by Clark et al. [9], who showed that Independent Set is NP-complete on unit disk graphs. It is easy to see from Definition 8 that diagonal grid graphs and their induced subgraphs are a (proper) subclass of unit disk graphs.

In the following, we give the main ideas of how we prove Theorem 11. The first building block for the reduction is the fact that we can embed cubic planar graphs into a grid [22]. More specifically, a cubic planar graph admits a planar embedding in such a way that the vertices are mapped to points of a grid and the edges are drawn along the grid lines. Moreover, such an embedding can be computed in polynomial time and the size of the grid is polynomially bounded in the size of the planar graph.



(a) Temporal graph $\mathcal{G} = (P_6, \lambda)$ with λ as visualized.

(b) 2-Temporal line graph $L_2(\mathcal{G})$.

Figure 1 A temporal line graph with a path as underlying graph where edges are *not* always active and its 2-temporal line graph.

Note that if we replace the edges of the original planar graph by paths of appropriate length, then the embedding in the grid is actually a subgraph of the grid. Furthermore, if we scale the embedding by a factor of two, i.e. subdivide every edge once, then the embedding is also guaranteed to be an *induced* subgraph of the grid. In other words, we argue that every cubic planar graph is an induced topological minor of a polynomially large grid graph. We then show how to modify the embedding in a way that insures that the resulting graph is also an induced topological minor of an polynomially large *diagonal* grid graph. The last step is to further modify the embedding such that it can be obtained from the original graph by subdividing each edge an even number of times, this ensures that NP-hardness of Independent Set is preserved.

It is easy to check that Theorem 11, Observation 4, and Corollary 10 together imply Theorem 7. Theorem 7 also has some interesting implications from the point of view of parameterized complexity: Parameterizing Temporal Matching by structural graph parameters of the underlying graph that are constant on a path cannot yield fixed-parameter tractability unless P = NP, even if combined with Δ. Note that a large number of popular structural parameters fall into this category, such as maximum degree, treewidth, pathwidth, feedback vertex number, etc.

4 Algorithms

Here, we show one approximation and two exact algorithms for Temporal Matching.

4.1 Approximation of Maximum Temporal Matching

In this section, we present a $\frac{\Delta}{2\Delta-1}$-approximation algorithm for Maximum Temporal Matching. Note that for $\Delta = 2$ this is a $\frac{2}{3}$-approximation, while for arbitrary constant Δ this is a $(\frac{1}{2} + \varepsilon)$-approximation, where $\varepsilon = \frac{1}{2(2\Delta-1)}$ is a constant, too. Specifically, we show the following.

▶ **Theorem 12** (⋆). *Maximum Temporal Matching admits an $O\left(Tm(\sqrt{n} + \Delta)\right)$-time $\frac{\Delta}{2\Delta-1}$-approximation algorithm.*

Algorithm 4.1 $\frac{\Delta}{2\Delta-1}$-Approximation Algorithm (Theorem 12).

```
M ← ∅.
foreach Δ-template 𝒮 do
    Compute a Δ-temporal matching M^𝒮 with respect to 𝒮.
    if |M^𝒮| > |M| then M ← M^𝒮.
return M.
```

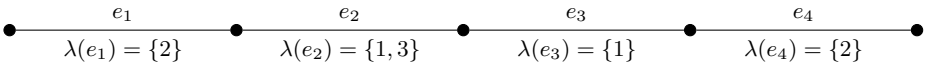


Figure 2 A temporal graph witnessing that the analysis of Algorithm 4.1 is tight for $\Delta = 2$.

The main idea of our approximation algorithm is to compute maximum matchings for slices of size Δ of the input temporal graph that are sufficiently far apart from each other such that they do not interfere with each other, and hence are computable in polynomial time. Then we greedily fill up the gaps. We try out certain combinations of non-interfering slices of size Δ in a systematic way and then take the largest Δ-matching that was found in this way. With some counting arguments we can show that this achieves the desired approximation ratio. In the following we describe and prove this claim formally.

We first introduce some additional notation and terminology. Recall that $\mu_\Delta(\mathcal{G})$ denotes the size of a maximum Δ-temporal matching in $\mathcal{G}$. Let Δ and T be fixed natural numbers such that $\Delta \leq T$. For every time slot $t \in [T - \Delta + 1]$, we define the Δ-*window* W_t as the interval $[t, t + \Delta - 1]$ of length Δ. We use this to formalize slices of size Δ of a temporal graph. An interval of length at most $\Delta - 1$ that either starts at slot 1, or ends at slot T is called a *partial* Δ-*window (with respect to lifetime* T*)*. For the sake of brevity, we write *partial* Δ-*window*, when the lifetime T is clear from the context. The *distance* between two disjoint intervals $[a_1, b_1]$ and $[a_2, b_2]$ with $b_1 < a_2$ is $a_2 - b_1 - 1$.

A Δ-*template (with respect to lifetime* T*)* is a maximal family $\mathcal{S}$ of Δ-windows or partial Δ-windows in the interval $[T]$ such that any two consecutive elements in $\mathcal{S}$ are at distance exactly $\Delta - 1$ from each other. Let $\mathcal{S}$ be a Δ-template. A Δ-temporal matching $M^{\mathcal{S}}$ in $\mathcal{G} = (G, \lambda)$ is called a Δ-temporal matching *with respect to* Δ-*template* $\mathcal{S}$ if $M^{\mathcal{S}}$ has the maximum possible number of edges in every interval $W \in \mathcal{S}$, i.e. $\left|M^{\mathcal{S}}|_W\right| = \mu_\Delta(\mathcal{G}|_W)$ for every $W \in \mathcal{S}$.

Now we are ready to present and analyze our $\frac{\Delta}{2\Delta-1}$-approximation algorithm, see Algorithm 4.1. The idea of the algorithm is simple: for every Δ-template $\mathcal{S}$ compute a Δ-temporal matching $M^{\mathcal{S}}$ with respect to $\mathcal{S}$ and among all of the computed Δ-temporal matchings return a matching of the maximum cardinality.

We remark that our analysis ignores the fact that the algorithm may add time-edges from the gaps between the Δ-windows defined by the template to the matching if they are not in conflict with any other edge in the matching. Hence, on the one hand, there is potential room for improvement. On the other hand, our analysis of the approximation factor of Algorithm 4.1 is tight for $\Delta = 2$. Namely, there exists a temporal graph $\mathcal{G}$ (see Figure 2) such that on the instance $(\mathcal{G}, 2)$ our algorithm (in the worst case) finds a 2-temporal matching of size two, while the size of a maximum 2-temporal matching in $\mathcal{G}$ is three. In this example any improvement of the algorithm that utilizes the gaps between the Δ-windows would not lead to a better performance.

4.2 Fixed-parameter tractability for the parameter solution size

In this section we provide a fixed-parameter algorithm for Temporal Matching parameterized by the solution size k. More specifically, we provide a linear-time algorithm for a fixed solution size k. Formally, the main result of this subsection is to show the following.

▶ **Theorem 13** ($\star$). *There is a linear-time FPT-algorithm for Temporal Matching parameterized by the solution size k.*

We discuss the proof Theorem 13 in the remainder of this section. Recall that due to Baste et al. [6] it is already known that Temporal Matching is fixed-parameter tractable when parameterized by the solution size k *and* Δ. In comparison to the algorithm of Baste et al. [6] the running time of our algorithm is independent of Δ, hence improving their result from a parameterized classification standpoint.

The rough idea of our algorithm is the following. We develop a preprocessing procedure that reduces the number of time-edges of the first Δ-window. After applying this procedure, the number of time-edges in the first Δ-window is upper-bounded in a function of the solution size parameter k. This allows us to enumerate all possibilities to select time-edges from the first Δ-window for the temporal matching. Then, for each possibility, we can remove the first Δ-window from the temporal graph and solve the remaining part recursively.

Next, we describe the preprocessing procedure more precisely. Referring to kernelization algorithms, we call this procedure *kernel for the first Δ-window*. If we count naively the number of Δ-temporal matchings in the first Δ-window of a temporal graph, then this number clearly depends on Δ. This is too large for Theorem 13. A key observation to overcome this obstacle is that if we look at an edge appearance of a Δ-temporal matching which comes from the first Δ-window, then we can exchange it with the first appearance of the edge.

▶ **Lemma 14** ($\star$). *Let (G, λ) be a temporal graph and let M be a Δ-temporal matching in (G, λ). Let also $e \in E_{t_1} \cap E_{t_2}$, where $t_1 < t_2 \leq \Delta$. If $(e, t_1) \notin M$ and $(e, t_2) \in M$, then $M' = (M \setminus \{(e, t_2)\}) \cup \{(e, t_1)\}$ is a Δ-temporal matching in (G, λ).*

We use Lemma 14 to construct a small set K of time-edges from the first Δ-window such that there exists a maximum Δ-temporal matching M in (G, λ) with the property that the restriction of M to the first Δ-window is contained in K.

▶ **Definition 15** (Kernel for the First Δ-Window). *Let Δ be a natural number and let $\mathcal{G}$ be a temporal graph. We call a set K of time-edges of $\mathcal{G}|_{[1,\Delta]}$* a kernel for the first Δ-window of $\mathcal{G}$ *if there exists a maximum Δ-temporal matching M in $\mathcal{G}$ with $M|_{[1,\Delta]} \subseteq K$.*

Informally, the idea for computing the kernel for the first Δ-window is to first select vertices that are suitable to be matched. Then, for each of these vertices, we select the earliest appearance of a sufficiently large number of incident time-edges, where each of these time-edges corresponds to a different edge of the underlying graph. We show that we can do this in a such way that the number of selected time-edges can be upper-bounded in a function of the size ν of a maximum matching of the underlying graph G. Formally, we aim at proving the following lemma.

▶ **Lemma 16** ($\star$). *Given a natural number Δ and a temporal graph $\mathcal{G} = (G, \lambda)$ we can compute in $O(\nu^2 \cdot |\mathcal{G}|)$ time a kernel K for the first Δ-window of $\mathcal{G}$ such that $|K| \in O(\nu^2)$.*

Algorithm 4.2 presents the pseudocode for the algorithm behind Lemma 16. We show correctness of Algorithm 4.2 in Lemma 17 and examine its running time in Lemma 18. Hence, Lemma 16 follows from Lemmas 17 and 18.

Algorithm 4.2 Kernel for the First Δ-Window (Lemma 16).

1 Let G' be the underlying graph of $\mathcal{G}|_{[1,\Delta]}$ and $K = \emptyset$.
2 $A \leftarrow$ a maximum matching of G'.
3 $V_A \leftarrow$ the set of vertices matched by A.
4 **foreach** $v \in V_A$ **do**
5 $\quad R_v \leftarrow \big\{(\{v,w\},t) \mid w \in N_{G'}(v) \text{ and } t = \min\{i \in [\Delta] \mid \{v,w\} \in E_i\}\big\}$.
6 $\quad$ **if** $|R_v| \leq 4\nu$ **then** $K \leftarrow K \cup R_v$.
7 $\quad$ **else**
8 $\quad\quad$ Form a subset $R' \subseteq R_v$ such that $|R'| = 4\nu + 1$ and for every $(e,t) \in R'$ and $(e',t') \in R_v \setminus R'$ we have $t \leq t'$.
9 $\quad\quad K \leftarrow K \cup R'$.
10 **return** K.

▶ **Lemma 17.** *Algorithm 4.2 is correct, that is, the algorithm outputs a size-$O(\nu^2)$ kernel K for the first Δ-window of $\mathcal{G}$.*

Proof. Let M be a maximum Δ-temporal matching of $\mathcal{G}$ such that $\big|M|_{[1,\Delta]} \setminus K\big|$ is minimized. Without loss of generality we can assume that every time-edge in $M|_{[1,\Delta]}$ is the first appearance of an edge. Indeed, by construction, K contains only the first appearances of edges, and therefore if $(e,t) \in M|_{[1,\Delta]}$ is not the first appearance of e, by Lemma 14 it can be replaced by the first appearance, and this would not increase $\big|M|_{[1,\Delta]} \setminus K\big|$. Now, assume towards a contradiction that $M|_{[1,\Delta]} \setminus K$ is not empty and let (e,t) be a time-edge in $M|_{[1,\Delta]} \setminus K$. Since A is a maximum matching in the underlying graph G' of $\mathcal{G}|_{[1,\Delta]}$, at least one of the end vertices of e is matched by A, i.e., it belongs to V_A. Then for a vertex $v \in V_A \cap e$ we have that $(e,t) \in R_v$. Moreover, observe that $|R_v| > 4\nu$, because otherwise (e,t) would be in K. For the same reason $(e,t) \notin R'$, where $R' \subseteq R_v$ is the set of time-edges computed in Line 8 of the algorithm. Let $W = \{(w,t) \mid (\{v,w\},t) \in R'\}$ be the set of vertex appearances which are adjacent to vertex appearance (v,t) by a time-edge in R'. Since R_v contains only the first appearances of edges, we know that W contains exactly $4\nu + 1$ vertex appearances of pairwise different vertices.

We now claim that W contains a vertex appearance which is not Δ-blocked by any time-edge in M. To see this, we recall that ν is the maximum matching size of the underlying graph of $\mathcal{G}$. Hence it is also an upper bound on the number of time-edges in $M|_{[1,\Delta]}$ and $M|_{[\Delta+1,2\Delta]}$, which implies that in the first Δ-window vertex appearances of at most 4ν distinct vertices are Δ-blocked by time-edges in M. Since W contains $4\nu + 1$ vertex appearances of pairwise different vertices, we conclude that there exists a vertex appearance $(w',t') \in W$ which is not Δ-blocked by M.

Observe that $t' \leq t$ because $(\{v,w'\},t') \in R'$ and $(e,t) \in R_v \setminus R'$. Hence, (v,t') is not Δ-blocked by $M \setminus \{(e,t)\}$. Thus, $M^* := (M \setminus \{(e,t)\}) \cup \{(\{v,w'\},t')\}$ is a Δ-temporal matching of size $|M|$ with $\big|M^*|_{[1,\Delta]} \setminus K\big| < \big|M|_{[1,\Delta]} \setminus K\big|$. This contradiction implies that $M|_{[1,\Delta]} \setminus K$ is empty and thus $M|_{[1,\Delta]} \subseteq K$.

It remains to show that $|K| \in O(\nu^2)$. Since each maximum matching in G' has at most ν edges, we have that $|V_A| \leq 2\nu$. For each vertex in V_A the algorithm adds at most $4\nu + 1$ time-edges to K. Thus, $|K| \leq 2\nu \cdot (4\nu + 1) \in O(\nu^2)$. ◀

▶ **Lemma 18** ($\star$). *Algorithm 4.2 runs in $O(\nu^2(n + m\Delta))$ time. In particular, the time complexity of Algorithm 4.2 is dominated by $O(\nu^2|\mathcal{G}|)$.*

Having Algorithm 4.2 at hand, we can formulate a recursive search tree algorithm which (1) picks a Δ-temporal matchings M in the kernel of the first Δ-window, (2) removes the first Δ-window from the temporal graph, (3) removes all time-edges which are not Δ-independent with M, and (4) calls itself until the temporal graph in empty. For pseudocode of this algorithm and the proof of correctness, we refer to the full version of this paper.

4.3 Fixed-parameter tractability for the combined parameter Δ and maximum matching size ν of the underlying graph

In this section we show that Temporal Matching is fixed-parameter tractable when parameterized by Δ and the maximum matching size ν of the underlying graph.

▶ **Theorem 19** ($\star$). *Temporal Matching can be solved in* $2^{O(\nu\Delta)} \cdot |\mathcal{G}| \cdot \frac{T}{\Delta}$ *time.*

Note that Theorem 19 implies that Temporal Matching is fixed-parameter tractable when parameterized by Δ and the maximum matching size ν of the underlying graph, because there is a simple preprocessing step so that we can assume afterwards that the lifetime T is polynomially upper-bounded in the input size. This preprocessing step modifies the temporal graph such that it does not contain Δ consecutive edgeless snapshots. This can be done by iterating once over the temporal graph. Observe that this procedure does not change the maximum size of a Δ-temporal matching and afterwards each Δ-window contains at least one time-edge. Hence, $\frac{T}{\Delta} \leq |\mathcal{G}|$.

Note that this result is incomparable to Theorem 13. In some sense, we trade off replacing the solution size parameter k with the structurally smaller parameter ν but we do not know how to do this without combining it with Δ. In comparison to the exact algorithm by Baste et al. [6] (who showed fixed-parameter tractability with k and Δ) we replace k by the structurally smaller ν, hence improving their result from a parameterized classification standpoint. Furthermore, we note that Theorem 19 is asymptotically optimal for any fixed Δ since there is no $2^{o(\nu)} \cdot |\mathcal{G}|^{f(\Delta,T)}$ algorithm for Temporal Matching, unless ETH fails (see Theorem 5).

In the reminder of this section, we sketch the main ideas of the algorithm behind Theorem 19. The algorithm works in three major steps:

1. The temporal graph is divided into disjoint Δ-windows,
2. for each of these Δ-windows a small family of Δ-temporal matchings is computed, and then
3. the maximum size of a Δ-temporal matching for the whole temporal graph is computed with a dynamic program based on the families from (Step 2).

We first discuss how the algorithm performs Step 2. Afterwards we formulate the dynamic program (Step 3). In a nutshell, Step 2 consists of an iterative computation of a small (upper-bounded in $\Delta + \nu$) family of Δ-temporal matchings for an arbitrary Δ-window such that at least one of them is "extendable" to a maximum Δ-temporal matching for the whole temporal graph.

Families of ℓ-complete Δ-temporal matchings. Throughout this section let $\mathcal{G} = (G = (V, E), \lambda)$ be a temporal graph of lifetime T and let ν be the maximum matching size in G. Let also Δ and ℓ be natural numbers such that $\ell\Delta \leq T$.

A family $\mathcal{M}$ of Δ-temporal matchings of $\mathcal{G}|_{[\Delta(\ell-1)+1,\Delta\ell]}$ is called *ℓ-complete* if for any Δ-temporal matching M of $\mathcal{G}$ there is $M' \in \mathcal{M}$ such that $\big(M \setminus M|_{[\Delta(\ell-1)+1,\Delta\ell]}\big) \cup M'$ is a Δ-temporal matching of $\mathcal{G}$ of size at least $|M|$. A central part of our algorithm is an efficient procedure for computing an ℓ-complete family. Formally, we aim for the following lemma.

▶ **Lemma 20** (⋆). *There exists a $2^{O(\nu\Delta)} \cdot |\mathcal{G}|$-time algorithm that computes an ℓ-complete family of size $2^{O(\nu\Delta)}$ of Δ-temporal matchings of $\mathcal{G}|_{[\Delta(\ell-1)+1,\Delta\ell]}$.*

In the proof of Lemma 20 we employ representative families and other tools from matroid theory [8, 10].

Dynamic program. Now we are ready to combine Step 2 of our algorithm with the remaining Steps 1 and 3. More precisely, we employ ℓ-complete families of Δ-temporal matchings of Δ-windows in a dynamic program (Step 3) to compute the Δ-temporal matching of maximum size for the whole temporal graph. The pseudocode of this dynamic program and its proof of correctness is stated in the full version of this paper. This is the algorithm behind Theorem 19. It computes a table $\mathcal{T}$ where each entry $\mathcal{T}[i, M']$ stores the maximum size of a Δ-temporal matching M in the temporal graph $\mathcal{G}|_{[1,\Delta i]}$ such that all the time-edges in $M|_{[\Delta(i-1)+1,\Delta i]} = M'$. Observe that a trivial dynamic program which computes all entries of $\mathcal{T}$ cannot provide fixed-parameter tractability of Temporal Matching when parameterized by Δ and ν, because the corresponding table is simply too large. The crucial point of the dynamic program is that it is sufficient to fix for each $i \in [\frac{T}{\Delta}]$ an i-complete family $\mathcal{M}_i$ of Δ-temporal matchings for $\mathcal{G}|_{[\Delta(i-1)+1,\Delta i]}$ and then compute only the entries $\mathcal{T}[i, M']$, where $M' \in \mathcal{M}_i$.

Kernelization lower bound. Lastly, we can show that we cannot hope to obtain a polynomial kernel for the parameter combination number n of vertices and Δ. In particular, this implies that, presumably, we also cannot get a polynomial kernel for the parameter combination ν and Δ, since $\nu \leq \frac{n}{2}$.

▶ **Proposition 21** (⋆). *Temporal Matching parameterized by the number n of vertices does not admit a polynomial kernel for all $\Delta \geq 2$, unless NP $\subseteq$ coNP/poly.*

5 Conclusion

The following issues remain research challenges. First, on the side of polynomial-time approximability, improving the constant approximation factors is desirable and seems feasible. Beyond, lifting polynomial time to FPT time, even approximation schemes in principle seem possible, thus circumventing our APX-hardness result. Taking the view of parameterized complexity analysis in order to cope with NP-hardness, a number of directions are naturally coming up. For instance, based on our fixed-parameter tractability result for the parameter solution size, the following questions naturally arise:

1. Is there a polynomial-size kernel for the solution size parameter k?
2. Is there a faster algorithm or a matching lower-bound for the running time of Theorem 13?

To enlarge the range of promising and relevant parameterizations, one may extend the parameterized studies to structural graph parameters combined with Δ or the lifetime of the temporal graph. In particular, treedepth combined with Δ is left open, since it is a "stronger" parameterization than in Theorem 19 but has an unbounded value in all known NP-hardness reductions.

References

1 E. C. Akrida, G. B. Mertzios, P. G. Spirakis, and V. Zamaraev. Temporal vertex cover with a sliding time window. *J. Comput. Syst. Sci.*, 107:108–123, 2020.

2 P. Alimonti and V. Kann. Some APX-completeness results for cubic graphs. *Theor. Comput. Sci.*, 237(1-2):123–134, 2000.

3 J. Aronson, M. Dyer, A. Frieze, and S. Suen. Randomized greedy matching ii. *Random Struct. Algorithms*, 6(1):55–73, 1995.

4 G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 2012.

5 E. Bampis, B. Escoffier, M. Lampis, and V. Th. Paschos. Multistage matchings. In *Proc. of 16th SWAT*, volume 101 of *LIPIcs*, pages 7:1–7:13. Schloss Dagstuhl - LZI, 2018.

6 J. Baste, B. Bui-Xuan, and A. Roux. Temporal matching. *Theor. Comput. Sci.*, 806:184–196, 2020.

7 M. Bentert, A-S. Himmel, H. Molter, M. Morik, R. Niedermeier, and R. Saitenmacher. Listing all maximal k-plexes in temporal graphs. *ACM J. Exp. Algorithmics*, 24(1):1–13, 2019.

8 R. van Bevern, O. Y. Tsidulko, and P. Zschoche. Fixed-parameter algorithms for maximum-profit facility location under matroid constraints. In *Proc. of 11th CIAC*, volume 11485 of *LNCS*, pages 62–74. Springer, 2019.

9 B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Math.*, 86(1-3):165–177, 1990.

10 F. V. Fomin, D. Lokshtanov, F. Panolan, and S. Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4):29:1–29:60, 2016.

11 M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM J. Appl. Math.*, 32(4):826–834, 1977.

12 A. Gupta, K. Talwar, and U. Wieder. Changing bases: Multistage optimization for matroids and matchings. In *Proc. of 41st ICALP*, volume 8572 of *LNCS*, pages 563–575. Springer, 2014.

13 D. Hausmann and B. Korte. k-greedy algorithms for independence systems. *Oper. Res.*, 22(1):219–228, 1978.

14 A-S. Himmel, H. Molter, R. Niedermeier, and M. Sorge. Adapting the bron-kerbosch algorithm for enumerating maximal cliques in temporal graphs. *Soc. Netw. Anal. Min.*, 7(1):35:1–35:16, 2017.

15 P. Holme and J. Saramäki. Temporal networks. *Physics Reports*, 519(3):97–125, 2012.

16 D. Kempe, J. Kleinberg, and A. Kumar. Connectivity and inference problems for temporal networks. *J. Comput. Syst. Sci.*, 64(4):820–842, 2002.

17 B. Korte and D. Hausmann. An analysis of the greedy heuristic for independence systems. *Discrete Math.*, 2:65–74, 1978.

18 M. Latapy, T. Viard, and C. Magnien. Stream graphs and link streams for the modeling of interactions over time. *Soc. Netw. Anal. Min.*, 8(1):61, 2018.

19 G. B. Mertzios, H. Molter, and V. Zamaraev. Sliding window temporal graph coloring. In *Proc. of 33rd AAAI*, pages 7667–7674. AAAI Press, 2019.

20 M. Poloczek and M. Szegedy. Randomized greedy algorithms for the maximum matching problem with new analysis. In *Proc. of 53rd FOCS*, pages 708–717. IEEE, 2012.

21 A. Schrijver. Bipartite edge coloring in $O(\Delta m)$ time. *SIAM J. Comput.*, 28(3):841–846, 1998.

22 L. G. Valiant. Universality considerations in VLSI circuits. *IEEE Trans. Comput.*, 100(2):135–140, 1981.

23 H. Wu, J. Cheng, Y. Ke, S. Huang, Y. Huang, and H. Wu. Efficient algorithms for temporal path computation. *IEEE Trans. Knowl. Data. Eng.*, 28(11):2927–2942, 2016.

24 P. Zschoche, T. Fluschnik, H. Molter, and R. Niedermeier. The complexity of finding small separators in temporal graphs. *J. Comput. Syst. Sci.*, 107:72–92, 2020.

Tight Bounds for the Cover Times of Random Walks with Heterogeneous Step Lengths

Brieuc Guinard
IRIF, CNRS,
University of Paris, France
guinard@irif.fr

Amos Korman
IRIF, CNRS,
University of Paris, France
https://amoskorman.com/
amos.korman@irif.fr

Abstract

Search patterns of randomly oriented steps of different lengths have been observed on all scales of the biological world, ranging from microscopic to the ecological, including in protein motors, bacteria, T-cells, honeybees, marine predators, and more, see e.g., [21, 22, 31, 33, 34, 35, 36]. Through different models, it has been demonstrated that adopting a variety in the magnitude of the step lengths can greatly improve the search efficiency. However, the precise connection between the search efficiency and the number of step lengths in the repertoire of the searcher has not been identified.

Motivated by biological examples in one-dimensional terrains, a recent paper studied the best cover time on an n-node cycle that can be achieved by a random walk process that uses k step lengths [7]. By tuning the lengths and corresponding probabilities the authors therein showed that the best cover time is roughly $n^{1+\Theta(1/k)}$. While this bound is useful for large values of k, it is hardly informative for small k values, which are of interest in biology [2, 4, 25, 30]. In this paper, we provide a tight bound for the cover time of such a walk, for every integer $k > 1$. Specifically, up to lower order polylogarithmic factors, the cover time is $n^{1+\frac{1}{2k-1}}$. For $k = 2, 3, 4$ and 5 the bound is thus $n^{4/3}$, $n^{6/5}$, $n^{8/7}$, and $n^{10/9}$, respectively. Informally, our result implies that, as long as the number of step lengths k is not too large, incorporating an additional step length to the repertoire of the process enables to improve the cover time by a polynomial factor, but the extent of the improvement gradually decreases with k.

2012 ACM Subject Classification Theory of computation → Random walks and Markov chains; Applied computing → Computational biology

Keywords and phrases Computational Biology, Randomness in Computing, Search Algorithms, Random Walks, Lévy Flights, Intermittent Search, CCRW

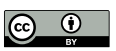
Digital Object Identifier 10.4230/LIPIcs.STACS.2020.28

Related Version A full version of this paper is available at https://hal.archives-ouvertes.fr/hal-02303873v1.

Funding This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 648032).

1 Introduction

This paper follows the "Natural Algorithms" line of research, aiming to contribute to biological studies from an algorithmic perspective [6, 10, 17, 26]. In particular, we follow a similar approach to Chazelle [10, 11], considering a process that has been extensively studied by physicists and biologists, and offering a more uniform algorithmic analysis based on techniques from probability theory. Our subject of interest is random walks with heterogeneous step



lengths, a family of processes that during the last two decades has become a central model for biological movement, see e.g., [12, 21, 22, 25, 27, 29, 31, 32, 34, 35, 36]. Our approach is to quantify by how much can the search efficiency improve when the searcher is allowed to use more steps. Specifically, our goal is to analyze, for every integer k, the best cover time achievable by a random walk that utilizes k step-lengths, and identify the parameters that achieve the optimal cover time. Hence, in some sense, we view the number of steps as a "hardware" constraint on the searcher, and ask what is the best "software" to utilize them, that is, the best way to set the lengths, and the probabilities of taking the corresponding steps. We focus on the one-dimensional terrain (an n-node cycle) as it is both biologically relevant, and, among other Euclidean spaces, it is the most sensitive to step-length variations (e.g., the simple random walk on the two-dimensional plain already enjoys a quasi-linear cover time). A preliminary investigation of this question was recently done by the authors of the current paper together with collaborating researchers [7], yielding asymptotic bounds with respect to k. Unfortunately, these bounds are not very informative for small values of k, which are of particular interest in biology [2, 4, 25, 30]. For example, for processes that can use a small number of step-lengths, say $k = 2$ or $k = 3$, the bound in [7] merely says that the cover time is polynomial in n, which does not even imply that such a process can outperform the simple random walk – whose cover time is known to be $\Theta(n^2)$. In this paper we improve both the lower bound and the upper bound in [7], identifying the tight cover time for every integer k.

1.1 Background and Motivation

The exploration-exploitation dilemma is fundamental to almost all search or foraging processes in biology [19]. An efficient search strategy needs to strike a proper balance between the need to explore new areas and the need to exploit the more promising ones found. At an intuitive level, this is often perceived as a tradeoff between two scales: the global scale of exploration and the local scale of exploitation. This paper studies the benefits of incorporating a hierarchy of multiple scales, where lower scales serve to exploit the exploration made by higher scales. We demonstrate this concept by focusing on random walk search patterns with heterogeneous step lengths, viewing the usage of steps of a given length as searching on a particular scale.

In the last two decades, random walks with heterogeneous step lengths have been used by biologists and physicists to model biological processes across scales, from microscopic to macroscopic, including in DNA binding proteins [5, 14], immune cells [18], crawling amoeba [33], locomotion mode in mussels [15, 22], snails [31], marine predators [21, 34], albatrosses [35, 36], and even in humans [9, 32, 29]. Most of these biological examples concern search contexts, e.g., searching for pathogens or food. Indeed, from a search efficiency perspective, it has been argued that the heterogeneity of step lengths in such processes allows to reduce oversample, effectively improving the balance between global exploration and local exploitation [4, 36]. However, the precise connection between the search efficiency and the number of step lengths in the repertoire of the searcher has not been identified.

Due to possible cognitive conflicts between motion and perception, in some of the aforementioned search contexts it was argued that biological entities are essentially unable to detect targets while moving fast, and hence targets are effectively found only between jumps, see e.g., [4, 25] and the references therein. Those models are often called *intermittent*. When the search is intermittent, we say that a site is *visited* whenever the searcher completes a jump landing on this site. It is also typically assumed that the searcher has some radius of visibility r, and a target can only be detected if it is in the r-vicinity of a site currently

visited by the searcher. Discretizing the space, one may view a Euclidian space as a grid of the appropriate dimension, in which each edge is of length r. In this discretization, sites are nodes, and the searcher can detect a target at a node, only if it makes a random jump that lands on it.

In general, two families of processes with heterogeneous step lengths have been extensively studied in Euclidean spaces: *Lévy Flights* (named after the mathematician Paul Lévy), and *Composite Correlated Random Walks (CCRW)*, see e.g., [2, 4, 25]. Both have been claimed to be optimal under certain conditions and both have certain empirical support. In the Lévy Flight process, step lengths have a probability distribution that is heavy-tailed: at each step a direction is chosen uniformly at random, and the probability to perform a step of length d is proportional to $d^{-\mu}$, for some fixed parameter $1 < \mu < 3$.

Searchers employing a CCRW can potentially alternate between multiple modes of search[1], but apart for few exceptions [30], such patterns have mostly been studied when assuming that the number of search modes is 2. Specifically, a diffusive phase in which targets can be detected and a ballistic phase in which the searcher moves in a random direction in a straight line whose length is exponentially distributed with some mean L. This CCRW with 2 modes can be approximated as a discrete random walk with two step lengths, hereafter called *2-scales search*: first, choose a direction uniformly at random. Then, with some probability p take a step of unit length, and otherwise, with probability $1 - p$, take a step of some predetermined length L.

Lévy Flights and 2-Scales searches have been studied extensively using differential equation techniques and computer simulations. These studies aimed to both compare the performances of these processes as well as to identify the parameters that maximize the rate of target detection or minimize the hitting time under various target distributions [4, 12, 25, 27, 36].

Most of the literature on the subject has concentrated on either one or two dimensional Euclidian spaces. In particular, the one-dimensional case has attracted attention due to several reasons. First, it finds relevance in several biological contexts, including in the reaction pathway of DNA binding proteins [5, 14]. One-dimension can also serve as an approximation to general narrow and long topologies, which can be found for example in blood veins or other organs. Second, from a computational perspective, the one-dimension is the only dimension where the simple random walk has a large cover time, namely, quadratic, whereas in all higher dimensions the cover time is nearly linear. This implies that in terms of the cover time, heterogeneous random walks can potentially play a much more significant role in one-dimension than in higher dimensions.

1.2 Definitions

We model the one-dimension space as an n-node cycle, termed C_n. For an integer k, we define the random walks process with k step lengths as follows.

▶ **Definition 1** (k-scales search)**.** *A random walk process X is called a* k-scales search *on C_n if there exists a probability distribution $\boldsymbol{p} = (p_i)_{i=0}^{k-1}$, where $\sum_i p_i = 1$, and integers $L_0, L_1, \ldots, L_{k-1}$ such that, on each step, X makes a jump $\{0, -L_i, +L_i\}$ with probability*

[1] CCRW have also been classified as either *cue-sensitive*, i.e., they can change their mode of operation upon detecting a target [3], or *internally-driven*, i.e., their movement pattern depends only on the mechanism internal to the searcher [23]. However, when targets are extremely rare and there is no a-priori knowledge about their distribution, one must cover a large portion of the terrain before finding a target, and hence the aforementioned distinction becomes irrelevant.

respectively $p_i/2, p_i/4, p_i/4$. *Overall, with probability* $1/2$, *the process* X *stays in place*[2]. *The numbers* (p_i) *and* (L_i) *are called the* parameters *of the search process* X. *The speed is assumed to be a unitary constant, that is, it takes* L *time to do a step of length* L.

Our goal is to show upper and lower bounds on the *cover time* of a k-scales search, that is, the expected time to visit every node of the ambient graph C_n, where it is assumed that a jump from some point x to y visits only the endpoint y, and not any of the intermediate nodes. We denote by $\mathbb{E}(t_{cov}(n,k))$ the smallest cover time achievable by a k-scales search over the n-node cycle. The parameters n and k are omitted when clear from the context.

We also define the following k-scales search which is often referred to in the mathematical literature as a *Weierstrassian random walk* [20]. In the biology literature, it has been used as a model for the movement strategy of snails [31] and mussels [30].

▶ **Definition 2** (Weierstrassian random walk). *Let* $b \geq 2$ *and* k *be integers such that* $b^{k-1} < n \leq b^k$. *The* Weierstrassian random walk *with parameter* b *is the* k*-scales search defined by:* $L_i = b^i$ *and* $p_i = c_b b^{-i}$, *for every* $0 \leq i \leq k-1$, *with the normalizing constant* $c_b = \frac{b-1}{b-b^{1-k}}$.

Note that c_b is an increasing function of $b > 1$, and so $c_b \geq c_2 \geq 1/2$ for $b \geq 2$. Hence, $p_0 = c_b \geq 1/2$. Also $p_0 = c_b \leq 1$, hence $c_b = \Theta(1)$ is indeed a constant.

1.3 Previous Bounds on the Cover Time of k-scales search

The work of Lomholt et al. [25] considered intermittent search on the one-dimensional cycle of length n, and compared the performances of the best 2-scales search to the best Lévy Flight. With the best parameters, they showed that the best 2-scales search can find a target in roughly $n^{4/3}$ expected time, but introducing Lévy distributed relocations with exponent μ close to 2 can reduce the search time to quasi linear.

Taking a more unified computational approach, a recent paper [7] analyzed the impact of having k heterogeneous step lengths on the cover time (or hitting time[3]) of the n-node cycle C_n. Specifically, the following bounds were established in [7].

▶ **Theorem** (Upper bound on the cover time of Weierstrassian random walk from [7]). *Let* b, n *be integers such that* $2 \leq b < n$ *and set* $k = \log n / \log b$. *The cover time of the Weierstrassian random walk with parameter* b *on the* n*-cycle is at most* $\mathit{poly}(k) \cdot \mathit{poly}(b) \cdot n \log n$.

Taking $b = \lceil n^{1/k} \rceil$ yields the following corollary.

▶ **Corollary** (Upper bound from [7]). *For any* $k \leq \frac{\log n}{\log \log n}$, *there exists a* k*-scales search with cover time* $n^{1+O(\frac{1}{k})} \log n$.

Note that for small values of k, this bound is not very informative. For example, for $k = 2, 3$ the bound merely says that the cover time is polynomial in n, which is known already for $k = 1$, i.e., the simple random walk, whose cover time is $\Theta(n^2)$.

---

[2] This laziness assumption is used for technical reasons, as is common in many other contexts of random walks. Note that this assumption does not affect the time performance of the process, as we consider it takes time 0 to stay in place.

[3] Note that in connected graphs, the notion of *cover time*, namely the expected time until all sites (of a finite domain) are visited when starting the search from the worst case site, is highly related to the *hitting time*, namely, the expected time to visit a node x starting from node y, taken on the worst case pair x and y; the cover time is always at least the hitting time, and in connected graphs it is at most a logarithmic multiplicative factor more than the hitting time, see [24][Matthews method, Theorem 11.2].

▶ **Theorem** (Lower bound from [7]). *For every $\varepsilon > 0$, there exist sufficiently small constants $c, c' > 0$ such that for $k \leq c' \frac{\log n}{\log \log n}$, any k-scales search cannot achieve a cover time better than $c \cdot n^{1+\frac{1/2-\varepsilon}{k+1}}$.*

The aforementioned lower bound of [7] is more precise than the upper bound, but still not tight, as we show in the next subsection. For example, for $k = 2$, the lower bound in [7] gives $n^{7/6}$ instead of $n^{4/3}$, which is the tight bound.

1.4 Our Results

This paper provides tight bounds for the cover times of k-scales searches, for any integer $k > 1$. Specifically, we prove that the optimal cover (or hitting) time achievable by a k-scales search is $n^{1+\frac{1}{2k-1}}$, up to lower order polylogarithmic factors. Our bound implies that for small k, the improvement in the cover time incurred by employing one more step length is polynomial, but the extent of the improvement gradually decreases with k.

In order to establish the tight bound, we first had to understand what should be a good candidate for the tight bound to aim to. This was not a trivial task, as the precise bound takes an unusual form. After identifying the candidate for the bound, we had to improve both the upper and the lower bounds from [7], which required us to overcome some key technical difficulties. For the lower bound, [7] established that the cover time is bounded from below by a function (specifically the square-root) of the ratio L_{i+1}/L_i, for every i. As it turns out, what was required to tighten the analysis is a better understanding about the relationships between the cover time and the extreme step-lengths, namely, L_0, L_1 and L_{k-1}. Specifically, in proving the precise lower bound we have two components, one for the "local" part (exploitation) and the other for the "global" part (exploration). We showed that in order to be efficient on the local part, the small step-lengths need to be small, whereas in order to be efficient on the global part (traversing large distances fast), the largest step-length needs to be large. This allowed us to widen the ratios between consecutive step-lengths, consequently increasing the lower bound.

In order to obtain the precise upper bound, we improved the analysis in [7] of the Weierstrassian random walk process. This, in particular, required overcoming non-trivial issues concerning dependencies between variables that were overlooked in [7]. By doing this, we also refined the estimates on the order of magnitude of other dependencies. In addition, we had to incorporate short-time probability bounds for each step-length used by the process, and perform a tighter analysis of the part of the walk that corresponds to the largest step length L_{k-1}.

We next describe our contribution in more details.

1.4.1 The Lower Bound

We begin with the statement of the lower bound. The formal proof is given in Section 2.

▶ **Theorem 3.** *Let k and n be positive integers. The cover time of any k-scales search X on C_n is:*

$$\mathbb{E}(t_{cov}(n,k)) = n^{1+\frac{1}{2k-1}} \cdot \Omega(1/k).$$

1.4.2 The Upper Bound

The following theorem implies that up to lower order terms, the cover time of the Weierstrassian random walk matches the lower bound of the cover time of any k-scales search, as given by Theorem 3, for $2 \leq k \leq \log n$, i.e., for all potential scales.

▶ **Theorem 4.** *Let k be an integer such that $2 \leq k \leq \log_2 n$. The Weierstrassian random walk with parameter $b = \lfloor n^{\frac{2}{2k-1}} \rfloor$ is a k-scales search that achieves a cover time of:*

$$n^{1+\frac{1}{2k-1}} \cdot O\left(k^2 \log^2 n\right).$$

Observe that combining Theorems 3 and 4 we obtain the best cover time $Cov_{k,n}$ achievable by a k-scales search on C_n, which is $\tilde{\Theta}\left(n^{1+\frac{1}{2k-1}}\right)$ for any $2 \leq k \leq \log n$. For particular values of k, we thus have:

k	1	2	3	4	5	...	$\log n$
$\mathbb{E}(t_{cov}(n,k))$	$\Theta(n^2)$	$\tilde{\Theta}(n^{\frac{4}{3}})$	$\tilde{\Theta}(n^{\frac{6}{5}})$	$\tilde{\Theta}(n^{\frac{8}{7}})$	$\tilde{\Theta}(n^{\frac{10}{9}})$	...	$O(n \log^3 n)$

Theorem 4 follows immediately from the following more general theorem, by taking $b = n^{\frac{2}{2k-1}}$.

▶ **Theorem 5.** *Let b, k, n be integers such that $b^{k-1} < n \leq b^k$. The cover time of the Weierstrassian random walk on C_n with parameter b is*

$$O\left(n \max\left\{\frac{b^k}{n}, \frac{n}{b^{k-1}}\right\} \cdot k^2 \cdot \log b \cdot \log n\right) = \tilde{O}\left(\max\left\{b^k, \frac{n^2}{b^{k-1}}\right\}\right).$$

The formal proof of Theorem 5 is deferred to the full version. In Section 3 we provide a sketch of the proof.

As mentioned, Theorem 5 using the particular value $b = n^{\frac{2}{2k-1}}$ gives a tight upper bound for k-scales search. However, since the Weierstrassian random walk is of independent interest as it is used in biology, it might be useful to understand its cover time also for other values of b. Note that Lemmas 6 and 7 below, when applied to the Weierstrassian random walk on C_n, show that the cover time is at least $\Omega\left(\max\{n\sqrt{b}, \frac{n^2}{b^{k-1}}\}\right)$. This is quite close to the bound $\tilde{O}\left(\max\left\{b^k, \frac{n^2}{b^{k-1}}\right\}\right)$ of the theorem. Indeed if $n \geq b^{k-\frac{1}{2}}$, both bounds match, up to logarithmic terms. If $n \leq b^{k-\frac{1}{2}}$, the ratio of the bounds is $\frac{b^{k-\frac{1}{2}}}{n}$.

2 The Lower Bound Proof

The goal of this section is to establish the lower bound in Theorem 3. For this purpose, consider a k-scales search X on the cycle C_n and denote $(L_i)_{i=0}^{k-1}$ its step lengths with $L_i < L_{i+1}$ for all $i \in [k-2]$. For convenience of writing we also set $L_k = n$, but it should be clear that it is actually not a step length of the walk. Let p_i denote the probability of taking the step length L_i.

The theorem will follow from the combination of two lemmas. The first one, Lemma 6, stems from the analysis of the number of nodes that can be visited during L_{i+1} time steps. It forces $L_0 L_1$ as well as the ratios L_{i+1}/L_i for all $1 \leq i \leq k-1$ to be small enough in order to have a small cover time. The second one, Lemma 7, comes from bounding the cover time by the time it takes to go to a distance of at least $n/3$. It forces L_{k-1} to be big enough to have a small cover time.

▶ **Lemma 6.** *The cover time of X is at least*

- $\mathbb{E}(t_{cov}) = \Omega(n\sqrt{L_0 L_1})$.
- $\mathbb{E}(t_{cov}) = \Omega\left(\frac{n}{k}\sqrt{\frac{L_{i+1}}{L_i}}\right)$ *for any* $1 \leq i \leq k-1$.

The second part of Lemma 6 was already given in [7]. We sketch here the ideas behind the proof of the first part, namely, that the cover time is at least of order $n\sqrt{L_0 L_1}$. Essentially, we count the expected number of nodes that can be visited in a time duration of L_1, which we call a *phase*. A jump of length $L_i \geq L_1$ will not contribute to visiting a new node during this time duration. Thus, we may suppose that there are only jumps of length L_0. Since $L_1 \leq n$, the process does not do a turn of the cycle and, therefore, it can be viewed as a walk on $\mathbb{Z}$. Furthermore, since every jump has length L_0, we can couple this walk by a corresponding simple random walk, that does steps of length 1, during a time duration of L_1/L_0. The expected number of nodes visited during a phase is thus of order $\sqrt{L_1/L_0}$. It follows that we need at least $n/(\sqrt{L_1/L_0})$ such phases before covering the cycle. Since a phase lasts for L_1 time, the cover time is at least of order $n\sqrt{L_0 L_1}$. The full proof of Lemma 6, including the part that was proven in [7], appears in the full version.

▶ **Lemma 7.** *The cover time of X is at least $\Omega(n^2 \frac{\mu}{\sigma^2})$, where $\mu = \frac{1}{2}\sum_{i\leq k-1} p_i L_i$ and $\sigma^2 = \frac{1}{2}\sum_{i\leq k-1} p_i L_i^2$ are the mean and variance of the jump lengths, respectively. In particular, the cover time is:*

$$\mathbb{E}(t_{cov}) = \Omega\left(\frac{n^2}{L_{k-1}}\right).$$

Proof. Let m_{cov} denote the random number of *steps* before all nodes of C_n are covered, and let t_{cov} be the random cover time of the process. By Wald's identity, we have:

$$\mathbb{E}(t_{cov}) = \mathbb{E}(m_{cov}) \cdot \mu, \tag{1}$$

where $\mu = \frac{1}{2}\sum_{i=0}^{k-1} p_i L_i$ is the expected length, and hence the expected time, of a jump (the factor $\frac{1}{2}$ comes from the laziness). By Markov's inequality, we have:

$$\Pr\left(m_{cov} < 2\mathbb{E}(m_{cov})\right) \geq 1/2.$$

Let N_m be the (random) number of nodes visited by step m. We have:

$$\mathbb{E}(N_{2\mathbb{E}(m_{cov})}) \geq \mathbb{E}\left(N_{2\mathbb{E}(m_{cov})} \mid m_{cov} < 2\mathbb{E}(m_{cov})\right) \cdot \Pr\left(m_{cov} < 2\mathbb{E}(m_{cov})\right) \geq n \cdot \frac{1}{2}.$$

Define D_m as the maximal distance of the process from step 0 up to step m, i.e., $D_m = \max_{s\leq m}|X(s)|$. Since $N_m \leq 2D_m + 1$, we have:

$$2\mathbb{E}(D_{2\mathbb{E}(m_{cov})}) + 1 \geq \mathbb{E}(N_{2\mathbb{E}(m_{cov})}) \geq n/2.$$

As shown in [13] for general one-dimensional random walks, we have $\mathbb{E}(D_m) = O\left(\sigma\sqrt{m}\right)$, where σ is the standard deviation of the length distribution, i.e., $\sigma^2 = \frac{1}{2}\sum_i p_i L_i^2$. Thus, we have:

$$\sqrt{\mathbb{E}(m_{cov})}\sigma = \Omega(n),$$

and so:

$$\mathbb{E}(m_{cov}) = \Omega\left(\frac{n^2}{\sigma^2}\right),$$

and by Eq. (1), we get:

$$\mathbb{E}(t_{cov}) = \Omega\left(n^2 \frac{\mu}{\sigma^2}\right) = \Omega\left(n^2 \frac{\sum_{i=0}^{k-1} L_i p_i}{\sum_{i=0}^{k-1} L_i^2 p_i}\right),$$

which proves the first part of the lemma.

In order to prove the second part, note that since L_{k-1} is the biggest step length, we have $\sum_{i=0}^{k-1} p_i L_i (1 - \frac{L_i}{L_{k-1}}) \geq 0$, and so $\frac{\sum_{i=0}^{k-1} L_i p_i}{\sum_{i=0}^{k-1} L_i^2 p_i} \geq \frac{1}{L_{k-1}}$. Therefore,

$$\mathbb{E}(t_{cov}) = \Omega\left(\frac{n^2}{L_{k-1}}\right),$$

which completes the proof of Lemma 7. ◀

Next, it remains to show how Theorem 3 follows by combining Lemma 6 and Lemma 7. First, consider the lower bound of $\Omega(n^2/L_{k-1})$ in Lemma 7. If $L_{k-1} \leq n^{1-\frac{1}{2k-1}}$ then the bound in Theorem 3 immediately follows. Let us therefore assume that $L_{k-1} > n^{1-\frac{1}{2k-1}}$.

Define $\alpha_0 = L_0 L_1$ and $\alpha_i = \frac{L_{i+1}}{L_i}$ for $i \in \{1, 2, \ldots, k-2\}$. As

$$\prod_{i=0}^{k-2} \alpha_i = L_0 L_{k-1},$$

there must exist an index $0 \leq i \leq k-2$ such that $\alpha_i \geq (L_0 L_{k-1})^{\frac{1}{k-1}}$. Thus, by Lemma 6, the cover time is at least

$$\Omega\left(\frac{n}{k} (L_0 L_{k-1})^{\frac{1}{2(k-1)}}\right).$$

Since $L_{k-1} > n^{1-\frac{1}{2k-1}} = n^{\frac{2k-2}{2k-1}}$ and $L_0 \geq 1$, we conclude that the cover time is at least

$$\mathbb{E}(t_{cov}) = \Omega\left(\frac{n}{k} \cdot n^{\frac{1}{2k-1}}\right),$$

as desired. This completes the proof of Theorem 3. ◀

3 Upper Bound Proof (Sketch)

Let us give the key ideas of the proof of Theorem 5. Some of the initial steps in the proof follow the technique in [7] (by doing so, we also corrected some mistakes in [7]). These parts are clearly mentioned below. Our main technical contribution that allowed us to obtain the precise upper bound, is the use of short-time probability bounds (see Eq. (6)), and a tighter analysis of the part of the walk that corresponds to the largest step length L_{k-1}.

In more details, let us consider the Weierstrassian walk on C_n, termed X. The following lemma establishes a link between the cover time of X and the point-wise probabilities of X. For completeness, we provide a formal proof of it in the full version, although it is not hard to obtain it using the technique in [7].

▶ **Lemma 8.** *If $p > 0$ and $m_0 > 0$ are such that, for any $x \in \{0, \ldots, n-1\}$,*

$$\frac{\sum_{m=m_0}^{2m_0} \Pr(X(m) = x)}{\sum_{m=0}^{m_0} \Pr(X(m) = 0)} \geq p, \tag{2}$$

then the cover time of the Weierstrassian random walk X on the cycle C_n is $O\left(m_0 p^{-1} k \log n\right)$.

Using Lemma 8, the bound of Theorem 5 can be established by proving bounds on the probability to visit node $x \in [0, n)$ at step m.

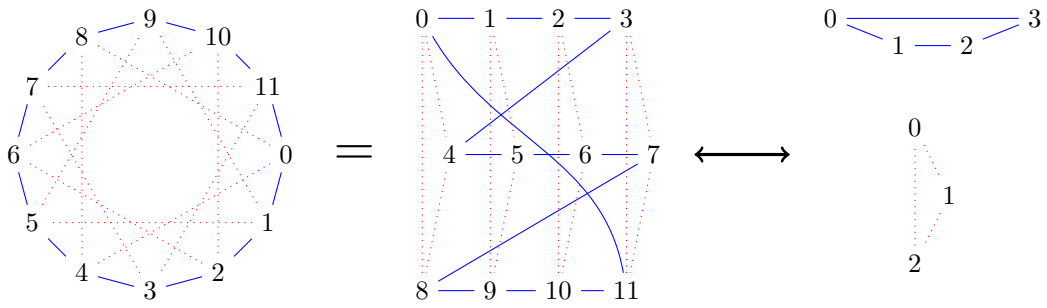


Figure 1 The first two graphs represent, in different node disposition, the Weierstrassian walk on C_{12} with parameter $b = 4$. There are $k = 2$ jump lengths, $L_0 = 1$ (blue edges) and $L_1 = b = 4$ (red, dotted edges). To the right, we show the decomposition of C_{12} as $C_4 \times C_3$. For instance the node $x = 7 \in C_{12}$ will be represented by $x_0 = 3 \in C_4$ and $x_1 = 1 \in C_3$.

In order to simplify the presentation, assume first that $n = b^k$. Proceeding first as in [7], we view the k-lengths Weierstrassian random walks as k (dependent) random walks, by grouping together the jumps of the same length (see Figure 1). Define $S_i(m)$ as the algebraic count of the jumps of lengths b^i. E.g., if, by step m, there are exactly four positive jumps of length b^i, and one negative, then $S_i(m) = 3$. We have:

$$X(m) = \sum_{i=0}^{k-1} S_i(m) b^i.$$

Define also the following decomposition of C_n.

▶ **Definition 9** (Base b decomposition). *For any $x \in C_n$, we may decompose x in base b as*

$$x = \sum_{i=0}^{k-1} x_i b^i,$$

with $0 \leq x_i < b$. We call x_i the i-th coordinate of x (in base b).

It follows from Euclidean division, and the fact that $n = b^k$, that the base b decomposition is well-defined and unique for every $x \in C_n$. This decomposition is illustrated in Figure 1 (where we have taken $n = \hat{n} b^{k-1}$ to anticipate the more general case to follow).

Note that $X(m) = x$ in C_n if and only if

$$\sum_i (S_i(m) - x_i) b^i = 0 \mod n. \tag{3}$$

By taking Eq. (3) modulo b^i, for $i \leq k-1$, it is easy to show that Eq. (3) is equivalent to

$$S_i(m) = y_i \mod b,$$

for $y_i := x_i - b^{-i} \sum_{j<i} (S_j(m) - x_j) b^j \mod b$.

Thus, $X(m) = x$ is equivalent to $R_i(m) = y_i$ for all i, where $R_i = S_i \mod b$ is a random walk on C_b that moves with probability $\frac{p_i}{2}$. This process is illustrated in Figure 1, where $X(m) = 7$ is equivalent to $R_0(m) = 3$ and $R_1(m) = 2$.

Unfortunately, the R_i's and the y_i's are not independent, due to the fact that only one of the R_i can change between steps m and $m+1$, however, let us overlook this issue in this

informal outline. We then have:

$$\Pr(X(m) = x) \approx \prod_{i=0}^{k-1} \Pr(R_i(m) = y_i). \tag{4}$$

Recall that R_i is a random walk over C_b that moves with probability p_i. The following is a well-known property of the random walk over a cycle (see, e.g., Example 5.7 and Proposition 6.18 in [1]):

▷ Claim 10. For a simple random walk R on C_b that moves with probability $\frac{1}{2}$, and any $y \in C_b$,

$$\Pr\left(R(m) = y\right) = \begin{cases} O\left(1/\sqrt{m}\right) \text{ if } m < b^2 \\ b^{-1}(1 \pm \varepsilon_m) \text{ if } m \geq b^2, \end{cases} \tag{5}$$

with $\varepsilon_m = O(e^{-cmb^{-2}})$ where $c > 0$.

Considering that R_i moves with probability $\frac{p_i}{2} = \Theta(b^{-i})$, we can expect that, at step m, $R_i(m)$ has the same distribution as the lazy random walk with mp_i steps that moves with probability $\frac{1}{2}$. This is proved formally in the full version. Hence, by substituting m with mp_i in Claim 10, we obtain:

$$\Pr\left(R_i(m) = y_i\right) = \begin{cases} O\left(1/\sqrt{mp_i}\right) \text{ if } m < b^{i+2} \\ b^{-1}(1 \pm \varepsilon_{mp_i}) \text{ if } m \geq b^{i+2}. \end{cases} \tag{6}$$

Theorem 5 then follows from Eq. (4), Eq. (6) and Lemma 8. Essentially, to cover C_n, we need that each $R_i(m)$ is mixed, i.e., has some significant probability to visit any node y_i in C_b, which happens, as shown by Eq. (6), for $m > b^{k-1+2} = b^{k+1}$. Let us apply Lemma 8 with

$$m_0 := b^{k+1}.$$

We first establish a lower bound on $\sum_{m=m_0}^{2m_0} \Pr(X(m) = x)$. By Eq. (4) and Eq. (6), we have, for $m > m_0$,

$$\Pr(X(m) = x) \approx \prod_{0 \leq i \leq k-1} b^{-1}\left(1 - \varepsilon_{mp_i}\right) = \Theta\left(b^{-k}\right),$$

where the last equality is justified in the full version. Thus,

$$\sum_{m=m_0}^{2m_0} \Pr(X(m) = x) = \Omega\left(m_0 b^{-k}\right) = \Omega\left(b\right).$$

We need also to upper bound $\sum_{m=0}^{m_0} \Pr(X(m) = 0)$, which is the expected number of returns to the origin up to step m_0. To do this, we shall use the short-time bounds of Eq. (6).

Let us decompose the aforementioned sum as follows.

$$\sum_{m=0}^{m_0} \Pr(X(m) = 0) = 1 + \frac{1}{2} + \sum_{j=0}^{k-1} \sum_{m=1+b^j}^{b^{j+1}} \Pr(X(m) = 0) + \sum_{m=1+b^k}^{m_0} \Pr(X(m) = 0). \tag{7}$$

Fix j, such that $1 \leq j \leq k-1$ and let $m \in (b^j, b^{j+1}]$. By Eq. (4), in order to upper bound $\Pr(X(m) = 0)$ it is enough to bound $\Pr(R_i(m) = y_i)$ for every $i \leq k-1$. For $i > j$, we

bound $\Pr(R_i(m) = y_i)$ by 1. For $i \leq j-2$, we use Eq. (6) to upper bound $\Pr(R_i(m) = y_i)$ by $b^{-1}(1+\varepsilon_{mp_j})$. For $i = j-1$ and $i = j$, we bound $\Pr(R_i(m) = y_i)$ by $O\left(1/\sqrt{mp_{j-1}}\right)$ and $O\left(1/\sqrt{mp_j}\right)$, respectively. We thus obtain, by Eq. (4),

$$\begin{aligned}\Pr(X(m) = x) &= O\left(\frac{1}{\sqrt{mp_{j-1}}} \cdot \frac{1}{\sqrt{mp_j}} \cdot \prod_{0\leq i\leq j-2} b^{-1}\left(1+\varepsilon_{mp_j}\right)\right) \\ &= O\left(b^{-(j-1)} \cdot \frac{\sqrt{b}b^{j-1}}{m}\right) = O\left(\frac{\sqrt{b}}{m}\right),\end{aligned}$$

where we justify in the full version that $\prod_{0\leq i\leq j-2}(1+\varepsilon_{mp_j}) = O(1)$. Hence, we get:

$$\sum_{m=1+b^j}^{b^{j+1}} \Pr(X(m) = 0) = O(\sqrt{b}\log b), \tag{8}$$

by using that $\sum_{m=1+b^j}^{b^{j+1}} m^{-1} = \Theta\left(\int_{m=b^j}^{b^{j+1}} u^{-1}du\right) = \Theta(\log b)$. For the case $j = 0$, we bound $\Pr(R_i(m) = y_i)$ by 1 for $i > 1$ and $\Pr(R_0(m) = y_0)$ by $O(m^{-\frac{1}{2}})$, so that, by Eq. (4), $\Pr(X(m) = 0) = O(\frac{1}{\sqrt{m}})$. Hence, we get:

$$\sum_{m=2}^{b} \Pr(X(m) = 0) = O\left(\sqrt{b}\right). \tag{9}$$

Similarly, for $m \in (b^k, b^{k+1}]$, $\Pr(R_i(m) = y_i)$ is bounded by $b^{-1}(1+\varepsilon_{mp_i})$ for $i \leq k-2$, and by $\frac{1}{\sqrt{mp_{k-1}}}$ for $i = k-1$. Thus, for $m \in (b^k, b^{k+1}]$,

$$\Pr(X(m) = 0) = O\left(\frac{1}{\sqrt{m}\sqrt{b^{k-1}}}\right)$$

and, since $\sum_{m=1+b^k}^{b^{k+1}} \frac{1}{\sqrt{m}} = O\left(\int_{b^k}^{b^{k+1}} \frac{1}{\sqrt{u}}du\right) = O\left(\sqrt{b^{k+1}}\right)$, we get:

$$\sum_{m=1+b^k}^{b^{k+1}} \Pr(X(m) = x) = O\left(\frac{\sqrt{b^{k+1}}}{\sqrt{b^{k-1}}}\right) = O(b). \tag{10}$$

In total, by Eq. (7), combining Eqs. (8), (9) and (10), we find that the expected number of returns to the origin up to step b^{k+1} is

$$\sum_{m=0}^{m_0} \Pr(X(m) = 0) = O\left(k\sqrt{b}\log b + b\right) = O\left(kb\log b\right).$$

So that all together we have:

$$\frac{\sum_{m=m_0}^{2m_0} \Pr(X(m) = x)}{\sum_{m=0}^{m_0} \Pr(X(m) = 0)} = \Omega\left(\frac{b}{kb\log b}\right) = \Omega\left(\frac{1}{k\log b}\right).$$

Thus, by Lemma 8, the cover time of X is at most:

$$O(m_0 \cdot k\log b \cdot k\log n) = O(b^{k+1}k^2\log b\log n) = O(nbk^2\log b\log n), \tag{11}$$

as claimed by Theorem 5, for the case where $n = b^k$.

Consider now a more general case, in which n is a multiple of b^{k-1}. Here, we can write $n = \hat{n}b^{k-1}$, where $\hat{n} \in (0, b]$ is an integer. What changes in this case is that the last coordinate, R_{k-1}, is now a random walk over $C_{\hat{n}}$ instead of over C_b, as depicted in Figure 1. R_{k-1} is thus mixed after a number of steps:

$$\hat{n}^2 p_{k-1}^{-1} = \Theta(b^{k-1}\hat{n}^2) = \Theta(n^2/b^{k-1}).$$

On the other hand, after $\Theta(b^{k-2+2}) = \Theta(b^k)$ steps, the other coordinates are mixed. Thus, the number of steps needed before every coordinate R_i is mixed is:

$$m_0 = \Theta\left(\max\{b^k, n^2/b^{k-1}\}\right), \tag{12}$$

which is again the order of magnitude of the cover time of X, up to polylogarithmic factors. Note that when $n = b^k$, Eq. (12) recovers the cover time of order $\tilde{\Theta}(b^{k+1})$. Furthermore, the ratio of the cover time for $n = b^k$ and $n = \hat{n}b^{k-1}$ is of order $\frac{b^{k+1}}{\max\{b^k, b^{k-1}\hat{n}^2\}} = \min\{b, \frac{b^2}{\hat{n}^2}\}$. When b is large (which corresponds to k being small), this can be significant. Hence, naively bounding $\hat{n}$ from above by b would not suffice to yield an optimal bound.

The general case, when n is not necessarily a multiple of b^{k-1}, needs to be treated with more care. What changes in this case is that we can no longer decompose X as k dependent random walks on $C_b \times \cdots \times C_b \times C_{\frac{n}{b^{k-1}}}$, since $\frac{n}{b^{k-1}}$ is not an integer. Instead, we define Z as the process that does the same jumps as X, but on the infinite line $\mathbb{Z}$, and we also define

$$\hat{n} := \lfloor n/b^{k-1} \rfloor.$$

Then, we use almost the same decomposition, where Z is viewed as k dependent random walks over $C_b \times \cdots \times C_b \times \mathbb{Z}$. The process corresponding to the last coordinate, R_{k-1}, is now a random walk on $\mathbb{Z}$, and we are interested especially on the probability of the event $R_{k-1}(m) = x_{k-1}$ for $x_{k-1} \in [0, \hat{n}]$. As the coordinate R_{k-1} is not restricted to $[0, \hat{n}]$, we need to pay attention that the walk does not go too far.

4 Discussion

The upper bound in Theorem 4 implies that almost linear time performances, as those obtained by Lévy Flights, can be achieved with a number of step lengths that ranges from logarithmic to linear. This further suggests that cover time performances similar to those of Lévy Flights can be seen by a large number of different processes. In practice, if one aims to fit empirical statistics of an observed process to a theoretical model of a particular heterogeneous step length distribution, the large degree of freedom can make this task extremely difficult, if not impossible. On the other hand, the fact that so many processes yield similar cover times may justify viewing all of them as essentially equivalent. This interpretation may also be relevant to the current debate regarding whether animals' movement is better represented by Lévy Flights or by CCRW distributions with 2 or 3 scales [28, 15, 22, 30]. Moreover, the fact that many heterogeneous step processes yield similar performances to Lévy Flights may imply that limiting the empirical fit to either Lévy Flights or CCRW searches with 2 or 3 scales may be too restrictive. Our work may suggest that instead, the focus could shift to identifying the number of scales involved in the search.

When combined with appropriate empirical measurements, our lower bound can potentially be used to indirectly show that a given intermittent process uses strictly more than a certain number of step lengths. For example, if the process is empirically shown as a heterogeneous random walk whose cover time is almost linear, then Theorem 3 implies that it must use

roughly logarithmic number of step lengths. From a methodological perspective, such a result would be of particular appeal as demonstrating lower bounds in biology through mathematical arguments is extremely rare [8, 16].

Finally, we note that most of the theoretical research on heterogeneous search processes which is based on differential equation techniques and computer simulations. In contrast, and similarly to [7], our methodology relies on algorithmic analysis techniques and discrete probability arguments, which are more commonly used in theoretical computer science. We believe that the computational approach presented here can contribute to a more fundamental understanding of these search processes.

References

1. D. Aldous and J. A. Fill. Reversible markov chains and random walks on graphs, 2002.
2. M. Auger-Méthé, A. Derocher, M. Plank, E. Codling, and M.. Lewis. Differentiating the lévy walk from a composite correlated random walk. *Methods in Ecology and Evolution*, 6(10):1179–1189, 2015.
3. Simon Benhamou and Julien Collet. Ultimate failure of the lévy foraging hypothesis: Two-scale searching strategies outperform scale-free ones even when prey are scarce and cryptic. *Journal of theoretical biology*, 387, October 2015.
4. Olivier Bénichou, C Loverdo, M Moreau, and R Voituriez. Intermittent search strategies. *Reviews of Modern Physics*, 83(1), 2011.
5. Otto G. Berg, Robert B. Winter, and Peter H. Von Hippel. Diffusion-driven mechanisms of protein translocation on nucleic acids. 1. models and theory. *Biochemistry*, 20(24), 1981.
6. L. Boczkowski, O. Feinerman, A. Korman, and E. Natale. Limits for rumor spreading in stochastic populations. In *9th Innovations in Theoretical Computer Science Conference*, pages 49:1–49:21, 2018.
7. L. Boczkowski, B. Guinard, A. Korman, Z. Lotker, and M. Renault. Random walks with multiple step lengths. *LATIN 2018: Theoretical Informatics*, pages 174–186, January 2018.
8. L. Boczkowski, E. Natale, O. Feinerman, and A. Korman. Limits on reliable information flows through stochastic populations. *PLOS Computational Biology*, 14(6):1–15, June 2018.
9. D. Boyer, G. Ramos-Fernandez, O. Miramontes, J. Mateos, G. Cocho, H. Larralde, H. Ramos, and F. Rojas. Scale-free foraging by primates emerges from their interaction with a complex environment. *Proceedings of the Royal Society of London B: Biological Sciences*, 273(1595), 2006.
10. Bernard Chazelle. Natural algorithms. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 422–431, 2009.
11. Bernard Chazelle. The convergence of bird flocking. *J. ACM*, 61(4):21:1–21:35, 2014.
12. M. Chupeau, O. Bénichou, and R. Voituriez. Cover times of random searches. *Nature Physics*, 11(10):844, 2015.
13. A. Comtet and S. N. Majumdar. Precise asymptotics for a random walker's maximum. *Journal of Statistical Mechanics: Theory and Experiment*, 6:06013, June 2005.
14. M. Coppey, O. Bénichou, R. Voituriez, and M. Moreau. Kinetics of Target Site Localization of a Protein on DNA: A Stochastic Approach. *Biophysical Journal*, 87(3), 2004.
15. Monique de Jager, Franz J Weissing, Peter MJ Herman, Bart A Nolet, and Johan van de Koppel. Lévy walks evolve through interaction between movement and environmental complexity. *Science*, 332(6037):1551–1553, 2011.
16. Ofer Feinerman and Amos Korman. Theoretical distributed computing meets biology: A review. In *International Conference on Distributed Computing and Internet Technology*, pages 1–18. Springer, 2013.
17. Ofer Feinerman and Amos Korman. The ANTS problem. *Distributed Computing*, 30(3):149–168, 2017.

18 T. Harris, E. Banigan, D. Christian, et al. Generalized lévy walks and the role of chemokines in migration of effector cd8(+) t cells. *Nature*, 486(7404), 2012.

19 T. Hills, P. Todd, D. Lazer, A. Redish, I. Couzin, and Cognitive Search Research Group. Exploration versus exploitation in space, mind, and society. *Trends in cognitive sciences*, 19(1):46–54, 2015.

20 B. D. Hughes, M. F. Shlesinger, and E. W. Montroll. Random Walks with Self-Similar Clusters. *Proceedings of the National Academy of Science*, 78:3287–3291, June 1981.

21 N. E. Humphries, N. Queiroz, J. R. M. Dyer, N. G. Pade, M. K. Musyl, K. M. Schaefer, D. W. Fuller, J. M. Brunnschweiler, T. K. Doyle, J. D. R. Houghton, G. C. Hays, C. S. Jones, L. R. Noble, V. J. Wearmouth, E. J. Southall, and D. W. Sims. Environmental context explains Lévy and Brownian movement patterns of marine predators. *Nature*, 465:1066–1069, June 2010.

22 Vincent A. A. Jansen, Alla Mashanova, and Sergei Petrovskii. Comment on "lévy walks evolve through interaction between movement and environmental complexity". *Science*, 335(6071):918–918, 2012.

23 A. Koelzsch, A. Alzate, F. Bartumeus, M. de Jager, E. Weerman, G. Hengeveld, M. Naguib, B. Nolet, and J. van de Koppel. Experimental evidence for inherent lévy search behaviour in foraging animals. *Proceedings of the Royal Society B*, 282, May 2015.

24 David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. *Markov Chains and Mixing Times*. American Mathematical Society, 2008.

25 Michael Lomholt, Koren Tal, Ralf Metzler, and Joseph Klafter. Lévy strategies in intermittent search processes are advantageous. *Proceedings of the National Academy of Sciences*, 105(32), 2008.

26 Cameron Musco, Hsin-Hao Su, and Nancy A. Lynch. Ant-inspired density estimation via random walks: Extended abstract. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 469–478, 2016.

27 G. Oshanin, H. Wio, K. Lindenberg, and S. Burlatsky. Intermittent random walks for an optimal search strategy: one-dimensional case. *Journal of Physics: Condensed Matter*, 19(6), 2007.

28 G. H. Pyke. Understanding movements of organisms: it's time to abandon the lévy foraging hypothesis. *Methods in Ecology and Evolution*, 6(1):1–16, 2015.

29 D. A. Raichlen, B. M. Wood, A. D. Gordon, A. Z. P. Mabulla, F. W. Marlowe, and H. Pontzer. Evidence of Lévy walk foraging patterns in human hunter-gatherers. *Proceedings of the National Academy of Science*, 111:728–733, January 2014.

30 A. Reynolds. Mussels realize Weierstrassian Lévy walks as composite correlated random walks. *Scientific Reports*, 4:4409, March 2014.

31 A. Reynolds, G. Santini, G. Chelazzi, and S. Focardi. The weierstrassian movement patterns of snails. *Royal Society Open Science*, 4(6):160941, 2017.

32 I. Rhee, M. Shin, S. Hong, K. Lee, and S. Chong. On the lévy-walk nature of human mobility. In *IEEE INFOCOM 2008*, 2011.

33 F. L. Schuster and M. Levandowsky. Chemosensory responses of acanthamoeba castellanii: Visual analysis of random movement and responses to chemical signals. *Journal of Eukaryotic Microbiology*, 43(2):150–158, 1996.

34 D. W. Sims, E. J. Southall, N. E. Humphries, G. C. Hays, C. J. A. Bradshaw, J. W. Pitchford, A. James, M. Z. Ahmed, A. S. Brierley, M. A. Hindell, D. Morritt, M. K. Musyl, D. Righton, E. L. C. Shepard, V. J. Wearmouth, R. P. Wilson, M. J. Witt, and J. D. Metcalfe. Scaling laws of marine predator search behaviour. *Nature*, 451:1098–1102, February 2008.

35 G. Viswanathan, V. Afanasyevt, S. Buldyrev, E. Murphyt, P. Princet, and H. Stanley. Lévy flight search patterns of wandering albatrosses. *Nature*, 381(6581), 1996.

36 G. Viswanathan, S. Buldyrev, S. Havlin, M. da Luz, E. Raposo, and E. Stanley. Optimizing the success of random searches. *Nature*, 401(6756), 1999.

Solving Connectivity Problems Parameterized by Treedepth in Single-Exponential Time and Polynomial Space

Falko Hegerfeld
Humboldt-Universität zu Berlin, Germany
hegerfeld@informatik.hu-berlin.de

Stefan Kratsch
Humboldt-Universität zu Berlin, Germany
kratsch@informatik.hu-berlin.de

Abstract

A breakthrough result of Cygan et al. (FOCS 2011) showed that connectivity problems parameterized by treewidth can be solved much faster than the previously best known time $\mathcal{O}^*(2^{\mathcal{O}(\mathrm{tw}\log \mathrm{tw})})$. Using their inspired Cut&Count technique, they obtained $\mathcal{O}^*(\alpha^{\mathrm{tw}})$ time algorithms for many such problems. Moreover, they proved these running times to be optimal assuming the Strong Exponential-Time Hypothesis. Unfortunately, like other dynamic programming algorithms on tree decompositions, these algorithms also require *exponential space*, and this is widely believed to be unavoidable. In contrast, for the slightly larger parameter called treedepth, there are already several examples of matching the time bounds obtained for treewidth, but using only polynomial space. Nevertheless, this has remained open for connectivity problems.

In the present work, we close this knowledge gap by applying the Cut&Count technique to graphs of small treedepth. While the general idea is unchanged, we have to design novel procedures for counting consistently cut solution candidates using only polynomial space. Concretely, we obtain time $\mathcal{O}^*(3^d)$ and polynomial space for Connected Vertex Cover, Feedback Vertex Set, and Steiner Tree on graphs of treedepth d. Similarly, we obtain time $\mathcal{O}^*(4^d)$ and polynomial space for Connected Dominating Set and Connected Odd Cycle Transversal.

2012 ACM Subject Classification Mathematics of computing → Paths and connectivity problems; Theory of computation → Parameterized complexity and exact algorithms; Mathematics of computing → Combinatorial algorithms

Keywords and phrases Parameterized Complexity, Connectivity, Treedepth, Cut&Count, Polynomial Space

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.29

Related Version A full version of the paper is available at [18], `http://arxiv.org/abs/2001.05364`.

1 Introduction

The goal of parameterized complexity is to reign in the combinatorial explosion present in NP-hard problems with the help of a secondary parameter. This leads us to the search for fixed-parameter tractable (FPT) algorithms, i.e., algorithms with running time $\mathcal{O}(f(k)n^c)$ where n is the input size, k is the secondary parameter, f is a computable function, and c is a constant. There are several books giving a broad overview of parameterized complexity [10, 12, 13, 28]. One of the success stories of parameterized complexity is a graph parameter called treewidth. A large swath of graph problems admit FPT-algorithms when parameterized by treewidth as witnessed by, amongst other things, Courcelle's theorem [9]. However, the function f resulting from Courcelle's theorem is non-elementary [16]. Thus, a natural goal is to find algorithms with a smaller, or ideally minimal, dependence on the treewidth in the running time, i.e. algorithms where f is as small as possible. Problems only involving

local constraints usually permit a single-exponential dependence on the treewidth (tw) in the running time, i.e. time $\mathcal{O}^*(\alpha^{\mathrm{tw}})$ for some small constant α,[1] by means of dynamic programming on tree decompositions [1, 31, 32, 33]. For many of these problems we also know the optimal base α if we assume the strong exponential-time hypothesis (SETH) [22]. For a long time a single-exponential running time seemed to be out of reach for problems involving global constraints, in particular for connectivity constraints. This changed when Cygan et al. [11] introduced the Cut&Count technique, which allowed them to obtain single-exponential-time algorithms for many graph problems involving connectivity constraints. Again, many of the resulting running times can be shown to be optimal assuming SETH [11].

The issue with treewidth-based algorithms is that dynamic programming on tree decompositions seems to inherently require exponential space. In particular, Chen et al. [8] devised a model for single-pass dynamic programming algorithms on tree decompositions and showed that such algorithms require exponential space for Vertex Cover and 3-Coloring. Algorithms requiring exponential time and exponential space usually run out of available space before they hit their time limit [34]. Hence, it is desirable to reduce the space requirement while maintaining the running time. As discussed, this seems implausible for treewidth. Instead, we consider a different, but related, parameter called treedepth. Treedepth is a slightly larger parameter than treewidth and of great importance in the theory of sparse graphs [25, 26, 27]. It has been studied under several names such as minimum elimination tree height [7], ordered chromatic number [19], and vertex ranking [6]. Fürer and Yu [17] established an explicit link between treedepth and tree decompositions, namely that treedepth is obtained by minimizing the maximum number of forget nodes in a root-leaf-path over all nice tree decompositions (see [20] for a definition). Many problems parameterized by treedepth allow branching algorithms on *elimination forests*, also called *treedepth decompositions*, that match the running time of the treewidth-algorithms, but replacing the dependence on treewidth by treedepth, while only requiring polynomial space [8, 17, 29].

Our contribution. The Cut&Count technique reduces problems with connectivity constraints to counting problems of certain cuts, called *consistent cuts*. We show that for several connectivity problems the associated problem implied by the Cut&Count technique can be solved in time $\mathcal{O}^*(\alpha^d)$ and polynomial space, where α is a constant and d is the depth of a given elimination forest. Furthermore, the base α matches the base in the running time of the corresponding treewidth-algorithm. Concretely, given an elimination forest of depth d for a graph G we prove the following results:

- Connected Vertex Cover, Feedback Vertex Set, and Steiner Tree can be solved in time $\mathcal{O}^*(3^d)$ and polynomial space.
- Connected Dominating Set and Connected Odd Cycle Transversal can be solved in time $\mathcal{O}^*(4^d)$ and polynomial space.

Related work. The Cut&Count technique leads to randomized algorithms as it relies on the Isolation Lemma. At the cost of a worse base in the running time, Bodlaender et al. [5] present a generic method, called the rank-based approach, to obtain deterministic single-exponential-time algorithms for connectivity problems parameterized by treewidth; the rank-based approach is also able to solve counting variants of several connectivity problems. Fomin et al. [15] use matroid tools to, amongst other results, reobtain the deterministic running times of the rank-based approach. In a follow-up paper, Fomin et al. [14] manage to

[1] The $\mathcal{O}^*$-notation hides polynomial factors in the input size.

improve several of the deterministic running times using their matroid tools. Multiple papers adapt the Cut&Count technique and rank-based approach to graph parameters different from treewidth. Bergougnoux and Kanté [3] apply the rank-based approach to obtain single-exponential-time algorithms for connectivity problems parameterized by cliquewidth. The same authors [4] generalize, incurring a loss in the running time, this approach to a wider range of parameters including rankwidth and mim-width. Pino et al. [30] use the Cut&Count technique and rank-based approach to obtain fast deterministic and randomized algorithms for connectivity problems parameterized by branchwidth.

Lokshtanov and Nederlof [23] present a framework using algebraic techniques, such as Fourier, Möbius, and Zeta transforms, to reduce the space usage of certain dynamic programming algorithms from exponential to polynomial. Fürer and Yu [17] adapt this framework to the setting where the underlying set (or graph) is dynamic instead of static, in particular for performing dynamic programming along the bags of a tree decomposition, and obtain a $\mathcal{O}^*(2^d)$-time, where d is the depth of a given elimination forest, and polynomial-space algorithm for counting perfect matchings. Using the same approach, Belbasi and Fürer [2] design an algorithm counting the number of Hamiltonian cycles in time $\mathcal{O}^*((4k)^d)$, where k is the width and d the depth of a given tree decomposition, and polynomial space. Furthermore, they also present an algorithm for the traveling salesman problem with the same running time, but requiring pseudopolynomial space.

Organization. We describe the preliminary definitions and notations in Section 2. In Section 3 we first discuss the Cut&Count setup and give a detailed exposition for Connected Vertex Cover. Afterwards, we explain what general changes can occur for the other problems and then discuss Feedback Vertex Set; the remaining problems can be found in the full version. We conclude in Section 4. Proofs that are delegated to the full version [18] are denoted by $\star$.

2 Preliminaries

2.1 Notation

Let $G = (V, E)$ be an undirected graph. We denote the number of vertices by n and the number of edges by m. For a vertex set $X \subseteq V$, we denote by $G[X]$ the subgraph of G that is induced by X. The *open neighborhood* of a vertex v is given by $N(v) = \{u \in V \mid \{u, v\} \in E\}$, whereas the *closed neighborhood* is given by $N[v] = N(v) \cup \{v\}$. We extend these notations to sets $X \subseteq V$ by setting $N[X] = \bigcup_{v \in X} N[v]$ and $N(X) = N[X] \setminus X$. Furthermore, we denote by $\mathtt{cc}(G)$ the number of connected components of G.

A *cut* of a set $X \subseteq V$ is a pair (X_L, X_R) with $X_L \cap X_R = \emptyset$ and $X_L \cup X_R = X$, we also use the notation $X = X_L \uplus X_R$. We refer to X_L and X_R as the *left* and *right side* of the cut, respectively. Note that either side may be empty, although usually the left side is nonempty.

For two integers a, b we write $a \equiv b$ to indicate equality modulo 2, i.e., a is even if and only if b is even. We use Iverson's bracket notation: for a predicate p, we have that $[p]$ is 1 if p is true and 0 otherwise. For a function f we denote by $f[v \mapsto \alpha]$ the function $(f \setminus \{(v, f(v))\}) \cup \{(v, \alpha)\}$. By $\mathbb{F}_2$ we denote the field of two elements. For a field or ring $\mathbb{F}$ we denote by $\mathbb{F}[Z_1, Z_2, \ldots, Z_t]$ the ring of polynomials in the indeterminates $Z_1, Z_2, \ldots, Z_t$ with coefficients in $\mathbb{F}$. With $\mathcal{O}^*$ we hide polynomial factors, i.e. $\mathcal{O}^*(f(n)) = \mathcal{O}(f(n)\mathrm{poly}(n))$. For a natural number n, we denote by $[n]$ the set of integers from 1 to n.

2.2 Treedepth

▶ **Definition 2.1.** An *elimination forest* of an undirected graph $G = (V, E)$ is a rooted forest $\mathcal{T} = (V, E_{\mathcal{T}})$ such that for every edge $\{u, v\} \in E$ either u is an ancestor of v in $\mathcal{T}$ or v is an ancestor of u in $\mathcal{T}$. The *depth* of a rooted forest is the largest number of nodes on a path from a root to a leaf. The *treedepth* of G is the minimum depth over all elimination forests of G and is denoted by $\mathrm{td}(G)$.

We slightly extend the notation for elimination forests used by Pilipczuk and Wrochna [29]. For a rooted forest $\mathcal{T} = (V, E_{\mathcal{T}})$ and a node $v \in V$ we denote by $\texttt{tree}[v]$ the set of nodes in the subtree rooted at v, including v. By $\texttt{tail}[v]$ we denote the set of all ancestors of v, including v. Furthermore, we define $\texttt{tree}(v) = \texttt{tree}[v] \setminus \{v\}$, $\texttt{tail}(v) = \texttt{tail}[v] \setminus \{v\}$, and $\texttt{broom}[v] = \{v\} \cup \texttt{tail}(v) \cup \texttt{tree}(v)$. By $\texttt{child}(v)$ we denote the children of v.

Note that an elimination forest $\mathcal{T}$ of a connected graph consists only of a single tree.

2.3 Isolation Lemma

▶ **Definition 2.2.** A function $\mathbf{w} \colon U \to \mathbb{Z}$ *isolates* a set family $\mathcal{F} \subseteq 2^U$ if there is a unique $S' \in \mathcal{F}$ with $\mathbf{w}(S') = \min_{S \in \mathcal{F}} \mathbf{w}(S)$, where for subsets X of U we define $\mathbf{w}(X) = \sum_{u \in X} \mathbf{w}(u)$.

▶ **Lemma 2.3** (Isolation Lemma, [24])**.** *Let $\mathcal{F} \subseteq 2^U$ be a nonempty set family over a universe U. Let $N \in \mathbb{N}$ and for each $u \in U$ choose a weight $\mathbf{w}(u) \in [N]$ uniformly and independently at random. Then $\mathbb{P}[\mathbf{w}$ isolates $\mathcal{F}] \geq 1 - |U|/N$.*

When counting objects modulo 2 the Isolation Lemma allows us to avoid unwanted cancellations by ensuring with high probability that there is a unique solution. In our applications, we will choose N so that we obtain an error probability of less than $1/2$.

3 Cut&Count

In this section $G = (V, E)$ always refers to a connected undirected graph. For the sake of a self-contained presentation, we state the required results, with their proofs in the appendix, for the Cut&Count technique again, mostly following the presentation of Cygan et al. [11]. Our approach only differs from that of Cygan et al. [11] in the counting sub-procedure.

We begin by describing the Cut&Count setup and then present the counting sub-procedure for Connected Vertex Cover. Afterwards we explain how to adapt the counting sub-procedure for the other problems. Our exposition is the most detailed for Connected Vertex Cover, whereas the analogous parts of the other problems will not be discussed in such detail.

3.1 Setup

Suppose that we want to solve a problem on G involving connectivity constraints, then we can make the following general definitions. The solutions to our problem are subsets of a universe U which is related to G. Let $\mathcal{S} \subseteq 2^U$ denote the set of solutions and we want to determine whether $\mathcal{S}$ is empty or not. The Cut&Count technique consists of two parts:

- **The Cut part:** We relax the connectivity constraints to obtain a set $\mathcal{S} \subseteq \mathcal{R} \subseteq 2^U$ of possibly connected solutions. The set $\mathcal{Q}$ will contain pairs (X, C) consisting of a candidate solution $X \in \mathcal{R}$ and a consistent cut C of X, which is defined in Definition 3.1.
- **The Count part:** We compute $|\mathcal{Q}|$ modulo 2 using a sub-procedure. The consistent cuts are defined so that non-connected candidate solutions $X \in \mathcal{R} \setminus \mathcal{S}$ cancel, because they are consistent with an even number of cuts. Hence, only connected candidates $X \in \mathcal{S}$ remain.

If $|\mathcal{S}|$ is even, then this approach does not work, because the connected solutions would cancel out as well when counting modulo 2. To circumvent this difficulty, we employ the Isolation Lemma (Lemma 2.3). By sampling a weight function $\mathbf{w}\colon U \to [N]$, we can instead count pairs with a fixed weight and it is likely that there is a weight with a unique solution if a solution exists at all. Formally, we compute $|\mathcal{Q}_w|$ modulo 2 for every possible weight w, where $\mathcal{Q}_w = \{(X, C) \in \mathcal{Q} \mid \mathbf{w}(X) = w\}$, instead of computing $|\mathcal{Q}|$ modulo 2.

▶ **Definition 3.1** ([11]). A cut (V_L, V_R) of an undirected graph $G = (V, E)$ is *consistent* if $u \in V_L$ and $v \in V_R$ implies $\{u, v\} \notin E$. A *consistently cut subgraph* of G is a pair $(X, (X_L, X_R))$ such that $X \subseteq V$ and (X_L, X_R) is a consistent cut of $G[X]$. For $V' \subseteq V$, we denote the set of consistently cut subgraphs of $G[V']$ by $\mathcal{C}(V')$.

To ensure that connected solutions are not compatible with an even number of consistent cuts, we will usually force a single vertex to the left side of the consistent cut. This results in the following fundamental property of consistent cuts.

▶ **Lemma 3.2** ($\star$, [11]). *Let X be a subset of vertices such that $v_1 \in X \subseteq V$. The number of consistently cut subgraphs $(X, (X_L, X_R))$ such that $v_1 \in X_L$ is equal to $2^{\mathsf{cc}(G[X])-1}$.*

With Lemma 3.2 we can distinguish disconnected candidates from connected candidates by determining the parity of the number of consistent cuts for the respective candidate. We determine this number not for a single candidate but we determine the total for all candidates with a fixed weight. Corollary 3.3 encapsulates the Cut&Count technique for treedepth.

▶ **Corollary 3.3** ($\star$). *Let $\mathcal{S} \subseteq 2^U$ and $\mathcal{Q} \subseteq 2^{U \times (V \times V)}$ such that the following two properties hold for every weight function $\mathbf{w}\colon U \to [2|U|]$ and target weight $w \in \mathbb{N}$:*

1. $|\{(X, C) \in \mathcal{Q} \mid \mathbf{w}(X) = w\}| \equiv |\{X \in \mathcal{S} \mid \mathbf{w}(X) = w\}|$,
2. *There is an algorithm $\texttt{CountC}(\mathbf{w}, w, \mathcal{T})$ accepting weights $\mathbf{w}\colon U \to [N]$, a target weight w, and an elimination forest $\mathcal{T}$, such that $\texttt{CountC}(\mathbf{w}, w, \mathcal{T}) \equiv |\{(X, C) \in \mathcal{Q} \mid \mathbf{w}(X) = w\}|$.*

Then Algorithm 1 returns **false** *if $\mathcal{S}$ is empty and* **true** *with probability at least $1/2$ otherwise.*

Algorithm 1 Cut&Count.

Input: Set U, elimination forest $\mathcal{T}$, procedure `CountC` accepting $\mathbf{w}\colon U \to [N]$, $w \in \mathbb{N}$

```
for v ∈ U do
    Choose w(v) ∈ [2|U|] uniformly at random;
for w = 1, ..., 2|U|^2 do
    if CountC(w, w, T) ≡ 1 then return true;
return false;
```

We will use the same definitions as Cygan et al. [11] for $\mathcal{Q}$ and $\mathcal{S}$, hence it follows from their proofs that Condition 1 in Corollary 3.3 is satisfied. Our contribution is to provide the counting procedure `CountC` for problems parameterized by treedepth.

Given the sets $\mathcal{S}$, $\mathcal{R}$, and $\mathcal{Q}$, and a weight function $\mathbf{w}\colon U \to [N]$, we will define for every weight w the sets $\mathcal{S}_w = \{X \in \mathcal{S} \mid \mathbf{w}(X) = w\}$, $\mathcal{R}_w = \{X \in \mathcal{R} \mid \mathbf{w}(X) = w\}$, and $\mathcal{Q}_w = \{(X, C) \in \mathcal{Q} \mid \mathbf{w}(X) = w\}$.

3.2 Connected Vertex Cover

Connected Vertex Cover	
Input:	An undirected graph $G = (V, E)$ and an integer k.
Question:	Is there a set $X \subseteq V$, $\lvert X\rvert = k$, such that $G[X]$ is connected and X is a vertex cover of G, i.e., $e \cap X \neq \emptyset$ for all $e \in E$?

In the considered problems, one usually seeks a solution of size at most k. For convenience we choose to look for a solution of size exactly k and solve the other case in the obvious way. We define the objects needed for Cut&Count in the setting of Connected Vertex Cover. We let $U = V$ and define the candidate solutions by $\mathcal{R} = \{X \subseteq V \mid X \text{ is a vertex cover of } G \text{ and } \lvert X\rvert = k\}$, and the solutions are given by $\mathcal{S} = \{X \in \mathcal{R} \mid G[X] \text{ is connected}\}$.

To ensure that a connected solution is consistent with an odd number of cuts, we choose a vertex v_1 that is always forced to the left side of the cut (cf. Lemma 3.2). As we cannot be sure that there is a minimum connected vertex cover containing v_1, we take an edge $\{u, v\} \in E$ and run Algorithm 1 once for $v_1 := u$ and once for $v_1 := v$. Hence, for a fixed choice of v_1 we define the candidate-cut-pairs by $\mathcal{Q} = \{(X, (X_L, X_R)) \in \mathcal{C}(V) \mid X \in \mathcal{R} \text{ and } v_1 \in X_L\}$. We must check that these definitions satisfy the requirements of Corollary 3.3.

▶ **Lemma 3.4** ($\star$, [11]). *Let $\mathbf{w}\colon V \to [N]$ be a weight function, and let $\mathcal{Q}$ and $\mathcal{S}$ be as defined above. Then we have for every $w \in \mathbb{N}$ that $\lvert\mathcal{S}_w\rvert \equiv \lvert\mathcal{Q}_w\rvert$.*

Next, we describe the procedure `CountC` for Connected Vertex Cover.

Algorithm 2 `CountC` for Connected Vertex Cover.

Input: Elimination forest $\mathcal{T}$, weights $\mathbf{w}\colon V \to [2n]$, target weight $w \in [2n^2]$

1 Let r denote the root of $\mathcal{T}$;
2 $P :=$ `calc_poly_inc`$(r, \emptyset)$;
3 **return** the coefficient of $Z_W^w Z_X^k$ in P;

Algorithm 3 `calc_poly_exc`(v, f).

Input: Elimination forest $\mathcal{T}$, weights $\mathbf{w}\colon V \to [2n]$, vertex $v \in V$, previous choices $f\colon$ `tail`$[v] \to \{\mathbf{1}_L, \mathbf{1}_R, \mathbf{0}\}$

1 **if** *v is a leaf of $\mathcal{T}$* **then return** the result of equation (1);
2 **else**
3 $P := 1$;
4 **for** $u \in$ `child`(v) **do** `// cf. equation (2)`
5 $P := P \cdot$ `calc_poly_inc`(v, f);
6 **return** P;

▶ **Lemma 3.5.** *Given a connected graph $G = (V, E)$, a vertex $v_1 \in V$, an integer k, a weight function $\mathbf{w}\colon V \to [2n]$, and an elimination forest $\mathcal{T}$ of G of depth d, we can determine $\lvert\mathcal{Q}_w\rvert$ modulo 2 for every $0 \leq w \leq 2n^2$ in time $\mathcal{O}^*(3^d)$ and polynomial space. In particular, Algorithm 2 determines $\lvert\mathcal{Q}_w\rvert$ modulo 2 for a specified target weight w in the same time and space.*

Algorithm 4 `calc_poly_inc`(v, g).

Input: Elimination forest $\mathcal{T}$, weights $\mathbf{w}\colon V \to [2n]$, vertex $v \in V$, previous choices $g\colon \texttt{tail}(v) \to \{\mathbf{1}_L, \mathbf{1}_R, \mathbf{0}\}$

1 **for** $s \in \{\mathbf{1}_L, \mathbf{1}_R, \mathbf{0}\}$ **do**
2 $\quad P_s := \texttt{calc\_poly\_exc}(v, g[v \mapsto s])$;
3 **return** $P_{\mathbf{1}_L} Z_W^{\mathbf{w}(v)} Z_X + P_{\mathbf{1}_R} Z_W^{\mathbf{w}(v)} Z_X + P_{\mathbf{0}}$; // cf. equation (3)

Proof. For the discussion of the algorithm, it is convenient to drop the cardinality constraint in $\mathcal{R}$ and $\mathcal{Q}$ and to define these sets for every induced subgraph $G[V']$ of G. Hence, we define for every $V' \subseteq V$ the set $\widehat{\mathcal{R}}(V') = \{X \subseteq V' \mid X \text{ is a vertex cover of } G[V']\}$ and the set $\widehat{\mathcal{Q}}(V') = \{(X, (X_L, X_R)) \in \mathcal{C}(V') \mid X \in \mathcal{R}(V') \text{ and } (v_1 \in V' \to v_1 \in X_L)\}$.

Similar to Pilipczuk and Wrochna [29], our algorithm will compute a multivariate polynomial in the formal variables Z_W and Z_X, where the coefficient of $Z_W^w Z_X^i$ is the cardinality of $\widehat{\mathcal{Q}}_w^i(V) = \{(X, C) \in \widehat{\mathcal{Q}}(V) \mid \mathbf{w}(X) = w, |X| = i\}$ modulo 2, i.e., the formal variables track the weight and size of candidate solutions. In particular, we have that $\widehat{\mathcal{Q}}_w^k = \mathcal{Q}_w$ for every w. Polynomials act as an appropriate data structure, because addition and multiplication of polynomials naturally updates the weight and size trackers correctly.

The output polynomial is computed by a branching algorithm (see Algorithm 2) that starts at the root r of the elimination forest $\mathcal{T}$ and proceeds downwards to the leaves. At every vertex we branch into several states, denoted $\texttt{states} = \{\mathbf{1}_L, \mathbf{1}_R, \mathbf{0}\}$. The interpretation of the states $\mathbf{1}_L$ and $\mathbf{1}_R$ is that the vertex is inside the vertex cover and the subscript denotes to which side of the consistent cut it belongs. Vertices that do not belong to the vertex cover have state $\mathbf{0}$.

For each vertex v there are multiple subproblems on $G[\texttt{broom}[v]]$. When solving a subproblem, we need to take into account the choices that we have already made, i.e., the branching decisions for the ancestors of v. At each vertex we compute two different types of polynomials, which correspond to two different kinds of partial solutions. Those that are subsets of $\texttt{tree}(v)$ and respect the choices made on $\texttt{tail}[v]$ and those that are subsets of $\texttt{tree}[v]$ and respect the choices made on $\texttt{tail}(v)$. Distinguishing these two types of partial solutions is important when v has multiple children in $\mathcal{T}$. Formally, the previous branching decisions are described by assignments f or g from $\texttt{tail}[v]$ or $\texttt{tail}(v)$ to $\{\mathbf{1}_L, \mathbf{1}_R, \mathbf{0}\}$ respectively.

For every vertex v and assignment $f\colon \texttt{tail}[v] \to \{\mathbf{1}_L, \mathbf{1}_R, \mathbf{0}\}$ we define the partial solutions at v, but excluding v, that respect f by

$$\mathcal{P}_{(v)}(f) = \{(X, (X_L, X_R)) \in \mathcal{C}(\texttt{tree}(v)) \mid X' = X \cup f^{-1}(\{\mathbf{1}_L, \mathbf{1}_R\}), \\ C' = (X_L \cup f^{-1}(\mathbf{1}_L), X_R \cup f^{-1}(\mathbf{1}_R)), (X', C') \in \widehat{\mathcal{Q}}(\texttt{broom}[v])\}.$$

So, $\mathcal{P}_{(v)}(f)$ consists of consistently cut subgraphs $(X, (X_L, X_R))$ of $G[\texttt{tree}(v)]$ that are extended by f to valid candidate-cut-pairs (X', C') for $G[\texttt{broom}[v]]$, meaning that X' is a vertex cover of $G[\texttt{broom}[v]]$ and C' is a consistent cut of X'.

Very similarly, for every vertex v and assignment $g\colon \texttt{tail}(v) \to \{\mathbf{1}_L, \mathbf{1}_R, \mathbf{0}\}$ we define the partial solutions at v, possibly including v, that respect g by

$$\mathcal{P}_{[v]}(g) = \{(X, (X_L, X_R)) \in \mathcal{C}(\texttt{tree}[v]) \mid X' = X \cup g^{-1}(\{\mathbf{1}_L, \mathbf{1}_R\}), \\ C' = (X_L \cup g^{-1}(\mathbf{1}_L), X_R \cup g^{-1}(\mathbf{1}_R)), (X', C') \in \widehat{\mathcal{Q}}(\texttt{broom}[v])\}.$$

Thus, for the root r of $\mathcal{T}$ we have $\mathcal{P}_{[r]}(\emptyset) = \widehat{\mathcal{Q}}(V)$.

We keep track of the partial solutions $\mathcal{P}_{(v)}(f)$ and $\mathcal{P}_{[v]}(g)$ using polynomials which we define now. For every vertex v and assignment $f\colon \mathtt{tail}[v] \to \{\mathbf{1}_L, \mathbf{1}_R, \mathbf{0}\}$ we will compute a polynomial $P_{(v)}(f) \in \mathbb{F}_2[Z_W, Z_X]$ where $P_{(v)}(f) = \sum_{w=0}^{2n^2} \sum_{i=0}^{n} c_{w,i} Z_W^w Z_X^i$ and

$$c_{w,i} = |\{(X, C) \in \mathcal{P}_{(v)}(f) \mid \mathbf{w}(X) = w \text{ and } |X| = i\}| \mod 2.$$

Similarly, for every vertex v and assignment $g\colon \mathtt{tail}(v) \to \{\mathbf{1}_L, \mathbf{1}_R, \mathbf{0}\}$ we will compute a polynomial $P_{[v]}(g) \in \mathbb{F}_2[Z_W, Z_X]$ where $P_{[v]}(g) = \sum_{w=0}^{2n^2} \sum_{i=0}^{n} c'_{w,i} Z_W^w Z_X^i$ and

$$c'_{w,i} = |\{(X, C) \in \mathcal{P}_{[v]}(g) \mid \mathbf{w}(X) = w \text{ and } |X| = i\}| \mod 2.$$

Algorithm 2 computes the polynomial $P = P_{[r]}(\emptyset)$, where r is the root of $\mathcal{T}$, and extracts the appropriate coefficient of P. To compute P we employ recurrences for $P_{(v)}(f)$ and $P_{[v]}(g)$. We proceed by describing the recurrence for $P_{(v)}(f)$.

In the case that v is a leaf node in $\mathcal{T}$, i.e., $\mathtt{tree}(v) = \emptyset$, we can compute $P_{(v)}(f)$ by

$$\begin{aligned} P_{(v)}(f) \quad &= \quad [f^{-1}(\{\mathbf{1}_L, \mathbf{1}_R\}) \text{ is a vertex cover of } G[\mathtt{tail}[v]]] \\ &\cdot \quad [(f^{-1}(\mathbf{1}_L), f^{-1}(\mathbf{1}_R)) \text{ is a consistent cut of } G[f^{-1}(\{\mathbf{1}_L, \mathbf{1}_R\})]] \\ &\cdot \quad [v_1 \in \mathtt{tail}[v] \to f(v_1) = \mathbf{1}_L], \end{aligned} \tag{1}$$

which checks whether the assignment f induces a valid partial solution. This is the only step in which we explicitly ensure that we are computing only vertex covers; in all other steps this will not be required. If v is not a leaf, then $P_{(v)}(f)$ is computed by the recurrence

$$P_{(v)}(f) = \prod_{u \in \mathtt{child}(v)} P_{[u]}(f), \tag{2}$$

which combines disjoint partial solutions. The equations (1) and (2) are used by Algorithm 3 to compute the polynomial $P_{(v)}(f)$.

We proceed by giving the recurrence that is used by Algorithm 4 to compute the polynomial $P_{[v]}(g)$:

$$P_{[v]}(g) = P_{(v)}(g[v \mapsto \mathbf{1}_L])\, Z_W^{\mathbf{w}(v)} Z_X + P_{(v)}(g[v \mapsto \mathbf{1}_R])\, Z_W^{\mathbf{w}(v)} Z_X + P_{(v)}(g[v \mapsto \mathbf{0}]). \tag{3}$$

Equation (3) tests all three possible states for v in a candidate-cut-pair and multiplies by $Z_W^{\mathbf{w}(v)} Z_X$ if v is in the vertex cover to update the weight and size of the partial solutions.

Correctness. We will now prove the correctness of equations (1) through (3). First of all, observe that when $v_1 \in \mathtt{tail}[v]$ but $f(v_1) \neq \mathbf{1}_L$ then we must have that $P_{(v)}(f) = 0$; similarly, we must have $P_{[v]}(g) = 0$ when $g(v_1) \neq \mathbf{1}_L$ for $v_1 \in \mathtt{tail}(v)$. This property is ensured by equation (1) and preserved by the recurrences (2) and (3). To see that equation (1) is correct, notice that when v is a leaf node in $\mathcal{T}$ we have that $\mathtt{tree}(v) = \emptyset$ and hence the only consistently cut subgraph of $\mathtt{tree}(v)$ is $(\emptyset, (\emptyset, \emptyset))$. Therefore, we only need to verify whether this is a valid partial solution in $P_{(v)}(f)$, which reduces to the predicate on the right-hand side of (1).

For equations (2) and (3), we have to establish bijections between the objects counted on either side of the respective equation and argue that size and weight are updated correctly. We proceed by proving the correctness of equation (2), which is the only equation where the proof of correctness requires the special properties of elimination forests. We consider any $(X, (X_L, X_R)) \in \mathcal{P}_{(v)}(f)$. We can uniquely partition X into subsets X^u of $\mathtt{tree}[u]$ for $u \in \mathtt{child}(v)$ by setting $X^u = X \cap \mathtt{tree}[u]$. Furthermore, by setting $X_L^u = X_L \cap \mathtt{tree}[u]$ and

$X^u_R = X_R \cap \texttt{tree}[u]$ we obtain $(X^u, (X^u_L, X^u_R)) \in \mathcal{P}_{[u]}(f)$, because we are only restricting the vertex cover $X' = X \cup f^{-1}(\{\mathbf{1}_L, \mathbf{1}_R\})$ and consistent cut $(X_L \cup f^{-1}(\mathbf{1}_L), X_R \cup f^{-1}(\mathbf{1}_R))$ to the induced subgraph $G[\texttt{broom}[u]]$ of $G[\texttt{broom}[v]]$. Vice versa, any combination of partial solutions $(X^u, (X^u_L, X^u_R)) \in \mathcal{P}_{[u]}(f)$ for each $u \in \texttt{child}(v)$ yields a partial solution $(X, (X_L, X_R)) \in \mathcal{P}_{(v)}(f)$ as there are no edges in G between $\texttt{tree}[u]$ and $\texttt{tree}[u']$ for $u \neq u' \in \texttt{child}(v)$ by the properties of an elimination forest. Since the sets X^u partition X, we obtain the size and weight of X by summing over the sizes and weights of the sets X^u respectively. Hence, these values are updated correctly by polynomial multiplication.

It remains to prove the correctness of (3). This time, consider any $(X, (X_L, X_R)) \in \mathcal{P}_{[v]}(g)$. Now, there are three possible cases depending on the state of v in this partial solution.

1. If $v \in X_L \subseteq X$, then we claim that $(X \setminus \{v\}, (X_L \setminus \{v\}, X_R)) \in \mathcal{P}_{(v)}(f)$, where $f = g[v \mapsto \mathbf{1}_L]$. This is true due to the identities $(X \setminus \{v\}) \cup f^{-1}(\{\mathbf{1}_L, \mathbf{1}_R\}) = X \cup g^{-1}(\{\mathbf{1}_L, \mathbf{1}_R\})$, and $(X_L \setminus \{v\}) \cup f^{-1}(\mathbf{1}_L) = X_L \cup g^{-1}(\mathbf{1}_L)$, and $X_R \cup f^{-1}(\mathbf{1}_R) = X_R \cup g^{-1}(\mathbf{1}_R)$, which mean that this implicitly defined mapping preserves the definition of X' and C' in the predicates of $\mathcal{P}_{[v]}(g)$ and $\mathcal{P}_{(v)}(f)$. Vice versa, any partial solution in $\mathcal{P}_{(v)}(f)$ can be extended to such a partial solution in $\mathcal{P}_{(v)}(g)$ by adding v to X_L. Since $|X| - |X \setminus \{v\}| = 1$ and $\mathbf{w}(X) - \mathbf{w}(X \setminus \{v\}) = \mathbf{w}(v)$, multiplication by $Z_W^{\mathbf{w}(v)} Z_X$ updates size and weight correctly.
2. If $v \in X_R \subseteq X$, the proof is analogous to case 1.
3. If $v \notin X$, then we have that $(X, (X_L, X_R)) \in \mathcal{P}_{(v)}(f)$, where $f = g[v \mapsto \mathbf{0}]$. Vice versa, any $(X, (X_L, X_R)) \in \mathcal{P}_{(v)}(f)$ must also be in $\mathcal{P}_{[v]}(g)$. Since X does not change, we do not need to update size or weight and do not multiply by further formal variables in this case.

If $v = v_1$, then equation (3) simplifies to $P_{[v]}(g) = P_{(v)}(g[v \mapsto \mathbf{1}_L]) Z_W^{\mathbf{w}(v)} Z_X$, because $\mathcal{P}_{(v)}(g[v \mapsto \mathbf{1}_R]) = \mathcal{P}_{(v)}(g[v \mapsto \mathbf{0}]) = \emptyset$ and hence only the first case occurs. Note that by establishing these bijections in the proofs of correctness, we have actually shown that equations (1) through (3) are also correct when working in $\mathbb{Z}$ instead of $\mathbb{F}_2$.

Time and Space Analysis. We finish the proof by discussing the time and space requirement. Observe that the coefficients of our polynomials are in $\mathbb{F}_2$ and hence can be added and multiplied in constant time. Furthermore, all considered polynomials consist of at most polynomially many monomials as the weight and size of a candidate solution are polynomial in n. Therefore, we can add and multiply the polynomials in polynomial time and hence compute recurrences (1), (2), and (3) in polynomial time. Every polynomial $P_{(v)}(f)$ and $P_{[v]}(g)$ is computed at most once, because $P_{(v)}(f)$ is only called by $P_{[v]}(g)$ where f is an extension of g, i.e., $f = g[v \mapsto s]$ for some $s \in \texttt{states}$, and $P_{[v]}(g)$ is only called by $P_{(w)}(g)$ where w is the parent of v. Hence, the recurrences only make disjoint calls and no polynomial is computed more than once. For a fixed vertex v there are at most 3^d choices for f and g. Thus, Algorithm 2 runs in time $\mathcal{O}^*(3^d)$ for elimination forests of depth d. Finally, Algorithm 2 requires only polynomial space, because it has a recursion depth of $2d+1$ and every recursive call needs to store at most a constant number of polynomials, which require by the previous discussion only polynomial space each. ◀

▶ **Theorem 3.6.** *There is a Monte-Carlo algorithm that given an elimination forest of depth d for a graph G solves* Connected Vertex Cover *on G in time $\mathcal{O}^*(3^d)$ and polynomial space. The algorithm cannot give false positives and may give false negatives with probability at most 1/2.*

Proof. We pick an edge $\{u, v\} \in E$ and branch on $v_1 := u$ and $v_1 := v$. We run Algorithm 1 with $U = V$ and the procedure `CountC` as given by Algorithm 2. Correctness follows from Corollary 3.3 and Lemma 3.4. Running time and space bound follow from Lemma 3.5. ◀

We remark that calling Algorithm 2 for each target weight $w \in [2n^2]$ (as in Algorithm 1) would redundantly compute the polynomial $P = P_{[r]}(\emptyset)$ several times, although it suffices to compute P once and then look up the appropriate coefficient depending on w.

If one is interested in solving Weighted Connected Vertex Cover, then it is straightforward to adapt our approach to polynomially-sized weights: instead of using Z_X to track the size of the vertex covers, we let it track their cost and change recurrence (3) accordingly.

3.3 Adapting to Other Problems

The high-level structure of the counting procedure for the other problems is very similar to that of Algorithm 2 for Connected Vertex Cover. One possible difference is that we might have to consider the solutions over a more complicated universe U than just the vertex set V. Also, we might want to keep track of more data of the partial solutions and hence use more than just two formal variables for the polynomials. Both of these changes occur for Feedback Vertex Set, which is presented in the next section. The equation for the base case (cf. equation (1)) and the recurrence for $P_{[v]}(g)$ (cf. equation (3)) are also problem-dependent.

Time and Space Analysis. The properties that we require of the polynomials and equations in the time and space analysis, namely that the equations can be evaluated in polynomial time and every polynomial is computed at most once, remain true by the same arguments as for Connected Vertex Cover. The running time essentially results from the number of computed polynomials, which increases when we use more states for the vertices. Again denoting the set of states by `states`, we obtain a running time of $\mathcal{O}^*(|\texttt{states}|^d)$ on elimination forests of depth d. The space analysis also remains valid, because the recursion depth remains $2d + 1$ and for each call we need to store only a constant number of polynomials each using at most polynomial space.

3.4 Feedback Vertex Set

Feedback Vertex Set	
Input:	An undirected graph $G = (V, E)$ and an integer k.
Question:	Is there a set $X \subseteq V$, $\lvert X\rvert = k$, such that $G - X$ is a forest?

Feedback Vertex Set differs from the other problems in that we do not have a positive connectivity requirement, but a negative connectivity requirement, i.e., we need to ensure that the remaining graph is badly connected in the sense that it contains no cycles. Cygan et al. [11] approach this via the well-known Lemma 3.7.

▶ **Lemma 3.7.** *A graph with n vertices and m edges is a forest if and only if it has at most $n - m$ connected components.*

Applying Lemma 3.7 requires that we count how many vertices and edges remain after deleting a set $X \subseteq V$ from G. We do not need to count exactly how many connected components remain, it suffices to enforce that there are not too many connected components. We will achieve this, like Cygan et al. [11], by the use of *marker vertices*. In this case, our solutions are pairs (Y, M) with $M \subseteq Y$, where we interpret Y as the forest that remains after removing a feedback vertex set X and the marked vertices are represented by the set M. To

bound the number of connected components, we want that every connected component of $G[Y]$ contains at least one marked vertex. By forcing the marked vertices to the left side of the cut, we ensure that candidates (Y, M) where $G[Y]$ has a connected component not containing a marked vertex, in particular those with more than $|M|$ connected components, cancel modulo 2. The formal definitions are $\mathcal{R} = \{(Y, M) \mid M \subseteq Y \subseteq V \text{ and } |Y| = n - k\}$, and $\mathcal{S} = \{(Y, M) \in \mathcal{R} \mid G[Y] \text{ is a forest, every connected component of } G[Y] \text{ intersects } M\}$, and $\mathcal{Q} = \{((Y, M), (Y_L, Y_R)) \mid (Y, M) \in \mathcal{R} \text{ and } (Y, (Y_L, Y_R)) \in \mathcal{C}(V) \text{ and } M \subseteq Y_L\}$.

Since our solutions (Y, M) are pairs of two vertex sets, we need a larger universe to make the Isolation Lemma, Lemma 2.3, work. We use $U = V \times \{\mathbf{F}, \mathbf{M}\}$, hence a weight function $\mathbf{w}\colon U \to [N]$ assigns two different weights $\mathbf{w}(v, \mathbf{F})$ and $\mathbf{w}(v, \mathbf{M})$ to a vertex v depending on whether v is marked or not. To make these definitions compatible with Corollary 3.3 we associate to each pair (Y, M) the set $Y \times \{\mathbf{F}\} \cup M \times \{\mathbf{M}\} \subseteq U$, which also allows us to extend the weight function to such pairs (Y, M), i.e. $\mathbf{w}(Y, M) = \mathbf{w}(Y \times \{\mathbf{F}\} \cup M \times \{\mathbf{M}\})$.

▶ **Lemma 3.8** (⋆, [11]). *Let (Y, M) be such that $M \subseteq Y \subseteq V$. The number of consistently cut subgraphs $(Y, (Y_L, Y_R))$ such that $M \subseteq Y_L$ is equal to $2^{\overline{\mathsf{cc}}_M(G[Y])}$, where $\overline{\mathsf{cc}}_M(G[Y])$ is the number of connected components of $G[Y]$ that do not contain any vertex from M.*

To apply Lemma 3.7, we need to distinguish candidates by the number of edges, and markers, in addition to the weight, hence we make the following definitions for $j, \ell, w \in \mathbb{N}$:

$$\begin{aligned}
\mathcal{R}_w^{j,\ell} &= \{(Y, M) \in \mathcal{R} \mid \mathbf{w}(Y, M) = w, \ |E(G[Y])| = j, \ |M| = \ell\},\\
\mathcal{S}_w^{j,\ell} &= \{(Y, M) \in \mathcal{S} \mid \mathbf{w}(Y, M) = w, \ |E(G[Y])| = j, \ |M| = \ell\},\\
\mathcal{Q}_w^{j,\ell} &= \{(Y, M, (Y_L, Y_R)) \in \mathcal{Q} \mid \mathbf{w}(Y, M) = w, \ |E(G[Y])| = j, \ |M| = \ell\}.
\end{aligned}$$

▶ **Lemma 3.9** (⋆, [11]). *Let $\mathbf{w}\colon U \to [N]$ be a weight function, and $\mathcal{Q}$ and $\mathcal{S}$ as defined above. Then we have for every $w \in \mathbb{N}$ and $j \in [n - k - 1]$ that $|\mathcal{S}_w^{j,n-k-j}| \equiv |\mathcal{Q}_w^{j,n-k-j}|$.*

Note that by Lemma 3.7 a Feedback Vertex Set instance has a solution X if and only if there is a choice of $w, j \in \mathbb{N}$ and $M \subseteq Y := V \setminus X$ such that $(Y, M) \in \mathcal{S}_w^{j,n-k-j}$.

▶ **Lemma 3.10.** *Given a connected graph $G = (V, E)$, an integer k, a weight function $\mathbf{w}\colon U \to [4n]$ and an elimination forest $\mathcal{T}$ of G of depth d, we can determine $|\mathcal{Q}_w^{j,n-k-j}|$ modulo 2 for every $0 \leq w \leq 4n^2$, $0 \leq j \leq m$, in time $\mathcal{O}^*(3^d)$ and polynomial space.*

Proof. Again, we drop the cardinality constraints from $\mathcal{R}$ and $\mathcal{Q}$ and define for induced subgraphs $G[V']$ the variants $\widehat{\mathcal{R}}(V') = \{(Y, M) \mid M \subseteq Y \subseteq V'\}$ and $\widehat{\mathcal{Q}}(V') = \{((Y, M), (Y_L, Y_R)) \mid (Y, M) \in \widehat{\mathcal{R}}(V') \text{ and } (Y, (Y_L, Y_R)) \in \mathcal{C}(V') \text{ and } M \subseteq Y_L\}$.

We will compute a multivariate polynomial in the formal variables Z_W, Z_Y, Z_E, Z_M, where the coefficient of $Z_W^w Z_Y^i Z_E^j Z_M^\ell$ is the cardinality modulo 2 of

$$\widehat{\mathcal{Q}}_w^{i,j,\ell} = \{((Y, M), C) \in \widehat{\mathcal{Q}}(V) \mid \mathbf{w}(Y, M) = w, |Y| = i, |E(G[Y])| = j, |M| = \ell\}.$$

The coefficients of $Z_W^w Z_Y^{n-k} Z_E^j Z_M^{n-k-j}$ for every w and j then yield the desired numbers.

For Feedback Vertex Set we require three states which are given by $\texttt{states} = \{\mathbf{1}, \mathbf{0}_L, \mathbf{0}_R\}$. The state $\mathbf{1}$ represents vertices inside the feedback vertex set; the states $\mathbf{0}_L$ and $\mathbf{0}_R$ represent vertices inside the remaining forest and the subscript denotes to which side of the consistent cut a vertex belongs. Perhaps surprisingly, there is no state to represent marked vertices. It turns out that it is not important which vertices are marked; it is sufficient to know the number of marked vertices.

For every vertex v and assignment $f\colon \texttt{tail}[v] \to \{\mathbf{1}, \mathbf{0}_L, \mathbf{0}_R\}$ we define the partial solutions at v, but excluding v, that respect f by

$$\mathcal{P}_{(v)}(f) = \{((Y, M), (Y_L, Y_R)) \in \widehat{\mathcal{Q}}(\texttt{tree}(v)) \mid Y' = Y \cup f^{-1}(\{\mathbf{0}_L, \mathbf{0}_R\}),\\ C' = (Y_L \cup f^{-1}(\mathbf{0}_L), Y_R \cup f^{-1}(\mathbf{0}_R)), ((Y', M), C') \in \widehat{\mathcal{Q}}(\texttt{broom}[v])\}.$$

The partial solutions in $\mathcal{P}_{(v)}(f)$ are consistently cut subgraphs $(Y, (Y_L, Y_R))$ of $G[\texttt{tree}(v)]$ where a subset M of the left side is marked and the extension to (Y', C') by f is a consistently cut subgraph of $G[\texttt{broom}[v]]$.

Similarly, for every vertex v and assignment $g\colon \texttt{tail}(v) \to \{\mathbf{1}, \mathbf{0}_L, \mathbf{0}_R\}$ we define the partial solutions at v, possibly including v, that respect g by

$$\mathcal{P}_{[v]}(g) = \{((Y, M), (Y_L, Y_R)) \in \widehat{\mathcal{Q}}(\texttt{tree}[v]) \mid Y' = Y \cup g^{-1}(\{\mathbf{0}_L, \mathbf{0}_R\}),\\ C' = (Y_L \cup g^{-1}(\mathbf{0}_L), Y_R \cup g^{-1}(\mathbf{0}_R)), ((Y', M), C') \in \widehat{\mathcal{Q}}(\texttt{broom}[v])\}.$$

For every vertex v and assignment $f\colon \texttt{tail}[v] \to \{\mathbf{1}, \mathbf{0}_L, \mathbf{0}_R\}$ we will compute a polynomial $P_{(v)}(f) \in \mathbb{F}_2[Z_W, Z_Y, Z_E, Z_M]$ where the coefficient of $Z_W^w Z_Y^i Z_E^j Z_M^\ell$ in $P_{(v)}(f)$ is given by

$$|\{((Y, M), C) \in \mathcal{P}_{(v)}(f) \mid \mathbf{w}(Y, M) = w, |Y| = i, |E(G[Y])| = j, |M| = \ell\}| \mod 2.$$

For every vertex v and assignment $g\colon \texttt{tail}(v) \to \{\mathbf{1}, \mathbf{0}_L, \mathbf{0}_R\}$ we will compute a polynomial $P_{[v]}(g) \in \mathbb{F}_2[Z_W, Z_Y, Z_E, Z_M]$ where the coefficient of $Z_W^w Z_Y^i Z_E^j Z_M^\ell$ in $P_{[v]}(g)$ is given by

$$|\{((Y, M), C) \in \mathcal{P}_{[v]}(g) \mid \mathbf{w}(Y, M) = w, |Y| = i, |E(G[Y])| = j, |M| = \ell\}| \mod 2.$$

The exponents of the monomials $Z_W^w Z_Y^i Z_E^j Z_M^\ell$ in $P_{(v)}(f)$ and $P_{[v]}(g)$ range between $0 \leq w \leq 4n^2$, $0 \leq i \leq n$, $0 \leq j \leq m$, and $0 \leq \ell \leq n$.

We now present the recurrences used to compute the polynomials $P_{(v)}(f)$ and $P_{[v]}(g)$. If v is a leaf node in $\mathcal{T}$, then we can compute $P_{(v)}(f)$ by

$$P_{(v)}(f) = [(f^{-1}(\mathbf{0}_L), f^{-1}(\mathbf{0}_R)) \text{ is a consistent cut of } G[f^{-1}(\{\mathbf{0}_L, \mathbf{0}_R\})]]. \tag{4}$$

If v is not a leaf node, then we compute $P_{(v)}(f)$ by

$$P_{(v)}(f) = \prod_{u \in \texttt{child}(v)} P_{[u]}(f). \tag{5}$$

To compute $P_{[v]}(g)$ we use the recurrence

$$\begin{aligned} P_{[v]}(g) \quad &= P_{(v)}(g[v \to \mathbf{1}]) \\ &+ P_{(v)}(g[v \to \mathbf{0}_L]) \quad Z_W^{\mathbf{w}(v,\mathbf{F})} \quad Z_Y \quad Z_E^{|N(v) \cap \texttt{tree}[v]|} \\ &+ P_{(v)}(g[v \to \mathbf{0}_L]) \quad Z_W^{\mathbf{w}(v,\mathbf{F}) + \mathbf{w}(v,\mathbf{M})} \quad Z_Y \quad Z_E^{|N(v) \cap \texttt{tree}[v]|} \quad Z_M \\ &+ P_{(v)}(g[v \to \mathbf{0}_R]) \quad Z_W^{\mathbf{w}(v,\mathbf{F})} \quad Z_Y \quad Z_E^{|N(v) \cap \texttt{tree}[v]|}. \end{aligned} \tag{6}$$

This recurrence tests all three possible states for the vertex v and whether it is marked. In the last case v has state $\mathbf{0}_L$, but the formal variables Z_W and Z_M must be updated differently from the case where v is not marked but has state $\mathbf{0}_L$.

We will now prove the correctness of the equations (4) to (6). For the correctness of equation (4), notice that $\texttt{tree}(v) = \emptyset$ when v is a leaf. Hence, $\widehat{\mathcal{Q}}(\texttt{tree}(v))$ degenerates to $\{((\emptyset, \emptyset), (\emptyset, \emptyset))\}$ and we must check whether $((\emptyset, \emptyset), (\emptyset, \emptyset)) \in \mathcal{P}_{(v)}(f)$ which means that $((f^{-1}(\{\mathbf{0}_L, \mathbf{0}_R\}), \emptyset), (f^{-1}(\mathbf{0}_L), f^{-1}(\mathbf{0}_R))) \in \widehat{\mathcal{Q}}(\texttt{broom}[v])$ and checking the consistency of the cut is the only nontrivial requirement in this case.

The proof of correctness for equation (5) is similar to the proof for equation (2) of Connected Vertex Cover. Any solution in $\mathcal{P}_{(v)}(f)$ uniquely partitions into solutions in $\mathcal{P}_{[u]}(f)$ for each $u \in \texttt{child}(v)$. Vice versa, any combination of solutions for the children u of v yields a unique solution in $\mathcal{P}_{(v)}(f)$. The properties of an elimination forest are needed to show that the union of consistent cuts remains a consistent cut. We omit further details.

To prove the correctness of equation (6) we consider a partial solution $((Y, M), (Y_L, Y_R)) \in \mathcal{P}_{[v]}(g)$ and distinguish between four cases depending on the state of v.

1. If $v \notin Y$, then $((Y, M), (Y_L, Y_R)) \in \mathcal{P}_{(v)}(f)$, where $f = g[v \mapsto \mathbf{1}]$, because there is no constraint involving vertices with state $\mathbf{1}$. Vice versa, we have that any partial solution $((Y, M), (Y_L, Y_R)) \in \mathcal{P}_{(v)}(f)$ must also be in $\mathcal{P}_{[v]}(g)$. Since Y and M do not change, we do not need to multiply by further formal variables.
2. If $v \in Y_L \subseteq Y$ and $v \notin M$, then $((Y \setminus \{v\}, M), (Y_L \setminus \{v\}, Y_R)) \in \mathcal{P}_{(v)}(f)$, where $f = g[v \mapsto \mathbf{0}_L]$, because the definition of Y' and C' in the predicate in the definition of $\mathcal{P}_{[v]}(g)$ and $\mathcal{P}_{(v)}(f)$ do not change. Hence, we can also extend any partial solution of $\mathcal{P}_{(v)}(f)$ to such a partial solution of $\mathcal{P}_{[v]}(g)$ by adding v to Y_L. The number of vertices in Y increase by 1, $|E(G[Y])|$ increases by $|N(v) \cap \texttt{tree}[v]|$, and the weight increases by $\mathbf{w}(v, \mathbf{F})$. Therefore, multiplication with $Z_W^{\mathbf{w}(v,\mathbf{F})} Z_Y Z_E^{|N(v)\cap \texttt{tree}[v]|}$ is the correct update.
3. If $v \in M \subseteq Y_L \subseteq Y$, then $((Y \setminus \{v\}, M \setminus \{v\}), (Y_L \setminus \{v\}, Y_R)) \in \mathcal{P}_{(v)}(f)$, where $f = g[v \mapsto \mathbf{0}_L]$. The argument is similar to case 2. Note that, again, the definition of Y' and C' do not change in the predicates. The set of marked vertices M does change, but we only need to ensure that M remains a subset of the left side of the cut, which we do by removing v from M. In addition to the changes in the number of vertices and edges from case 2, the number of marked vertices has increased by 1 and the weight increases by an additional $\mathbf{w}(v, \mathbf{M})$, to keep track of these changes we further multiply by $Z_W^{\mathbf{w}(v,\mathbf{M})} Z_M$.
4. If $v \in Y_R \subseteq Y$, then the proof is analogous to case 2.

The running time and space bound follows from the general discussion in Section 3.3. ◀

▶ **Theorem 3.11.** *There exists a Monte-Carlo algorithm that given an elimination forest of depth d solves Feedback Vertex Set in time $\mathcal{O}^*(3^d)$ and polynomial space. The algorithm cannot give false positives and may give false negatives with probability at most 1/2.*

Proof. We set $U = V \times \{\mathbf{F}, \mathbf{M}\}$, but we need to slightly adapt the definition of $\mathcal{S}$ and $\mathcal{Q}$ to be able to apply Corollary 3.3. We define $\widetilde{\mathcal{S}} = \cup_{j=0}^{n-k-1} \cup_{w=0}^{4n^2} \mathcal{S}_w^{j,n-k-j}$ and $\widetilde{\mathcal{Q}} = \cup_{j=0}^{n-k-1} \cup_{w=0}^{4n^2} \mathcal{Q}_w^{j,n-k-j}$. Note that $\mathcal{S}$ is nonempty if and only if $\widetilde{\mathcal{S}}$ is nonempty by Lemma 3.7. The procedure `CountC` is given by running the algorithm from Lemma 3.10 and for a given target weight w adding up (modulo 2) the values of $|\mathcal{Q}_w^{j,n-k-j}|$ for $j = 0, \ldots, n-k-1$, thereby obtaining the cardinality of $\widetilde{\mathcal{Q}}_w$ modulo 2. The desired algorithm is then given by running Algorithm 1. The correctness follows from Lemma 3.9 and Corollary 3.3 with $\widetilde{\mathcal{S}}$ and $\widetilde{\mathcal{Q}}$ instead of $\mathcal{S}$ and $\mathcal{Q}$. The running time and space bound follows from Lemma 3.10. ◀

Theorem 3.11 allows us to easily reobtain a result by Cygan et al. [11] on Feedback Vertex Set parameterized by Feedback Vertex Set. Recently, this result has been superseded by results of Li and Nederlof [21]; they present an $\mathcal{O}^*(2.7^k)$-time and exponential-space algorithm and an $\mathcal{O}^*(2.8446^k)$-time and polynomial-space algorithm for this problem.

▶ **Corollary 3.12** (⋆, [11]). *There is a Monte-Carlo algorithm that given a feedback vertex set of size s solves Feedback Vertex Set in time $\mathcal{O}^*(3^s)$ and polynomial space. The algorithm cannot give false positives and may give false negatives with probability at most 1/2.*

4 Conclusion

The Cut&Count technique of Cygan et al. [11] has provided single-exponential-time and -space algorithms for many connectivity problems parameterized by treewidth. We have shown that this technique is just as useful for parameterization by treedepth, where we have obtained single-exponential-time and *polynomial-space* algorithms. Our algorithms run in time $\mathcal{O}^*(\alpha^d)$, where α is a small constant and d is the depth of a given elimination forest. The base α matches that obtained by Cygan et al. [11] for parameterization by treewidth. Assuming SETH, this base is optimal for treewidth, or even pathwidth [11]. In principle, since treedepth is a larger parameter than both treewidth and pathwidth, it may be possible to obtain better running times when parameterizing by treedepth, possibly at the cost of using exponential space. The style of construction, used to obtain lower bounds relative to treewidth, used by Lokshtanov et al. [22] and Cygan et al. [11], necessitates long paths and is thereby unsuitable for bounds relative to treedepth. Thus, the question remains whether our running times are optimal; it is tempting to conjecture that they are.

While we have not given the proofs, our techniques also extend to other problems like Connected Feedback Vertex Set and Connected Total Dominating Set. However, there are several problems, including Cycle Cover and Longest Cycle, for which Cygan et al. [11] obtain efficient algorithms, where it is yet unclear how to solve them in polynomial space when parameterizing by treedepth. In particular, Hamiltonian Path and Hamiltonian Cycle share the same issues, namely that the algorithms parameterized by treewidth keep track of the degrees in the partial solutions and it is not clear how to do that when branching on the elimination forest while only using polynomial space. Belbasi and Fürer [2] can count Hamiltonian cycles in polynomial space, but their running time also depends on the width of a given tree decomposition. An algorithm for any of these problems parameterized by treedepth, with single-exponential running time and requiring only polynomial space, would be quite interesting.

References

1 Jochen Alber and Rolf Niedermeier. Improved tree decomposition based algorithms for domination-like problems. In Sergio Rajsbaum, editor, *LATIN 2002: Theoretical Informatics, 5th Latin American Symposium, Cancun, Mexico, April 3-6, 2002, Proceedings*, volume 2286 of *Lecture Notes in Computer Science*, pages 613–628. Springer, 2002. doi:10.1007/3-540-45995-2_52.

2 Mahdi Belbasi and Martin Fürer. A space-efficient parameterized algorithm for the Hamiltonian cycle problem by dynamic algebraization. In René van Bevern and Gregory Kucherov, editors, *Computer Science - Theory and Applications - 14th International Computer Science Symposium in Russia, CSR 2019, Novosibirsk, Russia, July 1-5, 2019, Proceedings*, volume 11532 of *Lecture Notes in Computer Science*, pages 38–49. Springer, 2019. doi:10.1007/978-3-030-19955-5_4.

3 Benjamin Bergougnoux and Mamadou Moustapha Kanté. Fast exact algorithms for some connectivity problems parameterized by clique-width. *Theor. Comput. Sci.*, 782:30–53, 2019. doi:10.1016/j.tcs.2019.02.030.

4 Benjamin Bergougnoux and Mamadou Moustapha Kanté. More applications of the d-neighbor equivalence: Connectivity and acyclicity constraints. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany.*, volume 144 of *LIPIcs*, pages 17:1–17:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.17.

5 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.

6 Hans L. Bodlaender, Jitender S. Deogun, Klaus Jansen, Ton Kloks, Dieter Kratsch, Haiko Müller, and Zsolt Tuza. Rankings of graphs. *SIAM J. Discrete Math.*, 11(1):168–181, 1998. doi:10.1137/S0895480195282550.

7 Hans L. Bodlaender, John R. Gilbert, Hjálmtyr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *J. Algorithms*, 18(2):238–255, 1995. doi:10.1006/jagm.1995.1009.

8 Li-Hsuan Chen, Felix Reidl, Peter Rossmanith, and Fernando Sánchez Villaamil. Width, depth, and space: Tradeoffs between branching and dynamic programming. *Algorithms*, 11(7):98, 2018. doi:10.3390/a11070098.

9 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990.

10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

11 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.23.

12 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

13 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.

14 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Representative sets of product families. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 443–454. Springer, 2014. doi:10.1007/978-3-662-44777-2_37.

15 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 142–151. SIAM, 2014. doi:10.1137/1.9781611973402.10.

16 Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic*, 130(1-3):3–31, 2004. doi:10.1016/j.apal.2004.01.007.

17 Martin Fürer and Huiwen Yu. Space saving by dynamic algebraization based on tree-depth. *Theory Comput. Syst.*, 61(2):283–304, 2017. doi:10.1007/s00224-017-9751-3.

18 Falko Hegerfeld and Stefan Kratsch. Solving connectivity problems parameterized by treedepth in single-exponential time and polynomial space. *arXiv e-prints*, page arXiv:2001.05364, January 2020. arXiv:2001.05364.

19 Meir Katchalski, William McCuaig, and Suzanne M. Seager. Ordered colourings. *Discrete Mathematics*, 142(1-3):141–154, 1995. doi:10.1016/0012-365X(93)E0216-Q.

20 Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994. doi:10.1007/BFb0045375.

21 Jason Li and Jesper Nederlof. Detecting feedback vertex sets of size k in $\mathcal{O}^*(2.7^k)$ time. *CoRR*, abs/1906.12298, 2019. arXiv:1906.12298.

22 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.

23 Daniel Lokshtanov and Jesper Nederlof. Saving space by algebraization. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC*

2010, Cambridge, Massachusetts, USA, 5-8 June 2010, pages 321–330. ACM, 2010. doi:10.1145/1806689.1806735.

24 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987. doi:10.1007/BF02579206.

25 Jaroslav Nešetřil and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006. doi:10.1016/j.ejc.2005.01.010.

26 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 Jaroslav Nešetřil and Patrice Ossona de Mendez. On low tree-depth decompositions. *Graphs and Combinatorics*, 31(6):1941–1963, 2015. doi:10.1007/s00373-015-1569-7.

28 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. doi:10.1093/ACPROF:OSO/9780198566076.001.0001.

29 Michał Pilipczuk and Marcin Wrochna. On space efficiency of algorithms working on structural decompositions of graphs. *TOCT*, 9(4):18:1–18:36, 2018. doi:10.1145/3154856.

30 Willem J. A. Pino, Hans L. Bodlaender, and Johan M. M. van Rooij. Cut and count and representative sets on branch decompositions. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPIcs*, pages 27:1–27:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.27.

31 Jan Arne Telle and Andrzej Proskurowski. Practical algorithms on partial k-trees with an application to domination-like problems. In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, Nicola Santoro, and Sue Whitesides, editors, *Algorithms and Data Structures, Third Workshop, WADS '93, Montréal, Canada, August 11-13, 1993, Proceedings*, volume 709 of *Lecture Notes in Computer Science*, pages 610–621. Springer, 1993. doi:10.1007/3-540-57155-8_284.

32 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009. doi:10.1007/978-3-642-04128-0_51.

33 Johan M. M. van Rooij, Hans L. Bodlaender, Erik Jan van Leeuwen, Peter Rossmanith, and Martin Vatshelle. Fast dynamic programming on graph decompositions. *CoRR*, abs/1806.01667, 2018. arXiv:1806.01667.

34 Gerhard J. Woeginger. Space and time complexity of exact algorithms: Some open problems (invited talk). In Rodney G. Downey, Michael R. Fellows, and Frank K. H. A. Dehne, editors, *Parameterized and Exact Computation, First International Workshop, IWPEC 2004, Bergen, Norway, September 14-17, 2004, Proceedings*, volume 3162 of *Lecture Notes in Computer Science*, pages 281–290. Springer, 2004. doi:10.1007/978-3-540-28639-4_25.

Non-Rectangular Convolutions and (Sub-)Cadences with Three Elements

Mitsuru Funakoshi
Department of Informatics, Kyushu University, Japan
mitsuru.funakoshi@inf.kyushu-u.ac.jp

Julian Pape-Lange
Technische Universität Chemnitz, Straße der Nationen 62, 09111 Chemnitz, Germany
julian.pape-lange@informatik.tu-chemnitz.de

Abstract

The discrete acyclic convolution computes the $2n + 1$ sums

$$\sum_{\substack{i+j=k \\ (i,j)\in[0,1,2,\ldots,n]^2}} a_i b_j$$

in $\mathcal{O}(n \log n)$ time. By using suitable offsets and setting some of the variables to zero, this method provides a tool to calculate all non-zero sums

$$\sum_{\substack{i+j=k \\ (i,j)\in P\cap\mathbb{Z}^2}} a_i b_j$$

in a rectangle P with perimeter p in $\mathcal{O}(p \log p)$ time.

This paper extends this geometric interpretation in order to allow arbitrary convex polygons P with k vertices and perimeter p. Also, this extended algorithm only needs $\mathcal{O}\left(k + p(\log p)^2 \log k\right)$ time.

Additionally, this paper presents fast algorithms for counting sub-cadences and cadences with 3 elements using this extended method.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Combinatorics on words; Theory of computation → Computational geometry; Computing methodologies → Number theory algorithms

Keywords and phrases discrete acyclic convolutions, string-cadences, geometric algorithms, number theoretic transforms

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.30

Related Version The preprint of this paper can be found at `https://arxiv.org/abs/1910.11564`.

Acknowledgements The first author discovered an error in the algorithm for determining the existence of 3-cadences in "String cadences" of Amir et al., which led to false positives. After that, the first author reported this error to Travis Gagie, one of the authors of "String Cadences". Travis Gagie explained this error to the second author during the 30th Annual Symposium on Combinatorial Pattern Matching (CPM 2019) in Pisa. He also claimed that it should be possible to determine the existence of 3-cadences in sub-quadratic time. Juliusz Straszyński showed during the same conference that 3-sub-cadences beginning and ending in given intervals can efficiently be detected by convolution. Amihood Amir noted later in an email that we can also efficiently count these sub-cadences, which allows "subtractive" methods as used for arbitrary triangles.

1 Introduction

The convolution is a well-known and very useful method, which is not only closely linked to signal processing (e.g. [18]) but is also used to multiply polynomials (see [5, p. 905]) and large numbers (e.g. [17] (written in German)) in quasi-linear time. The convolution can be efficiently computed with the fast Fourier transform or its counterpart in residue class rings, the number theoretic transform:

▶ **Theorem 1.** *Let $a = (a_0, a_1, a_2, \ldots, a_n)$ and $b = (b_0, b_1, b_2, \ldots, b_n)$ be two sequences. The sequence $c = (c_0, c_1, c_2, \ldots, c_{2n})$ with $c_k = \sum_{i+j=k} (a_i b_j)$ can be computed in $\mathcal{O}(n \log n)$ operations.*

The most well-known proofs use additions and multiplications of arbitrary complex numbers. However, with the finite register lengths of real-world computers, one must either cope with the roundoff errors or do all calculations in a different ring. In Appendix A, we show that a suitable ring can be found deterministically in $\mathcal{O}\left(n(\log n)^2(\log\log n)\right)$ time if the generalized Riemann hypothesis is true.

The convolution can also be interpreted geometrically: Let $a = (a_0, a_1, a_2, \ldots, a_n)$ and $b = (b_0, b_1, b_2, \ldots, b_n)$ be sequences. Then the convolution calculates the partial sums

$$\sum_{\substack{i+j=k \\ (i,j)\in P\cap\mathbb{Z}^2}} a_i b_j,$$

where P is the square given by $\{(x, y) : 0 \leq x, y \leq n\}$.

This paper extends this geometric interpretation and shows that if P is an arbitrary convex polygon with k vertices and perimeter p, the partial sums can be calculated in $\mathcal{O}\left(k + p(\log p)^2 \log k\right)$ time.

We also use this extended method to solve an open problem of a string pattern called cadence. A cadence is given by an arithmetic progression of occurrences of the same character in a string such that the progression cannot be extended to either side without extending the string as well. For example, in the string 001001001 the indices $(3, 6, 9)$ corresponding to the "1"s form a 3-cadence. On the other hand, in the string 001010100 the indices $(3, 5, 7)$ corresponding to the "1"s do not form a 3-cadence since, for example, the index 1 is still inside of the string.

3-cadences can be found naïvely in quadratic time. In the paper [2], a quasi-linear time algorithm for detecting the existence of 3-cadences was proposed, but this algorithm also detects false positives as the aforementioned string 001010100.

This paper fixes this issue and also extends the algorithm to the slightly more general notion of (a, b, c)-partial-k-cadences. The resulting extended algorithm also allows counting those partial-cadences with a given character of an alphabet Σ of a string with length n and only needs $\mathcal{O}\left(n(\log n)^2\right)$ time. Using a method presented by Amir et al. in [2], this implies that all (a, b, c)-partial-k-cadences can be counted in $\mathcal{O}\left(\min(|\Sigma|n(\log n)^2, n^{3/2}\log n)\right)$ time.

Furthermore, we show that the output of the counting algorithm also allows finding x partial-cadences in $\mathcal{O}(xn)$ time.

This paper also gives similar results for 3-sub-cadences.

For the time complexity, we assume that arithmetic operations with $\mathcal{O}(\log n)$ bits can be done in constant time. In particular, we want to be able to get the remainder of a division by a prime $p < 2(2n\log(2n))^2$ in constant time.

Also, in this paper, we assume a suitable alphabet. I.e. the characters are given by sufficiently small integers in order to allow constant time reading of a given character in the string and in order to allow sorting the characters.

2 (Sub-)Cadences and Their Definitions

While the concept of cadences in the context of strings was already considered in [19] by Van der Waerden, the term cadence dates back to 1964 and was first introduced by Gardelle and Guilbaud in [8] (written in French). Since then, there were at least two other, slightly different and non-equivalent definitions given by Lothaire in [15, Chapter 3.3] and Amir et al. in [2].

This paper uses the most restrictive definition of the cadence, which was introduced by Amir et al. in [2], and also uses their definition of the sub-cadence, which is equivalent to Gardelle's cadence in [8] and Lothaire's arithmetic cadence in [15, Chapter 3.3].

A string S of length n is the concatenation $S = S[1..n] = S[1]S[2]S[3]\dots S[n]$ of characters from an alphabet Σ.

▶ **Definition 2.** *A* k-sub-cadence *is a triple* (i, d, k) *of positive integers such that*

$$S[i] = S[i+d] = S[i+2d] = \dots = S[i+(k-1)d]$$

holds.

In this paper, cadences are additionally required to start and end close to the boundaries of the string:

▶ **Definition 3.** *A* k-cadence *is a* k*-sub-cadence* (i, d, k) *such that the inequalities* $i - d \leq 0$ *and* $n < i + kd$ *hold.*

Since for any k-sub-cadence the inequality $i + (k-1)d \leq n$ holds, for any k-cadence $i + (k-1)d \leq n < i + kd$ holds. This implies $k - 1 \leq \frac{n-i}{d} < k$ and thereby $k = \lfloor \frac{n-i}{d} \rfloor + 1$. It is therefore sufficient to omit the variable k of a k-cadence (i, d, k) and just denote this k-cadence by the pair (i, d).

▶ Remark 4 (Comparison of the Definitions).

- The cadence as defined by Lothaire is just an ordered sequence of unequal indices such that the corresponding characters are equal.
- The cadence as defined by Gardelle and Guilbaud additionally requires the sequence to be an arithmetic sequence.
- The cadence as defined by Amir et al. and as used in this paper additionally requires that the cadence cannot be extended in any direction without extending the string as well.

For the analysis of cadences with errors, we need two more definitions:

▶ **Definition 5.** *A* k-cadence with at most m errors *is a tuple* (i, d, k, m) *of integers such that* $i, d, k \geq 1$ *and* $i - d \leq 0$ *and* $n < i + kd$ *hold and such that there are* $k - m$ *different integers* $\pi_j \in \{0, 1, 2, \dots, k-1\}$ *with* $j = 1, 2, 3, \dots, k - m$ *and*

$$S[i + \pi_1 d] = S[i + \pi_2 d] = S[i + \pi_3 d] \dots = S[i + \pi_{k-m} d].$$

A particularly interesting case of cadences with errors is given by the partial-cadences in which we know all positions where an error is allowed:

▶ **Definition 6.** *For some different integers* $\pi_j \in \{0, 1, 2, \dots, k-1\}$ *with* $j = 1, 2, 3, \dots, p$, *a* $(\pi_1, \pi_2, \pi_3, \dots, \pi_p)$-partial-$k$-cadence *is a triple* (i, d, k) *of positive integers with* $i - d \leq 0$ *and* $n < i + kd$ *such that*

$$S[i + \pi_1 d] = S[i + \pi_2 d] = S[i + \pi_3 d] \dots = S[i + \pi_p d]$$

hold.

In this paper, we will only consider the case of $k - 3$ errors. I.e. k*-cadences with at most* $k - 3$ *errors* and (a, b, c)*-partial-*k*-cadences* for three different integer $a, b, c \in \{0, 1, ..., k-1\}$.

3 3-Sub-Cadences and Rectangular Convolutions

It is a direct consequence of van der Waerden's theorem that sufficiently large strings are guaranteed to have sub-cadences of a given length:

▶ **Theorem 7** (Existence of sub-cadences (Van der Waerden [19] (written in German), see Lothaire [15, Chapter 3.3])).

Let Σ be an alphabet and k an integer. There exists an integer $N = N(|\Sigma|, k)$ such that every string containing at least N characters has at least one k-sub-cadence

However, this theorem does not provide the number of k-sub-cadences of a given string.

In this section, we will show that 3-sub-cadences with a given character of a string of length n can be efficiently counted in $\mathcal{O}(n \log n)$ time. We will also show that arbitrary 3-sub-cadences of a string of length n can be counted in $\mathcal{O}\left(n^{3/2}(\log n)^{1/2}\right)$ time and that both counting algorithms allow us to output x different 3-sub-cadences in $\mathcal{O}(xn)$ additional time if at least x different 3-sub-cadences exist.

Let $\sigma \in \Sigma$ be a character. We will now count all 3-sub-cadences with character σ.

Let (i, d) be a 3-sub-cadence. Since $i + d = \frac{i+(i+2d)}{2}$ holds, the position $i + d$ of the middle occurrence of σ only depends on the sum of the index i of first occurrence and the index $i + 2d$ of the third occurrence but does not depend on the individual indices of those two positions. Therefore, it is possible to determine the candidates for the middle occurrences with the convolution of the candidates of the first occurrence and the candidates of the third occurrence.

Let the sequence $\delta = (\delta_0, \delta_1, \delta_2, \ldots, \delta_n)$ be given by the indicator function for σ in S:

$$\delta_i := \begin{cases} 1 & \text{if } S[i] = \sigma \\ 0 & \text{if } S[i] \neq \sigma \text{ (this includes } i = 0) \end{cases}$$

With this definition, the product $\delta_i \delta_j$ is 1 if and only if $S[i] = S[j] = \sigma$ and otherwise is 0. Therefore $c_k = \sum_{i+j=k} (\delta_i \delta_j) = \#\{i : S[i] = S[k - i] = \sigma\}$ counts in how many ways the index $\frac{k}{2}$ lies in the middle of two σ. These partial sums can be calculated in $\mathcal{O}(n \log n)$ time by convolution.

If k is odd or $S\left[\frac{k}{2}\right] \neq \sigma$ holds, the index $\frac{k}{2}$ cannot be the middle index of a 3-sub-cadence. If $S\left[\frac{k}{2}\right] = \sigma$ holds, the indicator function $\delta_{\frac{k}{2}}$ is 1, and therefore $\delta_{\frac{k}{2}} \delta_{\frac{k}{2}} = 1$ holds as well. Since the arithmetic progression $(\delta_{\frac{k}{2}}, 0, 3)$ consisting of three times the number $\delta_{\frac{k}{2}}$ is not a 3-sub-cadence, the output element c_k contains one false positive. Additionally, for $i + j = k$ with $i \neq j$ and $S[i] = S[j] = \sigma$, the output element c_k counts the combination $\delta_i \delta_j$ as well as $\delta_j \delta_i$.

Therefore,

$$s_k := \begin{cases} \frac{c_{2k}-1}{2} & \text{if } S[k] = \sigma \\ 0 & \text{if } S[k] \neq \sigma \end{cases}$$

counts exactly the number of 3-sub-cadences with character σ such that the second occurrence of σ has index k. The sum of the s_k is the number of total 3-sub-cadences with character σ.

Also, for each $s_k \neq 0$, all those s_k 3-sub-cadences can be found in $\mathcal{O}(k) \subseteq \mathcal{O}(n)$ time by checking for each index $i < k$ whether $S[i] = S[k] = S[2k - i] = \sigma$ holds.

If the character σ is rare, we can also follow the idea of Amir et al. in [2] for detecting 3-cadences with rare characters: If all n_σ occurrences of the character are known, the c_k can be computed in $\mathcal{O}(n_\sigma^2)$ time by computing every pair of those occurrences. Therefore:

▶ **Theorem 8.** *For every character $\sigma \in \Sigma$, the 3-sub-cadences with σ can be counted in $\mathcal{O}(n \log n)$ time. Also, if all n_σ occurrences of σ are known, the 3-sub-cadences with σ can be counted in $\mathcal{O}(n_\sigma^2)$ time.*

Following the proof in [2], we can get all occurrences of every character by sorting the input string in $\mathcal{O}(n \log n)$ time. This implies that the algorithm needs at most $\mathcal{O}\left(\sum_{\sigma\in\Sigma} \min(n_\sigma^2, n \log n)\right) \subseteq \mathcal{O}\left(\frac{n}{(n\log n)^{1/2}} n \log n\right) = \mathcal{O}(n^{3/2}(\log n)^{1/2})$ time.

▶ **Theorem 9.** *The number of all 3-sub-cadences can be counted in*

$$\mathcal{O}\left(\min(|\Sigma| n \log n, n^{3/2}(\log n)^{1/2})\right) \text{ time.}$$

▶ **Theorem 10.** *After counting at least x 3-sub-cadences, it is possible to output x 3-sub-cadences in $\mathcal{O}(xn)$ time.*

4 Non-Rectangular Convolutions

In this section, we will extend the geometric interpretation of the convolution and show that for convex polygons P with k vertices and perimeter p it is possible to calculate the partial sums

$$c_k = \sum_{\substack{i+j=k \\ (i,j)\in P\cap\mathbb{Z}^2}} a_i b_j$$

in $\mathcal{O}\left(k + p(\log p)^2 \log k\right)$ time.

Let us imagine a graph where all integer coordinates (i, j) have the value $f(i, j) := a_i b_j$. We do not need the convolution in order to determine the sum of the function values in a given rectangle since we can use the simple factorization $\sum_{i=0}^{n}\sum_{j=0}^{m}(a_i b_j) = \left(\sum_{i=0}^{n} a_i\right)\left(\sum_{j=0}^{m} b_j\right)$ in $\mathcal{O}(n+m)$ time. However, the convolution provides the $2n$ partial sums on the 45°-diagonals in almost the same time of $\mathcal{O}((n + m)\log(n + m))$.

We will now extend this geometric interpretation firstly to triangles with a vertical cathetus and a horizontal cathetus, then to arbitrary triangles and lastly to convex polygons. In order to do this, we will cover the given polygon P in polygons P_p^+ and P_m^- such that for each integer point (i, j) the equality

$$\#\{P_p^+ | (i,j) \in P_p^+\} - \#\{P_m^- | (i,j) \in P_m^-\} = \begin{cases} 1 & \text{if } (i,j) \in P \\ 0 & \text{if } (i,j) \notin P \end{cases}$$

holds, and we define

$$(c_p)_k := \sum_{\substack{i+j=k \\ (i,j)\in P_p^+\cap\mathbb{Z}^2}} a_i b_j \text{ and } (c_m)_k := -\sum_{\substack{i+j=k \\ (i,j)\in P_m^-\cap\mathbb{Z}^2}} a_i b_j.$$

By construction, $c_k = (\sum(c_p)_k) + (\sum(c_m)_k)$ holds. However, if the edges and vertices of the polygons P_p^+ and P_m^- contain integer points, we need to carefully decide for every of these polygons, which edges and vertices are supposed to be included in the polygons and which are excluded from the polygons.

▶ **Lemma 11.** *Let P be a triangle with a vertical cathetus and a horizontal cathetus and perimeter p. Let also the sequences $a = (a_0, a_1, a_2, \ldots, a_n)$ and $b = (b_0, b_1, b_2, \ldots, b_n)$ be given.*

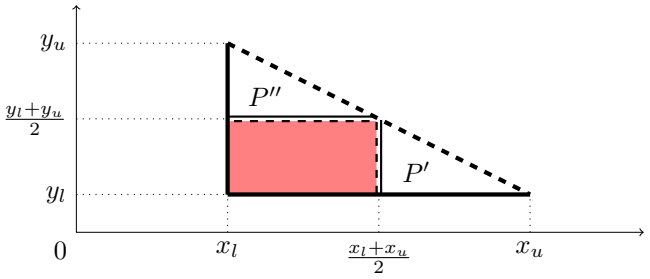


Figure 1 The right-angled triangle P in Lemma 11.

Then the partial sums

$$c_k = \sum_{\substack{i+j=k \\ (i,j)\in P\cap\mathbb{Z}^2}} a_i b_j$$

can be calculated in $\mathcal{O}\left(p(\log p)^2\right)$ *time.*

Proof. The proof will be symmetrical with regard to horizontal and vertical mirroring. Therefore, without loss of generality, we will assume that P is oriented as in Figure 1.

We first initialize the output vector $c = (c_{x_l+y_l}, c_{x_l+y_l+1}, c_{x_l+y_l+2}, \ldots, c_{x_u+y_u})$ with zero. This takes $\mathcal{O}(p)$ time.

In the following proof, we assume that both catheti are included in the polygon and that the hypotenuse as well as its endpoints are excluded. If this is not the expected behavior, we can traverse the edges in $\mathcal{O}(p)$ time and for each integer point (i, j) on the edge, we can decrease/increase the corresponding c_{i+j} by $a_i b_j$ if necessary.

If p is at most one, there is at most one integer point (i, j) in the triangle, and this point can be found in constant time. In this case, we only have to increase c_{i+j} by $a_i b_j$.

If p is bigger than one, we will separate the triangle P into three disjoint parts as seen in Figure 1.

- The triangle P' of points with x-coordinate of at least $\left\lceil \frac{x_l+x_u}{2} \right\rceil$,
- the triangle P'' of points with y-coordinate of at least $\left\lceil \frac{y_l+y_u}{2} \right\rceil$ and
- the red rectangle of points with x-coordinate of at most $\left\lceil \frac{x_l+x_u}{2} \right\rceil - 1$ and y-coordinate of at most $\left\lceil \frac{y_l+y_u}{2} \right\rceil - 1$.

There are no integers bigger than $\left\lceil \frac{x_l+x_u}{2} \right\rceil - 1$ but smaller than $\left\lceil \frac{x_l+x_u}{2} \right\rceil$ nor integers bigger than $\left\lceil \frac{y_l+y_u}{2} \right\rceil - 1$ but smaller than $\left\lceil \frac{y_l+y_u}{2} \right\rceil - 1$. Therefore, each integer point in P is in exactly one of the three parts.

For the red rectangle, we can calculate the convolution and thereby get the corresponding partial sums in $\mathcal{O}(p \log p)$ time. The partial sums corresponding to the sub-triangles are calculated recursively. Increasing the c_k by the partial results leads to the final result.

Hence, the algorithm takes

$$\mathcal{O}\left(p + \left(\sum_{i=0}^{\log_2 p} 2^i \left(\frac{p}{2^i} \log \frac{p}{2^i}\right)\right) + 2^{\log_2 p}\right) \subseteq \mathcal{O}\left(\sum_{i=0}^{\log p} p \log p\right) = \mathcal{O}\left(p(\log p)^2\right)$$

time. ◀

We will now further extend this result to arbitrary triangles:

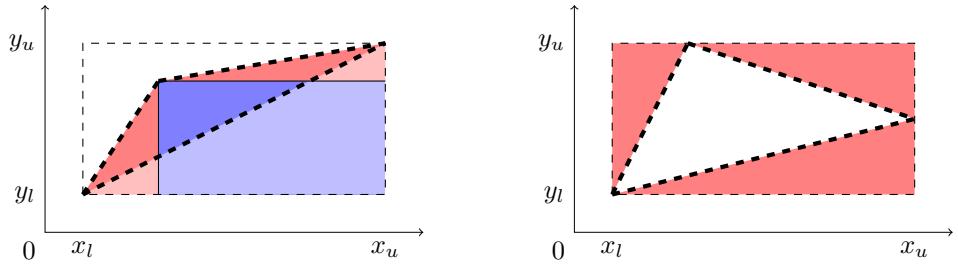


Figure 2 The two possible triangles P in Lemma 12.

▶ **Lemma 12.** *Let a triangle P with perimeter p and sequences $a = (a_0, a_1, a_2, \ldots, a_n)$ and $b = (b_0, b_1, b_2, \ldots, b_n)$ be given.*

Then the partial sums

$$c_k = \sum_{\substack{i+j=k \\ (i,j)\in P\cap\mathbb{Z}^2}} a_i b_j$$

can be calculated in $\mathcal{O}\left(p(\log p)^2\right)$ time.

Proof. Let x_l, y_l, x_u, y_u be the minimal and maximal x-coordinates and y-coordinates of the three vertices of the polygon P. As in the last lemma, we first initialize the output vector $c = (c_{x_l+y_l}, c_{x_l+y_l+1}, c_{x_l+y_l+2}, \ldots, c_{x_u+y_u})$.

Similarly to the last lemma, we can remove/add edges and vertices in linear time with respect to p. Since the number of edges and vertices is constant, we ignore them for the sake of simplicity.

Let R be the rectangle $\{(x, y)|x_l < x < x_u \wedge y_l < y < y_u\}$. Since R has four edges but P only has three vertices, at least one of the vertices of P is also a vertex of R. Without loss of generality, this vertex is (x_l, y_l).

Case 1: The opposing vertex (x_u, y_u) in R also coincides with a vertex of P (as in the left-hand side of Figure 2):

Without loss of generality, we can assume that the third vertex of P is above the diagonal from (x_l, y_l) to (x_u, y_u). In this case, the partial sums corresponding to P are given by the sum of the partial sums of the red triangles and the partial sums of the blue rectangle minus the partial sums of the lighter triangle.

There are only three triangles and one rectangle involved, and each of those polygons has perimeter $\mathcal{O}(p)$. Furthermore, all triangles have a vertical cathetus and a horizontal cathetus. Therefore, using Lemma 11, we can calculate all partial sums in $\mathcal{O}\left(p(\log p)^2\right)$ time.

Case 2: The opposing vertex (x_u, y_u) in R does not coincide with a vertex of P (as in the right-hand side of Figure 2):

In this case, one vertex of P lies on the right edge of R and one vertex of P lies on the upper edge of R.

The wanted partial sums are in this case the difference of the partial sums of the rectangle and of the partial sums of the three red triangles. Again, we can calculate all partial sums in $\mathcal{O}\left(p(\log p)^2\right)$ time.

Since both cases require $\mathcal{O}\left(p(\log p)^2\right)$ time, this concludes the proof. ◀

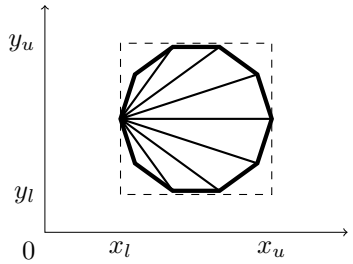


Figure 3 A regular k-gon. All chords from the leftmost vertex to the vertices on the right-hand side of the k-gon are at least $\frac{p}{4}$ long. The sum of all chords' lengths is therefore $\Theta(kp)$.

Now we will extend this algorithm to convex polygons with k vertices by dissecting them into $k-2$ triangles by adding $k-3$ chords. Since the time complexity of the triangular convolution given by Lemma 12 depends on the sum of the triangles' perimeters, it is not sufficient to just select one vertex and connect it with every other vertex in the polygon (see Figure 3). On the other hand, the triangulation algorithm itself should not take longer than the convolutions. Additionally, the order in which the chords are added does not matter for the convolutions. We will show that for convex polygons there is a triangulation which can be computed in linear time and only increases the perimeter by the factor $\mathcal{O}(\log k)$.

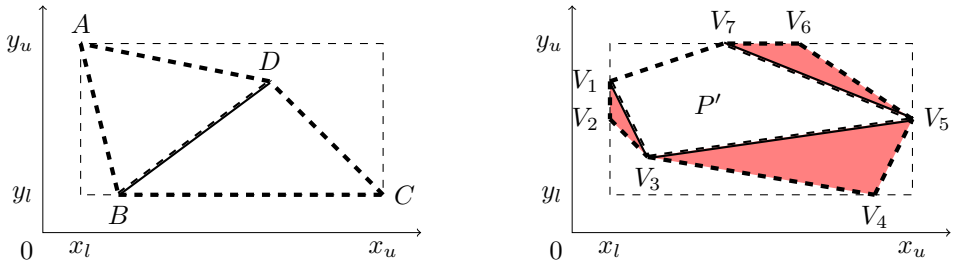


Figure 4 Two possible convex polygons P with more than 3 vertices in Lemma 13.

▶ **Theorem 13.** *Let P be a convex polygon with k vertices and perimeter p. Let also the sequences $a = (a_0, a_1, a_2, \ldots, a_n)$ and $b = (b_0, b_1, b_2, \ldots, b_n)$ be given.*

Then the partial sums

$$c_k = \sum_{\substack{i+j=k \\ (i,j)\in P\cap\mathbb{Z}^2}} a_i b_j$$

can be calculated in $\mathcal{O}\left(k + p(\log p)^2 \log k\right)$ time.

Proof. As in the last two Lemmata, we define x_l, y_l, x_u, y_u to be the minimal and maximal x-coordinates and y-coordinates of the k vertices of P. Also, we first initialize the output vector $c = (c_{x_l+y_l}, c_{x_l+y_l+1}, c_{x_l+y_l+2}, \ldots, c_{x_u+y_u})$. We further assume that none of the edges and vertices of P is included in P.

If P is a triangle, then this Lemma simplifies to Lemma 12 and there is nothing left to prove.

If P is a quadrilateral $ABCD$, as in the left-hand side of Figure 4, then it can be partitioned into the triangles ABD and CDB where the edge BD is included in exactly one triangle and all other edges are excluded. The triangle inequality proves that $|BD| \leq |DA| + |AB|$ and $|BD| \leq |BC| + |CD|$ hold. Therefore, both triangles have a perimeter of at most p. This implies that the partial sums can be calculated in $\mathcal{O}\left(p(\log p)^2\right)$.

If P is a polygon $V_1V_2V_3\dots V_k$ with more than four vertices, as in the right-hand side of Figure 4, it can be partitioned into

- the polygon $P' = V_1V_3V_5\dots V_{2\lceil\frac{k}{2}\rceil-1}$, which is given by the odd vertices without its edges,
- the red triangles $V_iV_{i+1}V_{i+2}$ with $i = 1, 3, 5, \dots, 2\left\lceil\frac{k}{2}\right\rceil - 3$ including the edge V_iV_{i+2} but excluding the other edges and the vertices,
- if k is even, the triangle $V_{k-1}V_k$ including the edge $V_{k-1}V_{k+1}$ but excluding the other edges and the vertices.

By construction and triangle inequality, the perimeter p' of P' is at most p. This, however, also implies that the total perimeter $\sum p_i$ of the triangles is at most $2p$. The inequality

$$\sum \min\left(1, p_i(\log p_i)^2\right) \leq k + \sum \left(p_i(\log p)^2\right) \leq k + p(\log p)^2$$

implies that the algorithm needs $\mathcal{O}\left(k + p(\log p)^2\right)$ time plus the time we need for processing P'. Since each step almost halves the number of vertices, we need $\mathcal{O}(\log k)$ steps. This results in a total time complexity of $\mathcal{O}\left(k + p(\log p)^2 \log k\right)$. ◀

5 (a,b,c)-Partial-k-Cadences

In this section, we will show how the non-rectangular convolution helps counting the (a, b, c)-partial-k-cadences as defined in Definition 6.

In particular, we will show that (a, b, c)-partial-k-cadences with a given character σ can be counted in $\mathcal{O}\left(n(\log n)^2\right)$ time. We will further show that all (a, b, c)-partial-k-cadences can be counted in $\mathcal{O}\left(\min(|\Sigma|n(\log n)^2, n^{3/2}\log n)\right)$ time and that both counting algorithms allow us to output x of those partial-cadences in $\mathcal{O}(xn)$ time.

As a special case, these results also hold for 3-cadences.

We further conclude from these results that the existence of k-cadences with at most $k-3$ errors as defined in Definition 5 can be detected in $\mathcal{O}\left(\min(|\Sigma|k^3n(\log n)^2, k^3n^{3/2}\log n)\right)$ time.

Without loss of generality, we will only deal with the case $a < b$ in this section.

▶ **Lemma 14.** *Three positions x, y and z form a (a, b, c)-partial-k-cadence if and only if*

- *the equation $\frac{y-x}{b-a} = \frac{z-y}{c-b} \in \mathbb{Z}$ holds,*
- *the equation $S[x] = S[y] = S[z]$ holds and*
- *the inequalities*

$$0 \geq \frac{(b+1)x - (a+1)y}{b-a}, \tag{1}$$

$$0 < \frac{bx - ay}{b-a}, \tag{2}$$

$$n \geq \frac{(b-k+1)x - (a-k+1)y}{b-a} \text{ and} \tag{3}$$

$$n < i + kd = \frac{(b-k)x - (a-k)y}{b-a} \text{ hold.} \tag{4}$$

Proof. Define $d := \frac{y-x}{b-a}$ and $i := x - ad$. Then $x = i + ad$ and $y = i + bd$. Furthermore, the equation $\frac{y-x}{b-a} = \frac{z-y}{c-b}$ holds if and only if $z = i + cd$ and $\frac{y-x}{b-a} \in \mathbb{Z}$ holds if and only if d is an integer.

Additionally, using $x = i + ad$ and $y = i + bd$, the four inequalities can be simplified to $0 \geq i - d$, $0 < i$, $n \geq i + (k-1)d$ and $n < i + kd$.

Therefore, the lemma follows from the definition of the partial-cadence. ◀

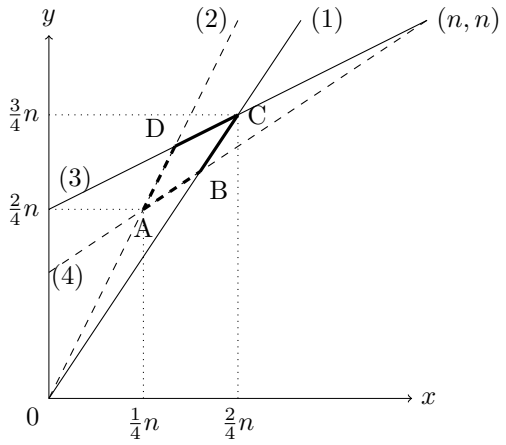


Figure 5 The four inequalities of Lemma 14 for $(1, 2, 3)$-partial-4-cadences.

The four inequalities hold if the points (x, y) lie inside the convex quadrilateral given, as shown in Figure 5, by the corners

$$A = \left(\frac{an}{k}, \frac{bn}{k}\right)$$
$$B = \left(\frac{(a+1)n}{k+1}, \frac{(b+1)n}{k+1}\right)$$
$$C = \left(\frac{(a+1)n}{k}, \frac{(b+1)n}{k}\right)$$
$$D = \left(\frac{an}{k-1}, \frac{bn}{k-1}\right)$$

including the vertex C and the edges between B and C as well as between C and D but excluding all other vertices and the edges between A and B as well as between D and A.

For given $x = i + ad$ and $y = i + bd$, the third occurrence $z = i + cd$ can be calculated with the equation $i + cd = \frac{(b-c)(i+ad)+(c-a)(i+bd)}{b-a}$ directly without calculating i and d first. The corresponding partial sums

$$c_k = \sum_{\substack{i+j=k \\ (i,j)\in P\cap\mathbb{Z}^2}} a_{\frac{i}{(b-c)}} b_{\frac{j}{(c-a)}}$$

can be calculated by using the partial sums

$$c_k = \sum_{\substack{i+j=k \\ (i,j)\in P'\cap\mathbb{Z}^2}} a'_i b'_j$$

with $a'_i := \begin{cases} a_{\frac{i}{b-c}} & \text{if } i \equiv 0 \pmod{b-c} \\ 0 & \text{otherwise} \end{cases}$ and $b'_j := \begin{cases} b_{\frac{j}{c-a}} & \text{if } j \equiv 0 \pmod{c-a} \\ 0 & \text{otherwise} \end{cases}$ and a polygon P', which is derived from P by stretching the first coordinate by $(b-c)$ and the second coordinate by $(c-a)$. The perimeter of P' is at most $\max(|b-c|, |c-a|)$ times the perimeter of P. Using the quadrilateral $P = ABCD$ with perimeter

$$p \le 2|C_x - A_x| + 2|C_y - A_y| = 2\left(\frac{(a+1)n}{k} - \frac{an}{k}\right) + 2\left(\frac{(b+1)n}{k} - \frac{bn}{k}\right) = \frac{4n}{k} \in \mathcal{O}\left(\frac{n}{k}\right),$$

the polygon P' has perimeter $p' \in \mathcal{O}(n)$. This proves the following three theorems.

▶ **Theorem 15.** *For every character $\sigma \in \Sigma$, the (a, b, c)-partial-k-cadences with σ can be counted in $\mathcal{O}(n(\log n)^2)$ time. Also, if all n_σ occurrences of σ are known, the (a, b, c)-partial-k-cadences with σ can be counted in $\mathcal{O}(n_\sigma^2)$ time.*

▶ **Theorem 16.** *The number of all (a, b, c)-partial-k-cadences can be counted in*

$$\mathcal{O}\left(\min(|\Sigma|n(\log n)^2, n^{3/2}\log n)\right) \text{ time.}$$

▶ **Theorem 17.** *After counting at least x (a, b, c)-partial-k-cadences, it is possible to output x (a, b, c)-partial-k-cadences in $\mathcal{O}(xn)$ time.*

Since every 3-cadence is an $(0, 1, 2)$-partial-3-cadence, we also obtain the special case:

▶ **Corollary 18.** *For every character $\sigma \in \Sigma$, the 3-cadences with σ can be counted in $\mathcal{O}(n(\log n)^2)$ time. Also, if all n_σ occurrences of σ are known, the 3-cadences with σ can be counted in $\mathcal{O}(n_\sigma^2)$ time.*

Therefore, the number of all 3-cadences can be counted in

$$\mathcal{O}\left(\min(|\Sigma|n(\log n)^2, n^{3/2}\log n)\right) \text{ time.}$$

Also, after counting at least x 3-cadences, it is possible to output x 3-cadences in $\mathcal{O}(xn)$ time.

Taking the sum over all possible triples (a, b, c), we can also search for k-cadences with at most $k - 3$ errors. It can be checked in

$$\mathcal{O}\left(\min(|\Sigma|k^3n(\log n)^2, k^3n^{3/2}\log n)\right)$$

time whether the given string has a k-cadence with at most $k - 3$ errors. However, since k-cadences with less than $k - 3$ errors are counted more than once, it seems to be difficult to determine the exact number of k-cadences with at most $k - 3$ errors.

6 Conclusion

This paper extends convolutions to arbitrary convex polygons. One might wonder whether these convolutions could be sped up or be further extended to non-convex polynomials.

Instead of just partitioning the interior of the polygon into triangles, it is also possible to identify polygons by the difference of a slightly bigger but less complex polygon and a triangle. However, if the algorithm presented in this paper is adapted to non-convex polygons, it can generate self-intersecting polygons. While the time-complexity stays the same for these polygons, it becomes hard to ensure that every vertex and every edge of the polygon is counted exactly once.

Another approach is given by Levcopoulos and Lingas in [13]. This paper shows that any simple polygon can be decomposed into convex components in $\mathcal{O}(k \log k)$ time while only increasing the total perimeter by the factor $\mathcal{O}(\log k)$. This paper also shows that if the input polygon is rectilinear, this partition only contains axis-aligned rectangles. Since the convolution handles rectangles quicker and more easily than triangles, this saves a logarithm. However, in general, it is not obvious how to transform arbitrary polygons into equivalent simple rectilinear polygons in quasilinear time without blowing-up the number of vertices too much.

The non-rectangular convolution, unlike the usual convolution, allows us to define a dependence between the indices of the convoluted sequences. This dependence is not usable in applications like the multiplication of polynomials, and for many signal processing applications

this extended method does not seem to bring any benefits either. However, in order to count the partial-cadences this dependence was essential. The non-rectangular convolution may also have future applications in image processing and convolutional neural networks.

In terms of cadences, this paper presents algorithms to count and find sub-cadences, cadences and partial-cadences with three elements. However, if there are linearly many c-positions of (a, b, c)-partial-k-cadences, the knowledge of those partial-cadences does not lead to a sub-quadratic-time-algorithm for determining the existence 4-cadences. On the other hand, it is also not shown that this problem needs quadratic time.

Also, the time-complexity $\mathcal{O}(xn)$ for finding x 3-cadences is quite pessimistic. If there are many 3-cadences, it is very likely that quite a few of these 3-cadences share one of their occurrences. These occurrences can be found in $\mathcal{O}(n)$ time. On the other hand, in the string $10^{n-1}1^{2n}$, for example, there are linearly many 3-cadences but every second occurrence and every third occurrence only occurs in at most one of those 3-cadences.

References

1. R. C. Agarwal and C. S. Burrus. Number theoretic transforms to implement fast digital convolution. *Proceedings of the IEEE*, 63(4):550–560, April 1975. doi:10.1109/PROC.1975.9791.
2. Amihood Amir, Alberto Apostolico, Travis Gagie, and Gad M. Landau. String cadences. *Theoretical Computer Science*, 698:4–8, 2017. Algorithms, Strings and Theoretical Approaches in the Big Data Era (In Honor of the 60th Birthday of Professor Raffaele Giancarlo). doi:10.1016/j.tcs.2017.04.019.
3. N. C. Ankeny. The Least Quadratic Non Residue. *Annals of Mathematics*, 55(1):65–72, 1952. URL: http://www.jstor.org/stable/1969420.
4. Eric Bach and Jonathan Sorenson. Explicit bounds for primes in residue classes. *Math. Comput.*, 65(216):1717–1735, October 1996. doi:10.1090/S0025-5718-96-00763-6.
5. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.
6. Svyatoslav Covanov and Emmanuel Thomé. Fast integer multiplication using generalized fermat primes. *Math. Comp.*, 88(317):1449–1477, 2019. doi:10.1090/mcom/3367.
7. Anindya. De, Piyush P. Kurur, Chandan. Saha, and Ramprasad. Saptharishi. Fast integer multiplication using modular arithmetic. *SIAM Journal on Computing*, 42(2):685–699, 2013. doi:10.1137/100811167.
8. J. Gardelle. Cadences. *Mathématiques et Sciences humaines*, 9:31–38, 1964. URL: http://www.numdam.org/item/MSH_1964__9__31_0.
9. D. Harvey and J. van der Hoeven. Faster integer multiplication using plain vanilla FFT primes. *Math. Comp.*, 88(315):501–514, 2019.
10. D. Harvey and J. van der Hoeven. Integer multiplication in time O(n log n). working paper or preprint, March 2019. URL: https://hal.archives-ouvertes.fr/hal-02070778.
11. D. Harvey, J. van der Hoeven, and G. Lecerf. Even faster integer multiplication. *Journal of Complexity*, 36:1–30, 2016. doi:10.1016/j.jco.2016.03.001.
12. D. R. Heath-Brown. Almost-primes in arithmetic progressions and short intervals. *Mathematical Proceedings of the Cambridge Philosophical Society*, 83(3):357–375, 1978. doi:10.1017/S0305004100054657.
13. Christos Levcopoulos and Andrzej Lingas. Bounds on the length of convex partitions of polygons. In Mathai Joseph and Rudrapatna Shyamasundar, editors, *Foundations of Software Technology and Theoretical Computer Science*, pages 279–295, Berlin, Heidelberg, 1984. Springer Berlin Heidelberg.
14. U. V. Linnik. On the least prime in an arithmetic progression. I. The basic theorem. *Rec. Math. [Mat. Sbornik] N.S.*, 15(57):139–178, 1944.

15 M. Lothaire. *Combinatorics on Words*. Cambridge Mathematical Library. Cambridge University Press, 1997. URL: `https://books.google.de/books?id=eATLTZzwW-sC`.

16 Franz Mertens. Ein Beitrag zur analytischen Zahlentheorie. *Journal für die reine und angewandte Mathematik*, 78:46–62, 1874. URL: `http://eudml.org/doc/148244`.

17 A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7(3):281–292, September 1971. `doi:10.1007/BF02242355`.

18 William B. Thompson, Peter Shirley, and James A. Ferwerda. A Spatial Post-Processing Algorithm for Images of Night Scenes. *Journal of Graphics Tools*, 7(1):1–12, 2002. `doi:10.1080/10867651.2002.10487550`.

19 Bartel Leendert van der Waerden. Beweis einer Baudet'schen Vermutung. *Nieuw Archief voor Wiskunde*, 15:212–216, 1927.

20 Samuel S. Wagstaff. Greatest of the least primes in arithmetic progressions having a given modulus. *Mathematics of Computation*, 33(147):1073–1080, 1979. URL: `http://www.jstor.org/stable/2006082`.

21 Triantafyllos Xylouris. *Über die Nullstellen der Dirichletschen L-Funktionen und die kleinste Primzahl in einer arithmetischen Progression*, volume 404 of *Bonner Mathematische Schriften [Bonn Mathematical Publications]*. Universität Bonn, Mathematisches Institut, Bonn, 2011. Dissertation for the degree of Doctor of Mathematics and Natural Sciences at the University of Bonn, Bonn, 2011.

A Convolutions

It is well-known that the discrete acyclic convolution can be calculated with $\mathcal{O}(n \log n)$ complex arithmetic operations. However, if the convolution is calculated with the fast Fourier transform, the finite register lengths introduce roundoff errors. These errors can propagate and accumulate throughout the calculation.

Therefore, in order to calculate the convolution of integer sequences, it seems more convenient to use the number theoretic transform, which is the generalization of the fast Fourier transform from the field of the complex numbers to certain residue class rings.

In this section, we will show that after some precomputation in $\mathcal{O}\left(n(\log n)^2(\log\log n)\right)$ time it is possible to calculate these convolutions in $\mathcal{O}(n \log n)$ time.

Agarwal and Burrus show in [1] that the circular convolution of two integer vectors of length n can be efficiently computed modulo a prime p if $p-1$ is a multiple of n. Therefore we want to find a prime p in the infinite arithmetic progression $\{n+1, 2n+1, 3n+1, \dots\}$.

The prime number theorem states that the number $\pi(N)$ of primes smaller than N asymptotically behaves like $\frac{N}{\log N}$. Furthermore, Dirichlet's prime number theorem states that for a given n and a sufficiently large N, the prime numbers are evenly distributed in all residue classes $mn + r$ with $\gcd(n, r) = 1$.

Therefore, for a given n and sufficiently large N, we should expect circa $\frac{N}{\varphi(n) \log N}$ prime numbers of the form $mn + 1$ that are smaller than N. However, the "sufficient largeness" of N depends on n. Therefore, these theorems do not provide the number of suitable primes smaller than the given number N.

Since the primes are expected to behave similarly in all coprime residue classes, Heath-Brown suggests in [12] that the least prime of the form $mn + 1$ is in $\mathcal{O}\left(n(\log n)^2\right)$. Wagstaff gives in [20] a heuristic argument to this conjecture and provides numerical evidence. However, the best proven upper bounds are much larger, even if the generalized Riemann hypothesis is assumed.

Linnik proves in [14] that there are constants c and L such that for each n, r with $\gcd(n, r) = 1$, there is a prime of the form $mn + r$ with $mn + r < cn^L$. While Linnik himself did not provide the values of c and L, there are some upper bounds: For example,

Xylouris proves in [21] (written in German) that there is a c such that for each n, r with $\gcd(n,r)=1$, there is a prime of the form $mn+r$ with $mn+r<cn^5$. More explicitly, Bach and Sorenson present in [4] that if the generalized Riemann hypothesis holds, for each n, r with $\gcd(n,r)=1$, there is a prime of the form $mn+r$ with $mn+r<2(n\log n)^2$.

Without a shortcut allowing us to check the existence of a prime in a given finite arithmetic progression quickly, we have to test for each single number in this progression whether it is prime.

Therefore, using only the generalized Riemann hypothesis, we cannot expect to find a prime deterministically in $o\left(\frac{(n\log n)^2}{n}\right) = o\left(n(\log n)^2\right)$, even if we use on average only a constant time for each possible prime number.

Since this is already too slow for fast multiplications, numerous ways to solve or circumvent this problem have been established:

- Harvey and van der Hoeven propose in [10] a multiplication algorithm which uses $\mathcal{O}(n\log n)$ time by using the fast Fourier transform based on complex numbers.
- Some algorithms use stronger assumptions for the distribution of prime numbers. For example, Harvey and van der Hoeven use in [11] an unproven lower bound for the number of Mersenne primes and in [9] they assume that the least prime of the form $mn+c$ is in all coprime residue classes in $\mathcal{O}\left(\varphi(n)(\log n)^2\right)$. Also, Covanov and Thomé use in [6] an unproven lower bound for the number of generalized Fermat primes.
- Many algorithms do not use convolution of length n but divide the number into blocks first and then use shorter convolutions over large rings. For example De et al. use the ring $\mathbb{Z}[\alpha]/\left(p^c,\alpha^m+1\right)$ in [7].
- While De et al. do not use it in their multiplication algorithm, they provide in [7] a randomized algorithm to find a suitable prime in expected running time $\widetilde{\mathcal{O}}((\log n)^3)$.

In the next theorem, we will show that the sieve of Eratosthenes comes close to the theoretical minimum of $\mathcal{O}\left(n(\log n)^2\right)$ for finding all primes of the form $mn+1$ up to $(n\log n)^2$.

The lengths of these primes is at most 4 times the length of n. Therefore, such a prime number p_n is a good modulus for the convolution of length n or any of its divisors.

▶ **Theorem 19.** *Let n be an integer. A prime $p_n \equiv 1 \pmod{n}$ with $p_n < 2(n\log n)^2$ can be found in $\mathcal{O}(n(\log n)^2\log\log(n))$ time.*

Proof. The main idea is to use the sieve of Eratosthenes to first find all primes up to $2n\log n$ and then sieve only the numbers up to $2(n\log n)^2$ that are congruent to 1 modulo n with these primes.

On the one hand, since $(2n\log n)^2 > 2(n\log n)^2$ holds, all numbers left after the second sieving are primes. On the other hand, the result of Bach and Sorenson in [4] guarantees that if the generalized Riemann hypothesis holds, there is a prime left. Also, by construction, all primes p_n left fulfill this theorem.

It remains to be shown that this algorithm can be done in $\mathcal{O}(n(\log n)^2\log\log(n))$ time.

For the usual sieve of Eratosthenes, one prepares a Boolean array for the first $2n\log n$ numbers. Then, for each number that has not been marked as non-prime, every multiple is marked as non-prime. Afterwards, all non-marked numbers are returned. The majority of the time is spent for the marking. This takes

$$\mathcal{O}\left(\sum_{\substack{p=2\\ p \text{ is prime}}}^{2n\log n} \frac{2n\log n}{p}\right) = \mathcal{O}\left(n\log n \sum_{\substack{p=2\\ p \text{ is prime}}}^{2n\log n} \frac{1}{p}\right) = \mathcal{O}\left(n(\log n)(\log\log n)\right)$$

time. The last equality is given by Mertens in [16, p. 46] (written in German) and the inequality $\log\log(2n\log n) < 2\log\log(n)$.

For the second part, we have a much larger interval of numbers. However, since we only have to consider the first residue class, only every n-th number has to be considered. Therefore we need

$$\mathcal{O}\left(\sum_{\substack{p=2\\ p \text{ is prime}}}^{2n\log n} \frac{2(n\log n)^2}{np}\right) = \mathcal{O}\left(n(\log n)^2 \sum_{\substack{p=2\\ p \text{ is prime}}}^{2n\log n} \frac{1}{p}\right) = \mathcal{O}\left(n(\log n)^2(\log\log n)\right)$$

markings. Using the extended Euclidean algorithm, for every prime p, we can find the smallest f such that $fp \equiv 1 \pmod{n}$ in $\mathcal{O}(\log p) \subseteq \mathcal{O}(\log n)$ time. Summing up over all primes, this takes

$$\mathcal{O}\left(\sum_{\substack{p=2\\ p \text{ is prime}}}^{2n\log n} \log n\right) \subseteq \mathcal{O}\left(n(\log n)^2\right)$$

time.

This concludes the proof. ◀

It is not only possible to find a suitable modulus for the number theoretic transform, but we can also find a suitable 2^t-th root in the corresponding residue ring:

▶ **Theorem 20.** *Let p_{2^t} be a prime with $p_{2^t} \equiv 1 \pmod{2^t}$ and $p_{2^t} < 2(2^t\log(2^t))^2$. A 2^t-th root of unity modulo p_{2^t} can be found in $\mathcal{O}\left((\log p_{2^t})^3\right)$ time.*

Proof. Let $p_{2^t} = 1 + o2^r$ for an odd number o.

Firstly, we will show that a residue q^o is a 2^r-th root of unity modulo p_{2^t} if and only if q is a quadratic nonresidue modulo p_{2^t}.

Since p_{2^t} is prime, there is a primitive root a modulo p_{2^t}.

Let $q \equiv a^i$. Then $q^o = a^{io}$ has the order $\frac{o2^r}{\gcd(io, o2^r)} = \frac{2^r}{\gcd(i, 2^r)}$. Therefore, q^o has order 2^r if and only if i is odd. On the other hand, if i is even, then q is a quadratic residue, and if i is odd, then $q \equiv a^i = a\left(a^{\frac{i-1}{2}}\right)^2$ is a quadratic nonresidue. This implies that q^o is a 2^r-th root of unity modulo p_{2^t} if and only if q is a quadratic nonresidue modulo p_{2^t}.

Ankeny shows in [3] that if the generalized Riemann hypothesis holds, there is a quadratic nonresidue in the first $\mathcal{O}\left((\log p_{2^t})^2\right)$ residue classes. For any residue q it can be tested with $\mathcal{O}(\log p_{2^t})$ multiplications and modulo operations whether q^o has order 2^r. As byproduct we get $(q^o)^{\left(2^{r-t}\right)}$. If and only if q^o has order 2^r, the power $(q^o)^{\left(2^{r-t}\right)}$ has order 2^t.

Therefore, a 2^t-th root of unity modulo p_{2^t} can be found in $\mathcal{O}\left((\log p_{2^t})^3\right)$ time. ◀

Therefore, we can efficiently compute the integer convolution with the help of the number theoretic transform.

▶ **Theorem 21.** *For a given integer N, we can find a modulus p_N and a suitable root q_N in $\mathcal{O}\left(N(\log N)^2(\log\log N)\right)$ time such that it is possible to calculate the acyclic convolution modulo p_N of two sequences of length $n \leq N$ in $\mathcal{O}(n\log n)$ time afterwards.*

Proof. The acyclic convolution of sequences of length n can be derived from a circular convolution of sequences with lengths of at least $2n$. Therefore, we will first prepare circular convolutions of length 2^T with $2N \leq 2^T < 4N$.

For this length, the last two theorems state that a suitable modulus p_N and a suitable 2^T-th root q_N of unity can be found in $\mathcal{O}\left(N(\log N)^2(\log\log N)\right)$.

Afterwards, for every $n \leq N$ we can append zeros to get the length 2^t with $2n \leq 2^t < 4n$. Since 2^t is a divisor of 2^T, we can use $(q_N)^{\left(2^{T-t}\right)}$ as 2^t-th root of unity.

This allows the calculation of the acyclic convolution modulo p_N in $\mathcal{O}\left(n \log n\right)$ time. ◀

Maximum Matchings in Geometric Intersection Graphs

Édouard Bonnet
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France
edouard.bonnet@ens-lyon.fr

Sergio Cabello
Faculty of Mathematics and Physics, University of Ljubljana, and IMFM, Slovenia
sergio.cabello@fmf.uni-lj.si

Wolfgang Mulzer
Institut für Informatik, Freie Universität Berlin, Germany
mulzer@inf.fu-berlin.de

Abstract

Let G be an intersection graph of n geometric objects in the plane. We show that a maximum matching in G can be found in $O(\rho^{3\omega/2}n^{\omega/2})$ time with high probability, where ρ is the density of the geometric objects and $\omega > 2$ is a constant such that $n \times n$ matrices can be multiplied in $O(n^\omega)$ time.

The same result holds for any subgraph of G, as long as a geometric representation is at hand. For this, we combine algebraic methods, namely computing the rank of a matrix via Gaussian elimination, with the fact that geometric intersection graphs have small separators.

We also show that in many interesting cases, the maximum matching problem in a general geometric intersection graph can be reduced to the case of bounded density. In particular, a maximum matching in the intersection graph of any family of translates of a convex object in the plane can be found in $O(n^{\omega/2})$ time with high probability, and a maximum matching in the intersection graph of a family of planar disks with radii in $[1, \Psi]$ can be found in $O(\Psi^6 \log^{11} n + \Psi^{12\omega} n^{\omega/2})$ time with high probability.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Graph algorithms analysis

Keywords and phrases computational geometry, geometric intersection graph, maximum matching, disk graph, unit-disk graph

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.31

Related Version A full version of the paper is available at `http://arxiv.org/abs/1910.02123`.

Funding *Sergio Cabello*: Supported by the Slovenian Research Agency (P1-0297, J1-9109, J1-8130, J1-8155).
Wolfgang Mulzer: Supported in part by ERC StG 757609.

1 Introduction

Let $\mathcal{U}$ be a family of (connected and compact) objects in $\mathbb{R}^2$. The ***intersection graph*** $G_\mathcal{U}$ of $\mathcal{U}$ is the undirected graph with vertex set $\mathcal{U}$ and edge set

$$E(G_\mathcal{U}) \;=\; \{UV \mid U, V \in \mathcal{U},\, U \cap V \neq \emptyset\}.$$

If the objects in $\mathcal{U}$ are partitioned into two sets, one can also define the *bipartite* intersection graph, a subgraph of $G_\mathcal{U}$, in the obvious way. Consider the particular case when $\mathcal{U}$ is a set of disks. Then, we call $G_\mathcal{U}$ a ***disk graph***, and if all disks in $\mathcal{U}$ have the same radius, a ***unit-disk graph***. Unit disk graphs are often used to model ad-hoc wireless communication

networks and sensor networks [11, 14, 29]. Disks of varying sizes and other shapes become relevant when different sensors cover different areas. Moreover, general disk graphs serve as a tool to approach other problems, like the barrier resilience problem [16].

We consider a classic optimization problem, *maximum matching*, in the setting of geometric intersection graphs, and introduce two new techniques, each interesting in its own. First, we provide an efficient algorithm to compute a maximum matching in any subgraph of the intersection graph of geometric objects of low density. Second, we provide a sparsification technique to reduce the maximum matching problem in a geometric intersection graph to the case of low density. The sparsification works for convex shapes of similar sizes for which certain range searching operations can be done efficiently.

In this paper, we use ω to denote a constant such that $\omega > 2$ and any two $n \times n$ matrices can be multiplied in time $O(n^\omega)$.[1]

Maximum matching in intersection graphs of geometric objects of low density

We first introduce some geometric concepts. The diameter of a set $X \subset \mathbb{R}^2$, denoted by $\operatorname{diam}(X)$, is the supremum of the distances between any two points of X. The ***density*** $\rho(\mathcal{U})$ of a family $\mathcal{U}$ of objects is

$$\rho(\mathcal{U}) \;=\; \max_{X \subseteq \mathbb{R}^2} \big|\{U \in \mathcal{U} \mid \operatorname{diam}(U) \geq \operatorname{diam}(X),\, U \cap X \neq \emptyset\}\big|. \tag{1}$$

One can also define the density by considering for X only disks. Since an object of diameter d can be covered by $O(1)$ disks of diameter d, this changes the resulting parameter by only a constant. (See, for example, the book by de Berg et al. [6, Section 12.5] for such a definition.) The ***depth*** (ply) of $\mathcal{U}$ is the largest number of objects that cover a single point:

$$\max_{p \in \mathbb{R}^2} \big|\{U \in \mathcal{U} \mid p \in U\}\big|.$$

For disk graphs and square graphs, the depth and the density are linearly related; see for example Har-Peled and Quanrud [13, Lemma 2.7]. More generally, bounded depth and bounded density are equivalent whenever we consider homothets of a constant number of shapes. Density and depth are usually considered in the context of realistic input models; see de Berg et al. [7] for a general discussion.

Let $\mathbb{G}_\rho$ be the family of *subgraphs* of intersection graphs of geometric objects in the plane with density at most ρ. Our goal is to compute a maximum matching in graphs of $\mathbb{G}_\rho$, assuming the availability of a geometric representation of the graph and a few basic geometric primitives on the geometric objects. For this, we consider the density ρ as an additional parameter. Naturally, the case $\rho = O(1)$ of *bounded density* is of particular interest.

In a general graph $G = (V, E)$ with n vertices and m edges, the best running time for computing a maximum matching in G depends on the ratio m/n. The classic algorithm of Micali and Vazirani [20, 27] is based on augmenting paths, and it finds a maximum matching in $O(\sqrt{n}m)$ time. Mucha and Sankowski [22] use algebraic tools to achieve running time $O(n^\omega)$. More recently, Mądry [19] showed that an approach through interior-point methods yields an algorithm with running time roughly $O(m^{10/7})$. As we shall see, for $G \in \mathbb{G}_\rho$, we have $m = O(\rho n)$, and this bound is asymptotically tight. Thus, for $G \in \mathbb{G}_\rho$, the running times of these three algorithms become $O(\rho n^{3/2})$, $O(n^\omega)$ and $O\big((\rho n)^{10/7}\big)$, respectively.

[1] In the literature, it is more common to assume $\omega \geq 2$. We adopt the stronger assumption $\omega > 2$ because it simplifies the bounds. If $\omega = 2$ is allowed, then the running times that we state have additional logarithmic factors.

There is a specialized algorithm for certain classes of *bipartite* geometric intersection graphs. Efrat, Itai, and Katz [9] show how to compute the maximum matching in bipartite unit disk graphs in $O(n^{3/2} \log n)$ time. Having bounded density does not help in this algorithm; it has $O(\sqrt{n})$ rounds, each of which needs $\Omega(n)$ time. The same approach can be used for other geometric shapes if a certain semi-dynamic data structure is available. In particular, using the data structure of Kaplan et al. [15] for additively-weighted nearest neighbors, finding a maximum matching in a bipartite intersection graph of disks takes $O(n^{3/2} \operatorname{polylog} n)$ time. We are not aware of any similar results for non-bipartite geometric intersection graphs.

We show that a maximum matching in a graph of $\mathbb{G}_\rho$ with n vertices can be computed in $O(\rho^{3\omega/2} n^{\omega/2}) = O(\rho^{3.56} n^{1.19})$ time. The algorithm is randomized and succeeds with high probability. It uses the algebraic approach by Mucha and Sankowski [23] for planar graphs with an extension by Yuster and Zwick [28] for H-minor-free graphs. As noted by Alon and Yuster [4], this approach works for *hereditary*[2] graph families with bounded average degree and small separators. We note that the algorithm can be used for graphs of $\mathbb{G}_\rho$, because we have average degree $O(\rho)$ and balanced separators of size $O(\sqrt{\rho n})$ [13, 25]. However, finding the actual dependency on ρ is difficult because it plays a role in the average degree, in the size of the separators, and the algorithm has a complex structure with several subroutines that must be distilled.

There are several noteworthy features in our approach. For one, we solve a geometric problem using linear algebra, namely Gaussian elimination. The use of geometry is limited to finding separators, bounding the degree, and constructing the graph explicitly. Note that the role of subgraphs in the definition of $\mathbb{G}_\rho$ is a key feature in our algorithm. On the one hand, we need a hereditary family of graphs, as needed to apply the algorithm. On the other hand, it brings more generality; for example, it includes the case of bipartite graphs defined by colored geometric objects.

Compared to the work of Efrat, Itai, and Katz [9], our algorithm is for arbitrary subgraphs of geometric intersection graphs, not only bipartite ones; it works for any objects, as it does not use advanced data structures that may depend on the shapes. On the other hand, it needs the assumption of low density. Compared to previous algorithms for arbitrary graphs and ignoring polylogarithmic factors, our algorithm is faster when $\rho = o(n^{(20-7\omega)/(21\omega-20)})$. Using the current bound $\omega < 2.373$, this means that our new algorithm is faster for $\rho = O(n^{0.113})$.

Our matching algorithm also applies for intersection graphs of objects in 3-dimensional space. However, in this case there is no algorithmic gain with the current bounds on ω: one gets a running time of $O(n^{2\omega/3})$ when $\rho = O(1)$, which is worse than constructing the graph explicitly and using the algorithm of Micali and Vazirani or the algorithm of Mądry.

Sparsification – Reducing to bounded depth

Consider a family of convex geometric objects $\mathcal{U}$ in the plane where each object contains a square of side length 1 and is contained in a square of side length $\Psi \geq 1$. Our objective is to compute a maximum matching in the intersection graph $G_\mathcal{U}$.[3] Our goal is to transform this problem to finding a maximum matching in the intersection graph of a subfamily $\mathcal{U}' \subset \mathcal{U}$ with bounded depth. Then we can employ our result from above for $G_{\mathcal{U}'}$ or, more generally, any algorithm for maximum matching (taking advantage of the sparsity of the new instance).

[2] closed under taking subgraphs

[3] Note that here we do not consider subgraphs of $G_\mathcal{U}$; we need the whole subgraph $G_\mathcal{U}$.

We describe a method that is fairly general and works under comparatively mild assumptions and also in higher dimensions. However, for an efficient implementation, we require that the objects under considerations support certain range searching operations efficiently. We discuss how this can be done for disks of arbitrary sizes, translates of a fixed convex shape in the plane, axis-parallel objects in constant dimension, and (unit) balls in constant dimension. In all these cases, we obtain a subquadratic time algorithm for finding a maximum matching, assuming that Ψ is small. We mostly focus on the planar case, mentioning higher dimensions as appropriate.

As particular results to highlight, we show that a maximum matching in the intersection graph of any family of translates of a convex object in the plane can be found in $O(n^{\omega/2})$ time with high probability, and a maximum matching in the intersection graph of a family of planar disks with radii in $[1, \Psi]$ can be found in $O(\Psi^6 \log^{11} n + \Psi^{12\omega} n^{\omega/2})$ time with high probability.

Organization

We begin with some general definitions and basic properties of geometric intersection graphs (Section 2). Then, in the first part of the paper, we present the new algorithm for finding a maximum matching in geometric intersection graphs of low density (Section 3). In the second part, we present our sparsification method. This is done in two steps. First, we describe a generic algorithm that works for general families of shapes that have roughly the same size, assuming that certain geometric operations can be performed quickly. (Section 4). Second, we explain how to implement these operations for several specific shape families, e.g., translates of a given convex objects and disks of bounded radius ratio (Section 5). The two parts are basically independent, where the second part uses the result from the first part as a black box, to state the desired running times. All the proofs are deferred to the long version [5].

2 Basics of (geometric intersection) graphs

Geometric objects

Several of our algorithms work under fairly weak assumptions on the geometric input: we assume that the objects in $\mathcal{U}$ have *constant description complexity*. This means that the boundary of each object is a continuous closed curve whose graph is a semialgebraic set, defined by a constant number of polynomial equalities and inequalities of constant maximum degree. For later algorithms we restrict attention to some particular geometric objects, like disks or squares.

To operate on $\mathcal{U}$, we require that our computational model supports primitive operations that involve a constant number of objects of $\mathcal{U}$ in constant time, e.g., finding the intersection points of two boundary curves; finding the intersection points between a boundary curve and a disk or a vertical line; testing whether a point lies inside, outside, or on the boundary of an object; decomposing a boundary curve into x-monotone pieces, etc. See, e.g., [15] for a further discussion and justification of these assumptions.

We emphasize that in addition to the primitives on the input objects, we do not require any special constant-time operations. In particular, even though our algorithms use algebraic techniques such as fast matrix multiplication or Gaussian elimination, we rely only on algebraic operations over $\mathbb{Z}_p$, where $p = \Theta(n^4)$. Thus, when analyzing the running time of our algorithms, we do not need to worry about the bit complexity of these operations.

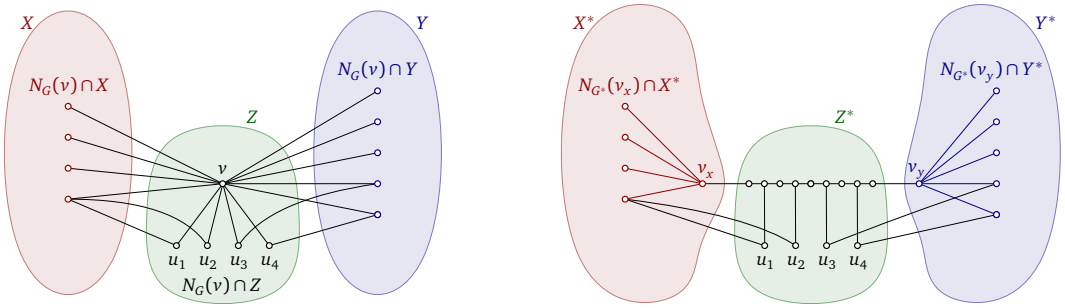


Figure 1 Splitting one single vertex of Z.

Geometric intersection graphs

The following well-known lemma bounds $|G_{\mathcal{U}}|$ in terms of ρ, and the time to construct $G_{\mathcal{U}}$.

▶ **Lemma 1.** *If $\mathcal{U}$ has n objects and density ρ, then $G_{\mathcal{U}}$ has at most $(\rho-1)n$ edges (this holds in any dimension). If $\mathcal{U}$ consists of objects in the plane, then $G_{\mathcal{U}}$ can be constructed in $O(\rho n \log n)$ time.*

Separators in geometric intersection graphs

The classic *planar separator theorem* by Lipton and Tarjan [8, 18] shows that any planar graph can be decomposed in a balanced way by removing a small number of vertices. Even though geometric intersection graphs can be far from planar, similar results are also available for them. These results are usually parameterized by the *depth* of the arrangement or by the *area* of the separator and the components [3, 10, 21]. The following recent result provides a small separator for general intersection graphs of bounded density.

▶ **Theorem 2** (Lemma 2.21 in [13]). *Let $\mathcal{U}$ be a set of n objects in $\mathbb{R}^2$ with density ρ. In $O(n)$ expected time, we can find a circle $\mathbb{S}$ such that $\mathbb{S}$ intersects at most $c\sqrt{\rho n}$ objects of $\mathcal{U}$, the exterior of $\mathbb{S}$ contains at most αn elements of $\mathcal{U}$, and the interior of $\mathbb{S}$ contains at most αn elements of $\mathcal{U}$. Here $0 < c$ and $0 < \alpha < 1$ are universal constants, independent of ρ and n.*

The proof of Theorem 2 goes roughly as follows: Pick a point in each object of $\mathcal{U}$, compute the smallest circle $\mathbb{S}'$ (or an approximation thereof) that contains, say, $n/20$ points, and then take a concentric scaled copy $\mathbb{S}$ of $\mathbb{S}'$, with scale factor uniformly at random in $[1, 2]$. With constant probability, the circle $\mathbb{S}'$ has the desired property. This can be checked easily in linear time by determining which objects of $\mathcal{U}$ are inside, outside, or intersected by $\mathbb{S}$. In expectation, a constant number of repetitions is needed to obtain the desired circle.

A family $\mathbb{G}$ of graphs is *hereditary* if for every $G \in \mathbb{G}$, it holds that all subgraphs H of G are also in $\mathbb{G}$. By definition, our family $\mathbb{G}_\rho$ of subgraphs of geometric intersection graphs with density ρ is hereditary. A graph G is ***δ-sparse*** if every subgraph H of G has at most $\delta|V(H)|$ edges. Lemma 1 implies that all graphs in $\mathbb{G}_\rho$ are ρ-sparse.

Consider a graph G and a vertex v of G. A ***vertex split*** at v consists of adding a pendant 2-path $vv'v''$, where v' and v'' are new vertices, and possibly replacing some edges uv incident to v by new edges uv''; see Figure 1 for a sequence of splits. We note that a vertex split may not replace any edges. In this case, we are just adding a pendant path of length 2.

Let G' be a graph obtained from G by a sequence of k vertex splits. Then, the size of a maximum matching in G' is the size of a maximum matching in G plus k. Furthermore,

from a maximum matching in G', we can easily obtain a maximum matching in G in $O(|V(G)| + |E(G)| + k)$ time. We will use vertex splits to ensure that the resulting graphs have bounded degree and a vertex set of a certain cardinality. (A vertex split may change the density, but that will not be important.) Note that if we perform a vertex split at v in a graph of $\mathbb{G}_\rho$, in general we obtain a graph of $\mathbb{G}_{\rho+2}$ because we can represent it by making two new copies of the object corresponding to v. Nevertheless, this increase in the density will not be problematic in our algorithm.

3 Maximum matching in low-density geometric intersection graphs

3.1 Separators and separator trees

A graph G has a (k, α)-separation if $V(G)$ can be partitioned into three pairwise disjoint sets X, Y, Z such that $|X \cup Z| \leq \alpha|V|$, $|Y \cup Z| \leq \alpha|V|$, $|Z| \leq k$, and such that there is no edge with one endpoint in X and one endpoint in Y. We say that Z *separates* X and Y. At the cost of making the constant α larger, we can restrict our attention to graphs of a certain minimum size.

Theorem 2 gives a $(c\sqrt{\rho n}, \alpha')$-separation for every graph of $\mathbb{G}_\rho$, for some constant $\alpha' < 1$. (A separator in $G_{\mathcal{U}}$ is a separator in each subgraph of $G_{\mathcal{U}}$.) Furthermore, such a separation can be computed in expected linear time, if the objects defining the graph are available.

A recursive application of separations can be represented as a binary rooted tree. We will use so-called (weak) *separator trees*, where the separator does not go into the subproblems. In such a tree, we store the separator at the root and recurse on each side to obtain the subtrees. We want to have small separators and balanced partitions at each level of the recursion, and we finish the recursion when we get to problems of a certain size. This leads to the following definition. Let $\gamma > 0$, $0 < \beta < 1$, and $0 < \alpha < 1$ be constants. We say that a graph G has a (γ, β, α)-***separator tree*** if there is a rooted binary full tree T with the following properties:

- **(i)** Each node $t \in T$ is associated with some set $Z_t \subseteq V(G)$.
- **(ii)** The sets Z_t, $t \in T$, partition $V(G)$, i.e., $\bigcup_{t \in T} Z_t = V(G)$, and $Z_t \cap Z_{t'} = \emptyset$, for distinct $t, t' \in T$.
- **(iii)** For each node $t \in T$, let $V_t = \bigcup_s Z_s$, where s ranges over the descendants of t (including t). Note that if t is an internal node with children u and v, then V_t is the disjoint union of Z_t, V_u, and V_v. If t is a leaf, then $V_t = Z_t$.
- **(iv)** For each internal node $t \in T$ with children u and v, (V_u, V_v, Z_t) is a $(\gamma m^\beta, \alpha)$-separation for $G[V_t]$, the subgraph of G induced by V_t, where $m = |V_t| = |Z_t| + |V_u| + |V_v|$.
- **(v)** For each leaf $t \in T$, we have $|V_t| \leq \Theta(\gamma^{1/(1-\beta)})$. We have chosen the size so that V_t is a $(\gamma|V_t|^\beta, \alpha)$-separator for the whole induced subgraph $G[V_t]$.

Yuster and Zwick [28] provide an algorithm that computes a separator tree of some split graph for a given graph from an H-minor-free family. As Alon and Yuster [4, Lemma 2.13] point out, this algorithm actually works for any δ-sparse hereditary graph family, as long as δ is constant. Thus, the result applies to $\mathbb{G}_\rho$. We revise the construction to make the dependency on ρ explicit.

▶ **Lemma 3.** *Given a graph G of $\mathbb{G}_\rho$ with n vertices, we can compute in $O(\rho n \log n)$ expected time a vertex-split graph G' of G and a separator tree T' for G' with the following properties:*

- **(i)** *the graph G' has $\Theta(\rho n)$ vertices and edges;*
- **(ii)** *the maximum degree of G' is at most 4;*
- **(iii)** *T' is a $(\gamma = O(\rho), \beta = 1/2, \alpha)$-separator tree for G', where $\alpha < 1$ is a constant (independent of ρ and n).*

Note that the split graph G' in Lemma 3 is not necessarily in $\mathbb{G}_\rho$. It is a subgraph of an intersection graph, but since we introduce copies of geometric objects when we split vertices, the density increases. In any case, this does not matter because G' will be accessed through the separator tree T'.

3.2 Nested dissection

We will need to compute with matrices. The arithmetic operations take place in $\mathbb{Z}_p$, where $p = \Theta(n^4)$ is prime. Thus, we work with numbers of $O(\log n)$-bits, and we simply need to bound the number of arithmetic operations. Using a word-RAM model of computation, each arithmetic operation needs constant time.

Let A be an $n \times n$ matrix. A Gaussian elimination step on row i is the following operation: for $j = i + 1, \ldots, n$, add an appropriate multiple of row i to row j so that the element at position (j, i) becomes 0. Elimination on row i can be performed if the entry at position (i, i) is nonzero. Gaussian elimination on A consists of performing Gaussian elimination steps on rows $i = 1, \ldots, n - 1$. This is equivalent to computing an LU decomposition of A, where L is a lower triangular matrix with units along the diagonal, and U is an upper triangular matrix. Gaussian elimination is performed *without pivoting* if, for all i, when we are about to do a Gaussian elimination step on row i, the entry at position (i, i) is non-zero. If Gaussian elimination is performed without pivoting, then the matrix is non-singular. (Pivoting is permuting the rows to ensure that the entry at position (i, i) is non-zero.)

Let $[n] = \{1, \ldots, n\}$. The ***representing graph*** $G(A)$ of an $n \times n$ matrix $A = (a_{i,j})_{i,j \in [n]}$ is

$$G(A) = \left([n], \left\{ ij \in \binom{[n]}{2} \mid a_{i,j} \neq 0 \text{ or } a_{j,i} \neq 0 \right\} \right).$$

Let T be a separator tree for $G(A)$. The row order of A is ***consistent*** with T if, whenever t' is an ancestor of t, all the rows of Z_t are before any row of $Z_{t'}$. We may assume that all the rows of Z_t are consecutive. In particular, if the rows are ordered according to a post-order traversal of T, then the row order of A is consistent with T. A careful but simple revision of the nested dissection algorithm by Gilbert and Tarjan [12] leads to the following theorem.

▶ **Theorem 4.** *Let A be an $n \times n$ matrix such that the representing graph $G(A)$ has bounded degree and assume that we are given a (γ, β, α)-separator tree T for $G(A)$, were $\gamma > 0$, $0 < \alpha < 1$, and $1/2 < \beta < 1$ are constants. Furthermore, assume that the row order of A is consistent with T and that Gaussian elimination on A is done without pivoting. We can perform Gaussian elimination (without pivoting) on A and find a factorization $A = LU$ of A in $O(\gamma^\omega n^{\beta\omega})$ time, where L is a lower triangular matrix with units along the diagonal and U is an upper triangular matrix.*

To prove Theorem 4, we need the following folklore lemma.

▶ **Lemma 5.** *Let A be an $n \times n$ matrix, and $k \leq n$. Suppose that Gaussian elimination on the first k rows of A needs no pivoting. Then, we can perform Gaussian elimination on the first k rows of A with $O(n^2 k^{\omega-2})$ arithmetic operations.*

Remark 1: Mucha and Sankowski [23] noted that the result holds when $G(A)$ is planar or, more generally, has recursive separators, using the approach by Lipton, Rose, and Tarjan [17] for nested dissection. This approach is based on the *strong* separator tree. Alon, Yuster, and Zwick [4, 28] showed that a similar result holds for graphs of bounded degree with recursive

separators if one instead uses the nested dissection given by Gilbert and Tarjan [12]. In this case, we need bounded degree, but a weak separator tree suffices. Again, since we want to make the dependency on ρ explicit and since the analysis in terms of matrix multiplication time does not seem to be written down in detail anywhere, we revise the method carefully.

Remark 2: Usually, the result is stated for symmetric positive definite matrices. Reindexing a symmetric positive definite matrix gives another symmetric positive definite matrix, and Gaussian elimination on such matrices can always be performed without pivoting. Thus, for positive semidefinite matrices, we do not need to assume that the row order is consistent with T because we can reorder the rows to make it consistent with T. However, Mucha and Sankowski [23] do need the general statement in their Section 4.2, and they mention this general case after their Theorem 13. Actually, they need it over $\mathbb{Z}_p$, where the concept of positive definiteness is not even defined!

3.3 The algorithm

Assume we have a graph G of $\mathbb{G}_\rho$ with n vertices and a geometric representation, i.e., geometric objects $\mathcal{U}$ of density at most ρ such that G is a subgraph of $G_\mathcal{U}$. We want to compute a maximum matching for G. For this, we adapt the algorithm of Mucha and Sankowski [23]. We provide an overview of the approach, explain the necessary modifications, and emphasize the dependency on ρ in the different parts of the algorithm.

Using Lemma 3, we get in $O(\rho n \log n)$ expected time a vertex-split graph G' of G and a separator tree T' for G' such that:

- **(i)** the graph G' has $\Theta(\rho n)$ vertices and edges;
- **(ii)** the maximum degree of G' is at most 4;
- **(iii)** T' is a $(\gamma = O(\rho), \beta = 1/2, \alpha)$-separator tree for G', where $\alpha < 1$ is a constant (independent of ρ and n).

Since G' is obtained from G by vertex splits, it suffices to find a maximum matching in G'. We set $m = |V(G')| = \Theta(\rho n)$, and we label the vertices of G' from 1 to m. We consider the variables $X = (x_{ij})_{ij \in E(G')}$; i.e., each edge ij of G defines a variable x_{ij}. Consider the $m \times m$ symbolic matrix $A[X] = A[X](G')$, defined as follows:

$$(A[X])_{i,j} = \begin{cases} x_{ij}, & \text{if } ij \in E(G') \text{ and } i < j, \\ -x_{ij}, & \text{if } ij \in E(G') \text{ and } j < i, \\ 0 & \text{otherwise.} \end{cases}$$

The symbolic matrix $A[X]$ is usually called the *Tutte matrix* of G'. It is known [24] that the rank of $A[X]$ is twice the size of the maximum matching in G'. In particular, G' has a perfect matching if and only if $\det(A[X])$ is not identically zero. Take a prime $p = \Theta(n^4)$, and substitute each variable in $A[X]$ with a value from $\mathbb{Z}_p$, each chosen independently uniformly at random. Let A be the resulting matrix. Then, with high probability, $\operatorname{rank}(A) = \operatorname{rank}(A[X])$, where on both sides we consider the rank over the field $\mathbb{Z}_p$.

From maximum matching to perfect matching

Let $B = AA^T$. Then, B is symmetric, and the rank of B equals the rank of A. Note that $(B)_{i,j}$ is nonzero only if i and j share a neighbor in G'. Since G' has bounded degree, from the separator tree T' for G', we can obtain a separator tree T_B for the representing graph $G(B)$.

Since T' was a $(\gamma = O(\rho), \beta = 1/2, \alpha)$-separator tree for G', T_B is a $(\gamma = O(\rho), \beta = 1/2, \alpha)$-separator tree for $G(B)$, where the constant hidden in $O(\rho)$ is increased by the maximum degree in G'. Using Theorem 4, we obtain that Gaussian elimination can be done in B in $O(\gamma^\omega m^{\omega/2}) = O(\rho^\omega(\rho n)^{\omega/2}) = O(\rho^{3\omega/2} n^{\omega/2})$ time, assuming that pivoting is not needed.

Mucha and Sankowski [23, Section 5] show how Gaussian elimination without pivoting can be used in B to find a collection of indices $W \subseteq [m]$ such that the centered matrix $(B)_{W,W}$, defined by rows and columns of B with indices in W, has the same rank as B. It follows that $\operatorname{rank}(A_{W,W}) = \operatorname{rank}(B_{W,W})$ and therefore $G'[W]$ contains a maximum matching of G' that is a perfect matching in $G'[W]$ (with high probability). The key insight to find such W is that, if during Gaussian elimination in B we run into a 0 along the diagonal, then the whole row and column are 0, which means that they can be removed from the matrix without affecting the rank. We summarize.

▶ **Lemma 6.** *In time $O(\rho^{3\omega/2} n^{\omega/2})$ we can find a subset W of vertices of G' such that, with high probability, $G'[W]$ has a perfect matching that is a maximum matching in G'.*

From now on, we can assume that G' has a perfect matching. We keep denoting by T' its separator tree, by A the matrix after substituting values of $\mathbb{Z}_p$ into $A[X]$, and by B the matrix AA^T. (We can compute the tree T' anew or we can reuse the same separator tree restricted to the subset of vertices.) Let Z_r denote the set stored at the root r of T'. Thus, Z_r is the first separator on G'. Let N_r be the set Z_r together with its neighbors in G'. Because G' has bounded degree, we have $|N_r| = O(|Z_r|) = O(\rho m^{1/2}) = O(\rho^{3/2} n^{1/2})$.

Mucha and Sankowski show how to compute with $O(1)$ Gaussian eliminations a matching M' in G' that covers all the vertices of Z_r and is contained in some perfect matching of G'. There are two ingredients for this. The first ingredient is to use Gaussian elimination on the matrix $B = AA^T$ to obtain a decomposition $AA^T = LDL^T$, and then use (partial) Gaussian elimination on a matrix C composed of $L_{[m],N_r}$ and $A_{N_r,[m]\setminus N_r}$ to compute $(A^{-1})_{N_r,N_r}$. (Note that in general $(A^{-1})_{N_r,N_r}$ is different from $(A_{N_r,N_r})^{-1}$. Computing the latter is simpler, while computing the former is a major insight by Mucha and Sankowski [23, Section 4.2].) Interestingly, T' is also a separator tree for the representing graph of this matrix C, and Gaussian elimination can be performed without pivoting. Thus, we can obtain in $O(\rho^\omega m^{\omega/2}) = O(\rho^{3\omega/2} n^{\omega/2})$ time the matrix $(A^{-1})_{N_r,N_r}$. The second ingredient is that, once we have $(A^{-1})_{N_r,N_r}$, we can compute for any matching M' contained in $G'[N_r]$ a maximal (with respect to inclusion) submatching M' that is contained in a perfect matching of G'. This is based on an observation by Rabin and Vazirani [24] that shows how to find edges that belong to some perfect matching using the inverse matrix, and Gaussian elimination on the matrix $(A^{-1})_{N,N}$ to identify subsets of edges that together belong to some perfect matching. The matrix $(A^{-1})_{N_r,N_r}$ is not necessarily represented by a graph with nice separators, but it is of size $|N_r| \times |N_r|$. Thus, Gaussian elimination in $(A^{-1})_{N_r,N_r}$ takes $O(|N_r|^\omega) = O(\rho^{3\omega/2} n^{\omega/2})$ time [23, Section 2.4].

Since the graph G' has bounded maximum degree, making $O(1)$ iterations of finding a maximal matching M' in $G'[N_r]$, followed by finding a maximal subset M'' of M' contained in a perfect matching of G', and removing the vertices contained in M' plus the edges of $M' \setminus M''$, gives a matching M_* that covers Z_r and is contained in a perfect matching of G'; see [23, Section 4.3]. The vertices of M_* can be removed, and we recurse on both sides of $G' - V(M_*) \subset G' - Z_r$ using the corresponding subtrees of T'. The running time is $T(n) = O(\rho^{3\omega/2} n^{\omega/2}) + T(n_1) + T(n_2)$, where $n_1, n_2 \leq \alpha n$. This solves to $T(n) = O(\rho^{3\omega/2} n^{\omega/2})$ because $\omega/2 > 1$. We summarize in the following result. If only the family $\mathcal{U}$ is given, first we use Lemma 1 to construct $G_{\mathcal{U}}$.

▶ **Theorem 7.** *Given a graph G of $\mathbb{G}_\rho$ with n vertices together with a family $\mathcal{U}$ of geometric objects with density ρ such that G is a subgraph of $G_\mathcal{U}$, we can find in $O(\rho^{3\omega/2}n^{\omega/2})$ time a matching in G that, with high probability, is maximum. In particular, for a family $\mathcal{U}$ of n geometric objects with density ρ, a maximum matching in $G_\mathcal{U}$ can be found in $O(\rho^{3\omega/2}n^{\omega/2})$ time. The same holds for the bipartite or k-partite version of $G_\mathcal{U}$.*

4 Sparsification

Let $\mathcal{U}$ be a family of convex geometric objects in the plane such that each object of $\mathcal{U}$ contains a square of side length 1 and is contained in a square of side length $\Psi \geq 1$. Through the discussion we will treat Ψ as a parameter. Our objective is to reduce the problem of computing a maximum matching in the intersection graph $G_\mathcal{U}$ to the problem of computing a maximum matching in $G_\mathcal{W}$ for some $\mathcal{W} \subseteq \mathcal{U}$ of small depth.

Let $P = \mathbb{Z}^2$ be the points in the plane with integer coordinates. Each square of unit side length contains at least one point of P and each square of side length Ψ contains at most $(1+\Psi)^2 = O(\Psi^2)$ points of P. In particular, each object $U \in \mathcal{U}$ contains at least one and at most $O(\Psi^2)$ points from P.

First we provide an overview of the idea. The objects intersected by a point $p \in P$ define a clique, and thus any even number of them defines a perfect matching. We show that, for each $p \in P$, it suffices to keep a few objects pierced by p, and we show how to obtain such a suitable subfamily. The actual number of objects to keep depends on Ψ, and whether the actual computation can be done efficiently depends on the geometric shape of the objects.

For each object $U \in \mathcal{U}$, we find the lexicographically smallest point in $P \cap U$. We assume that we have a primitive operation to compute $P \cap U$ for each object $U \in \mathcal{U}$ in $O(1 + |P \cap U|) = O(\Psi^2)$ time. A simple manipulation of these incidences allows us to obtain the ***clusters***

$$\mathcal{U}_p \;=\; \{U \in \mathcal{U} \mid p \text{ lexicographically minimum in } P \cap U\}, \qquad \text{for all } p \in P.$$

Note that the clusters $\mathcal{U}_p$, for $p \in P$, form a partition of $\mathcal{U}$. This will be useful later. Clearly, the subgraph of $G_\mathcal{U}$ induced by $\mathcal{U}_p$ is a clique, for each $p \in P$.

We will use the usual notation

$$E(\mathcal{U}_p, \mathcal{U}_q) \;=\; \{UV \mid U \in \mathcal{U}_p,\, V \in \mathcal{U}_q, U \cap V \neq \emptyset\} \;\subseteq\; E(G_\mathcal{U}).$$

The ***pattern graph*** $H = H(P, \Psi)$ has vertex set P and set of edges

$$E(H) \;=\; \{pq \mid \|p - q\|_\infty \leq 2\Psi\} \;\subseteq\; \binom{P}{2}.$$

The use of the pattern graph is encoded in the following property: if $U \in \mathcal{U}_p$, $V \in \mathcal{U}_q$ and $U \cap V \neq \emptyset$, then $pq \in E(H)$. Indeed, if U and V intersect, then the union $U \cup V$ is contained in a square of side length 2Ψ, and thus the L_∞-distance between each $p \in P \cap U$ and $q \in P \cap V$ is at most 2Ψ.

The definition of $H(P, \Psi)$ implies that the edge set of $G_\mathcal{U}$ is the disjoint union of $E(\mathcal{U}_p, \mathcal{U}_q)$, over all $pq \in E(H)$, and the edge sets of the cliques $G_{\mathcal{U}_p}$, over all $p \in P$. Thus, whenever $pq \notin E(H)$, there are no edges in $E(\mathcal{U}_p, \mathcal{U}_q)$.

Let λ be the maximum degree of H. Note that $\lambda = O(\Psi^2)$. The value of λ is an upper bound on how many clusters $\mathcal{U}_q$ may interact with a single cluster $\mathcal{U}_p$. We will use λ as a parameter to decide how many objects from each $\mathcal{U}_p$ are kept. We start with a simple observation.

▶ **Lemma 8.** *There exists a maximum matching in $G_{\mathcal{U}}$ that, for all $pq \in E(H)$, contains at most one edge of $E(\mathcal{U}_p, \mathcal{U}_q)$.*

Of course we do not know which object from the cluster $\mathcal{U}_p$ will interact with another cluster $\mathcal{U}_q$. We will explain how to get a large enough subset of cluster $\mathcal{U}_p$.

For each $pq \in E(H)$, we construct a set $\mathcal{W}(p,q) \subseteq \mathcal{U}_p \cup \mathcal{U}_q$ as follows. First, we construct a matching $M = M(p,q)$ in $E(\mathcal{U}_p, \mathcal{U}_q)$ such that M has $2\lambda + 1$ edges or M has fewer than $2\lambda + 1$ edges and is maximal in $E(\mathcal{U}_p, \mathcal{U}_q)$. For example, such a matching can be constructed incrementally. If M has $2\lambda + 1$ edges, we take $\mathcal{W}(p,q)$ to be the endpoints of M. Otherwise, for each endpoint $U \in \mathcal{U}_p$ (resp. $V \in \mathcal{U}_q$) of M, we place U (resp. V) and λ of its neighbors from $\mathcal{U}_q$ (resp. $\mathcal{U}_p$) into $\mathcal{W}(p,q)$. When U (resp. V) has fewer than λ neighbors, we place all its neighbors in $\mathcal{W}(p,q)$. This finishes the description of $\mathcal{W}(p,q)$; refer to Algorithm *Sparsify-one-edge* in the appendix (Figure 2) for pseudo-code.

Algorithm *Sparsify-one-edge*
Input: p, q, $\mathcal{U}_p$ and $\mathcal{U}_q$
Output: $\mathcal{W}(p,q)$

$\mathcal{A}_p \leftarrow \mathcal{U}_p$
$\mathcal{A}_q \leftarrow \mathcal{U}_q$
$(*$ compute matching M $*)$
$M \leftarrow \emptyset$
while $|M| < 2\lambda + 1$ **and** $\mathcal{A}_p \neq \emptyset$ **do**
    $U \leftarrow$ an arbitrary object of $\mathcal{A}_p$
    if U intersects some $V \in \mathcal{A}_q$ **then**
        $M \leftarrow M \cup \{UV\}$
        $\mathcal{A}_q \leftarrow \mathcal{A}_q \setminus \{V\}$
    $\mathcal{A}_p \leftarrow \mathcal{A}_p \setminus \{U\}$
$(*$ end of computation of M $*)$
$\mathcal{W} \leftarrow \cup_{UV \in M} \{U, V\}$ $(*$ endpoints of M $*)$
if $|M| = 2\lambda + 1$ **then** $(*$ M large enough matching $*)$
    return $\mathcal{W}$
else $(*$ M maximal but small; add neighbors of $\mathcal{W}$ to the output $*)$
    $\mathcal{W}' \leftarrow \mathcal{W}$
    for $W \in \mathcal{W}$ **do**
        if $W \in \mathcal{U}_p$ **then**
            add to $\mathcal{W}'$ $\min\{\lambda, |E(\{W\}, \mathcal{U}_q)|\}$ elements of $\mathcal{U}_q$ intersecting W
        else $(*$ $W \in \mathcal{U}_p$ $*)$
            add to $\mathcal{W}'$ $\min\{\lambda, |E(\mathcal{U}_p, \{W\})|\}$ elements of $\mathcal{U}_p$ intersecting W
    return $\mathcal{W}'$

Figure 2 Algorithm *Sparsify-one-edge*.

▶ **Lemma 9.** *A maximum matching in*

$$\tilde{G} = \left(\bigcup_{pq \in E(H)} G_{\mathcal{W}(p,q)} \right) \cup \left(\bigcup_{p \in P} G_{\mathcal{U}_p} \right).$$

is a maximum matching in $G_{\mathcal{U}}$.

▶ **Lemma 10.** *The family of objects* $\mathcal{W} = \cup_{pq \in E(H)} \mathcal{W}(p,q)$ *has depth* $O(\Psi^8)$.

▶ **Theorem 11.** *Let* $\mathcal{U}$ *be a family of* n *geometric objects in the plane such that each object of* $\mathcal{U}$ *contains a square of side length* 1 *and is contained in a square of side length* Ψ. *Suppose that, for any* $m \in \mathbb{N}$ *and for any* $p, q \in \mathbb{Z}^2$ *with* $|\mathcal{U}_p| + |\mathcal{U}_q| \leq m$, *we can compute the sparsification* $\mathcal{W}(p,q)$ *as described above in time* $T_{\text{spars}}(m)$, *where* $T_{\text{spars}}(m) = \Omega(m)$ *is convex. In* $O(\Psi^2 \cdot T_{\text{spars}}(n))$ *time we can reduce the problem of finding a maximum matching in* $G_{\mathcal{U}}$ *to the problem of finding a maximum matching in* $G_{\mathcal{W}}$ *for some* $\mathcal{W} \subseteq \mathcal{U}$ *with maximum depth* $O(\Psi^8)$.

Our use of properties in the plane is very mild, and similar results hold in any space with constant dimension.

▶ **Theorem 12.** *Let* $d \geq 3$ *be a constant. Let* $\mathcal{U}$ *be a family of* n *geometric objects in* $\mathbb{R}^d$ *such that each object of* $\mathcal{U}$ *contains a cube of side length* 1 *and is contained in a cube of side length* Ψ. *Suppose that, for any* $m \in \mathbb{N}$ *and for any* $p, q \in \mathbb{Z}^d$ *with* $|\mathcal{U}_p| + |\mathcal{U}_q| \leq m$, *we can compute the sparsification* $\mathcal{W}(p,q)$ *as described above in time* $T_{\text{spars}}(m)$, *where* $T_{\text{spars}}(m) = \Omega(m)$ *is convex. In* $O(\Psi^d \cdot T_{\text{spars}}(n))$ *time we can reduce the problem of finding a maximum matching in* $G_{\mathcal{U}}$ *to the problem of finding a maximum matching in* $G_{\mathcal{W}}$ *for some* $\mathcal{W} \subseteq \mathcal{U}$ *with maximum depth* $(1+\Psi)^{O(d)}$.

As we mentioned in the introduction, for fat objects, bounded depth implies bounded density; see Har-Peled and Quanrud [13, Lemma 2.7]. If a convex object contains a cube of unit side length and is contained in a cube of side length Ψ, then it is $O(1/\Psi)$-fat; see van der Stappen et al. [26], where the parameter $1/\Psi$ goes under the name of thickness. Combining both results, one obtains that the relation between depth and density differs by a factor of Ψ. For fixed shapes, they depth and density differ by a constant factor.

5 Efficient sparsification

Now, we implement Algorithm *Sparsify-one-edge* (Figure 2) efficiently. In particular, we must perform the test in line 7 and find the neighbors in line 19 (and the symmetric case in line 21). The shape of the geometric objects becomes relevant for this. First, we note that it suffices to obtain an efficient semi-dynamic data structure for intersection queries.

▶ **Lemma 13.** *Suppose there is a data structure with the following properties: for any* $m \in \mathbb{N}$ *and for any* $p, q \in \mathbb{Z}^2$ *with* $|\mathcal{U}_p| + |\mathcal{U}_q| \leq m$, *we can maintain a set* $\mathcal{A}_q \subseteq \mathcal{U}_q$ *under deletions so that, for any query* $U \in \mathcal{U}_p$, *we either find some* $V \in \mathcal{A}_q$ *with* $U \cap V \neq \emptyset$ *or correctly report that no such* V *exists. Let* $T_{\text{con}}(m)$ *be the time to construct the data structure,* $T_{\text{que}}(m)$ *an upper bound on the amortized query time, and* $T_{\text{del}}(m)$ *be an upper bound on the amortized deletion time. Then, the running time of Algorithm Sparsify-one-edge (Figure 2) for the input* $(p, q, \mathcal{U}_p, \mathcal{U}_q)$ *is* $T_{\text{sparse}}(m) = O(T_{\text{con}}(m) + mT_{\text{que}}(m) + \lambda^2 T_{\text{del}}(m))$.

5.1 Disks in the plane

When $\mathcal{U}$ consists of disks in the plane, we can use the data structure of Kaplan et al. [15] to sparsify an edge of the pattern graph. This leads to the following.

▶ **Proposition 14.** *Consider a family $\mathcal{U}$ of n disks in the plane with radii in $[1, \Psi]$. In $O(\Psi^6 n \log^{11} n)$ expected time, we can reduce the problem of finding a maximum matching in $G_{\mathcal{U}}$ to the problem of finding a maximum matching in $G_{\mathcal{W}}$ for some subfamily $\mathcal{W} \subseteq \mathcal{U}$ of disks with maximum depth $O(\Psi^8)$.*

Possibly, the method can be extended to homothets of a single object. For this one should consider the surfaces defined by weighted distances in the approach of Kaplan et al. [15].

Since the depth and the density of a family of disks are linearly related, Proposition 14 and Theorem 7 with $\rho = O(\Psi^8)$ imply the following.

▶ **Theorem 15.** *Consider a family $\mathcal{U}$ of n disks in the plane with radii in the interval $[1, \Psi]$. In $O(\Psi^6 n \log^{11} n + \Psi^{12\omega} n^{\omega/2})$ expected time, we can compute a matching in $G_{\mathcal{U}}$ that, with high probability, is maximum.*

5.2 Translates of a fixed convex shape in the plane

Now, suppose $\mathcal{U}$ consists of translates of a single convex object with non-empty interior in the plane. With an affine transformation, we ensure that the object is *fat*: the radii of the minimum enclosing disk and of the maximum enclosed disk are within a constant factor. Such a transformation is standard; e.g., [1, Lemma 3.2]. Thus, we may assume that $\Psi = O(1)$. We start with a standard lemma.

▶ **Lemma 16.** *Let $\mathcal{U}$ be a family of n translates of a convex object in the plane that are pierced by a given point q. The union of $\mathcal{U}$ can be computed in $O(n \log n)$ time.*

We will use the following lemma to "simulate" deletions. For this, we will keep a half-infinite interval of indices that contains the elements that are "deleted".

▶ **Lemma 17.** *Let $\mathcal{U} = \{U_1, \dots U_n\}$ be a family of n translates of a convex object in the plane that are pierced by a given point q. In $O(n \log^2 n)$ time, we can construct a data structure for the following queries: given $x \in \mathbb{R}^2$ and a value $a \in \{1, \dots, n\}$, find the smallest $i \geq a$ such that U_i contains x, or correctly report that x does not belong to $U_a \cup \dots \cup U_n$. The query time is $O(\log^2 n)$.*

▶ **Lemma 18.** *Let $\mathcal{U}_q = \{V_1, \dots V_n\}$ be a family of n translates of a convex object in the plane that are pierced by a given point q. Let U_0 be a convex object. In $O(n \log^2 n)$ time, we can construct a data structure for the following type of queries: given a translate U of U_0 and a value a, find the smallest $i \geq a$ such that U intersects V_i, or correctly report that U does not intersect $V_a \cup \dots \cup V_n$. Each query can be answered in $O(\log^2 n)$ time.*

Lemma 18 can be used to make queries and simulate deletions.

▶ **Proposition 19.** *Consider a family $\mathcal{U}$ of n translates of a convex object with non-empty interior in the plane. In $O(n \log^2 n)$ time, we can reduce the problem of finding a maximum matching in $G_{\mathcal{U}}$ to the problem of finding a maximum matching in $G_{\mathcal{W}}$ for some subfamily $\mathcal{W} \subseteq \mathcal{U}$ with maximum depth $O(1)$.*

Combining Proposition 19 and Theorem 7 we obtain the following.

▶ **Theorem 20.** *Consider a family $\mathcal{U}$ of translates of a convex object with non-empty interior in the plane. In $O(n^{\omega/2})$ time we can find matching in $G_{\mathcal{U}}$ that, with high probability, is maximum.*

If $\mathcal{U}$ consists of unit disks,the sparsification can be done slightly faster using a semi-dynamic data structure by Efrat, Itai, and Katz [9], which has $O(T_{\text{con}}(m)) = O(m \log m)$, and $O(T_{\text{op}}(m)) = O(\log m)$. However the current bottleneck is the computation of the maximum matching *after* the sparsification. Thus, improving the sparsification in the particular case of unit disks does not lead to an improved final algorithm.

Proposition 19 and Theorem 20 also holds if we have translations of $O(1)$ different convex objects (with nonempty interiors). Indeed, the data structure of Lemma 18 can be made for each pair of different convex shapes. In this case, the constant Ψ depends on the shapes, namely the size of the largest square that we can place inside each of the convex shapes and the size of the smallest square that can be used to cover each of the convex shapes. Also, the relation between the depth and the density depends on the shapes. However, for a fixed set of $O(1)$ shapes, both values are constants that depend on the shapes.

▶ **Theorem 21.** *Consider a family $\mathcal{U}$ of translates of a constant number of different convex objects in the plane with non-empty interiors. In $O(n^{\omega/2})$ time we can find matching in $G_{\mathcal{U}}$ that, with high probability, is maximum.*

5.3 Axis-parallel objects

A ***box*** is the Cartesian product of intervals. Combining standard data structures for orthogonal range searching [6, Sections 5.4 and 10.3] one obtains the following results.

▶ **Proposition 22.** *Let $d \geq 2$ be an integral constant. Consider a family $\mathcal{U}$ of n boxes in $\mathbb{R}^d$ such that each box of $\mathcal{U}$ contains a cube of side length 1 and is contained in a cube of side length Ψ. In $O(\Psi^d \cdot n \operatorname{polylog} n)$ time we can reduce the problem of finding a maximum matching in $G_{\mathcal{U}}$ to the problem of finding a maximum matching in $G_{\mathcal{W}}$, for some $\mathcal{W} \subseteq \mathcal{U}$ with maximum depth $(1+\Psi)^{O(d)}$.*

For $d = 2$, we can combine Theorem 7 and Proposition 22. Since we have assumed $\omega > 2$, the $O(n \operatorname{polylog} n)$ term is asymptotically smaller than $O(n^{\omega/2})$, and we obtain the following.

▶ **Theorem 23.** *Given a family $\mathcal{U}$ of n boxes in $\mathbb{R}^2$ such that each object of $\mathcal{U}$ contains a square of side length 1 and is contained in a square of side length Ψ, we can compute in $(1+\Psi)^{O(1)} n^{\omega/2}$ time a matching in $G_{\mathcal{U}}$ that, with high probability, is a maximum matching.*

Consider now the case $d \geq 3$. The set $\mathcal{W}$ that we obtain from Proposition 22 has depth and density $\rho = (1+\Psi)^{O(d)}$, and therefore the graph $G_{\mathcal{W}}$ has $O(\rho n)$ edges; see Lemma 1. We can thus use the algorithm of Mądry [19], which takes $\tilde{O}(|E(G_{\mathcal{W}})|^{10/7})) = \tilde{O}((1+\Psi)^{O(d)} n^{10/7})$ time. We summarize.

▶ **Corollary 24.** *Let $d \geq 3$ be an integral constant. Given a family $\mathcal{U}$ of n boxes in $\mathbb{R}^d$ such that each object of $\mathcal{U}$ contains a cube of side length 1 and is contained in a cube of side length Ψ, we can compute in $\tilde{O}((1+\Psi)^{O(d)} n^{10/7})$ time a matching in $G_{\mathcal{U}}$ that, with high probability, is a maximum matching.*

5.4 Congruent balls in $d \geq 3$ dimensions

Consider now the case of congruent balls in $\mathbb{R}^d$, for constant $d \geq 3$. Note that $\lambda = O(1)$ in this case. We use the dynamic data structure by Agarwal and Matoušek [2] for the sparsification. For each m with $n \leq m \leq n^{\lceil d/2 \rceil}$, the data structure maintains n points in $\mathbb{R}^d$, answers $O(n)$ queries for closest point and supports $O(\lambda^2)$ updates in

$$O\left(m^{1+\varepsilon} + \lambda^2 \frac{m^{1+\varepsilon}}{n} + n \cdot \frac{n \log^3 n}{m^{1/\lceil d/2 \rceil}}\right)$$

time. Here $\varepsilon > 0$, is an arbitrary constant whose choice affects to the constants hidden in the O-notation. For $d \in \{3, 4\}$, this running time is

$$O\left(m^{1+\varepsilon} + \lambda^2 \frac{m^{1+\varepsilon}}{n} + n \cdot \frac{n \log^3 n}{m^{1/2}}\right).$$

Setting $m = n^{4/3}$, we get a running time of $O(n^{4/3+\varepsilon} + \lambda^2 n^{1/3+\varepsilon}) = O(n^{4/3+\varepsilon})$ to handle $O(n)$ queries and $O(\lambda^2) = O(1)$ updates. Using this in Lemma 13 and Theorem 12, we get the following result

▶ **Proposition 25.** *Consider a family $\mathcal{U}$ of n unit balls objects in $\mathbb{R}^d$, for $d \in \{3, 4\}$. In $O(n^{4/3+\varepsilon})$ time, we can reduce the problem of finding a maximum matching in $G_{\mathcal{U}}$ to the problem of finding a maximum matching in $G_{\mathcal{W}}$ for some $\mathcal{W} \subseteq \mathcal{U}$ with maximum depth $O(1)$.*

For the resulting set $\mathcal{W}$ with depth $O(1)$, it is better to use the algorithm of Mądry [19] for sparse graphs. Note that $G_{\mathcal{W}}$ is sparse, and thus has $O(n)$ edges. Therefore, a maximum matching in $G_{\mathcal{W}}$ can be computed in $O(n^{10/7})$ time. In summary, we spend $O(n^{4/3+\varepsilon})$ for the sparsification and $O(n^{10/7})$ for computing the matching in the sparsified setting.

For $d > 4$, we set $m = n^{\frac{2\lceil d/2 \rceil}{1+\lceil d/2 \rceil}}$. The running time for the sparsification is then $O(n^{\frac{2\lceil d/2 \rceil}{1+\lceil d/2 \rceil}+\varepsilon})$. For each constant d, the resulting instance $G_{\mathcal{W}}$ has $O(n)$ edges. For $d = 5, 6$, the running time of the sparsification is $O(n^{3/2+\varepsilon})$. However, after the sparsification, we have a graph with $O(n)$ edges, and we can use the algorithm of Micali and Vazirani [20], which takes $O(n^{3/2})$ time. Thus, for $d \geq 5$, the running time is dominated by the sparsification.

▶ **Theorem 26.** *Let $d \geq 3$ be a constant. Consider a family $\mathcal{U}$ of congruent balls in $\mathbb{R}^d$. For $d = 3, 4$, we can find in $O(n^{10/7})$ time a maximum matching in $G_{\mathcal{U}}$. For $d \geq 5$, we can find in $O(n^{\frac{2\lceil d/2 \rceil}{1+\lceil d/2 \rceil}+\varepsilon})$ time a maximum matching in $G_{\mathcal{U}}$, for each $\varepsilon > 0$.*

6 Conclusion

We have proposed the density of a geometric intersection graph as a parameter for the maximum matching problem, and we showed that it can be fruitful in obtaining efficient matching algorithms. Then, we presented a sparsification method that lets us reduce the general problem to the case of bounded density for several interesting classes of geometric intersection graphs. In our sparsification method, we did not attempt to optimize the dependency on the radius ratio Ψ. It may well be that this can be improved by using more advanced grid-based techniques. Furthermore, our sparsification needs the complete intersection graph and does not apply to the bipartite setting. Here, we do not know of a method to reduce the general case to bounded density. In general, the complexity of the matching problem is wide open. To the best of our knowledge, there are no (even weak) superlinear lower bounds for the (static) matching problem in general graphs.

References

1 Pankaj. K. Agarwal, Sariel. Har-Peled, and Kasturi R. Varadarajan. Approximating extent measures of points. *J. ACM*, 51(4):606–635, 2004. doi:10.1145/1008731.1008736.

2 Pankaj K. Agarwal and Jiří Matoušek. Ray shooting and parametric search. *SIAM J. Comput.*, 22(4):794–806, 1993. doi:10.1137/0222051.

3 Jochen Alber and Jirí Fiala. Geometric separation and exact solutions for the parameterized independent set problem on disk graphs. *J. Algorithms*, 52(2):134–151, 2004.

4 Noga Alon and Raphael Yuster. Matrix sparsification and nested dissection over arbitrary fields. *J. ACM*, 60(4):25:1–25:18, 2013. doi:10.1145/2505989.

5 Édouard Bonnet, Sergio Cabello, and Wolfgang Mulzer. Maximum matchings in geometric intersection graphs. *CoRR*, abs/1910.02123, 2019. arXiv:1910.02123.

6 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. URL: http://www.worldcat.org/oclc/227584184.

7 Mark de Berg, A. Frank van der Stappen, Jules Vleugels, and Matthew J. Katz. Realistic input models for geometric algorithms. *Algorithmica*, 34(1):81–97, 2002. doi:10.1007/s00453-002-0961-x.

8 Hristo Djidjev and Shankar M. Venkatesan. Reduced constants for simple cycle graph separation. *Acta Inf.*, 34(3):231–243, 1997.

9 Alon Efrat, Alon Itai, and Matthew J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001. doi:10.1007/s00453-001-0016-8.

10 David Eppstein, Gary L. Miller, and Shang-Hua Teng. A deterministic linear time algorithm for geometric separators and its applications. *Fundam. Inform.*, 22(4):309–329, 1995.

11 J. Gao and L. Guibas. Geometric algorithms for sensor networks. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 370(1958):27–51, 2011. doi:10.1098/rsta.2011.0215.

12 John R. Gilbert and Robert E. Tarjan. The analysis of a nested dissection algorithm. *Numerische Mathematik*, 50(4):377–404, 1986. doi:10.1007/BF01396660.

13 Sariel Har-Peled and Kent Quanrud. Approximation algorithms for polynomial-expansion and low-density graphs. *SIAM J. Comput.*, 46(6):1712–1744, 2017. doi:10.1137/16M1079336.

14 M. L. Huson and A. Sen. Broadcast scheduling algorithms for radio networks. In *IEEE MILCOM '95*, volume 2, pages 647–651 vol.2, 1995. doi:10.1109/MILCOM.1995.483546.

15 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar voronoi diagrams for general distance functions and their algorithmic applications. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 2495–2504. SIAM, 2017. doi:10.1137/1.9781611974782.165.

16 Santosh Kumar, Ten H. Lai, and Anish Arora. Barrier coverage with wireless sensors. *Wireless Networks*, 13(6):817–834, 2007.

17 Richard J. Lipton, Donald J. Rose, and Robert E. Tarjan. Generalized nested dissection. *SIAM J. Numer. Anal.*, 16(2):346–358, 1979. doi:10.1137/0716027.

18 Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980.

19 Aleksander Mądry. Navigating central path with electrical flows: From flows to matchings, and back. In *Proceedings of the 54th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2013*, pages 253–262, 2013. doi:10.1109/FOCS.2013.35.

20 Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|}\cdot|E|)$ algorithm for finding maximum matching in general graphs. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science, FOCS 1980*, pages 17–27. IEEE Computer Society, 1980. doi:10.1109/SFCS.1980.12.

21 Gary L. Miller, Shang-Hua Teng, William P. Thurston, and Stephen A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *J. ACM*, 44(1):1–29, 1997.

22 Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. In *Proceedings of the 45th Symposium on Foundations of Computer Science, FOCS 2004*, pages 248–255. IEEE Computer Society, 2004. doi:10.1109/FOCS.2004.40.

23 Marcin Mucha and Piotr Sankowski. Maximum matchings in planar graphs via gaussian elimination. *Algorithmica*, 45(1):3–20, 2006. doi:10.1007/s00453-005-1187-5.

24 Michael O. Rabin and Vijay V. Vazirani. Maximum matchings in general graphs through randomization. *J. Algorithms*, 10(4):557–567, 1989. doi:10.1016/0196-6774(89)90005-9.

25 Warren D. Smith and Nicholas C. Wormald. Geometric separator theorems and applications. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science, FOCS*, pages 232–243, 1998.

26 A. Frank van der Stappen, Dan Halperin, and Mark H. Overmars. The complexity of the free space for a robot moving amidst fat obstacles. *Comput. Geom.*, 3:353–373, 1993. doi:10.1016/0925-7721(93)90007-S.

27 Vijay V. Vazirani. A simplification of the MV matching algorithm and its proof. *CoRR*, abs/1210.4594, 2012. URL: http://arxiv.org/abs/1210.4594.

28 Raphael Yuster and Uri Zwick. Maximum matching in graphs with an excluded minor. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007*, pages 108–117. SIAM, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283396.

29 F. Zhao and L. Guibas. *Wireless Sensor Networks: An Information Processing Approach*. Elsevier/Morgan-Kaufmann, 2004.

Unambiguous Separators for Tropical Tree Automata*

Thomas Colcombet
IRIF, CNRS, Université de Paris, Paris, France
https://www.irif.fr/~colcombe/
thomas.colcombet@irif.fr

Sylvain Lombardy
LaBRI, Institut Polytechnique de Bordeaux – Université de Bordeaux – CNRS, France
https://www.labri.fr/perso/slombard/
sylvain.lombardy@labri.fr

Abstract

In this paper we show that given a max-plus automaton (over trees, and with real weights) computing a function f and a min-plus automaton (similar) computing a function g such that $f \leqslant g$, there exists effectively an unambiguous tropical automaton computing h such that $f \leqslant h \leqslant g$.

This generalizes a result of Lombardy and Mairesse of 2006 stating that series which are both max-plus and min-plus rational are unambiguous. This generalization goes in two directions: trees are considered instead of words, and separation is established instead of characterization (separation implies characterization). The techniques in the two proofs are very different.

2012 ACM Subject Classification Theory of computation → Algebraic language theory; Theory of computation → Quantitative automata; Theory of computation → Tree languages

Keywords and phrases Tree automata, Tropical semiring, Separation, Unambiguity

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.32

Funding *Thomas Colcombet*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No.670624), and the DeLTA ANR project (ANR-16-CE40-0007).

Acknowledgements The authors are grateful to the anonymous referees whose remarks and comments have allowed to improve this paper.

1 Introduction

Tropical automata is a nickname for weighted automata (automata parameterized by a semiring as introduced by Schützenbgerger [17]) over a tropical semiring. This is a particularly simple model of finite state automata that describe functions rather than languages. It exists in two forms, max-plus and min-plus automata. Essentially, a tropical automaton $\mathcal{A}$ is a non-deterministic automaton for which each transition is labelled by a real weight (or an integer, or a natural number, depending on the variants). This weight is extended into a weight for a run: the sum of the weights of the transitions involved. A max-plus automaton computes the function $[\![\mathcal{A}]\!] \colon A^* \to \mathbb{R} \uplus \{\bot\}$ which to an input word associates the maximum weight of an accepting run over the input, or $\bot$ if there is no accepting runs. If it is a min-plus automaton, minimum is used instead of maximum.

The use of tropical automata arises naturally in different contexts: max-plus automata have been used for modeling scheduling constraints (see for instance [4]) or worst case behaviors (see for instance [3] for computing the asymptotic worst case execution time

* The authors are committed to making professional choices acknowledging the climate emergency. We submitted this work to STACS for its excellence and because its location induces for us a low carbon footprint.

of loops under the size-change abstraction); min-plus automata are used for optimisation questions (these are for instance used as a key tool in the decision of the star-height problem [6]). In all these situations, non-trivial decision procedures are used ([5, 12, 2]).

The starting point of this work is a result from 2006 of Lombardy and Mairesse:

▶ **Theorem 1** ([13, 14]). *A map $f\colon A^* \to \mathbb{R} \uplus \{\bot\}$ which is both definable by a min-plus and by a max-plus automaton is definable by an unambiguous tropical automaton.*

Recall that an automaton is unambiguous if there is at most one accepting run per input[1]. Unambiguous automata form a very particular class of tropical automata. Most of the problems which are open or undecidable for general tropical automata are easily decidable for unambiguous automata: equivalence with another tropical automaton [9], boundedness, existence of an equivalent deterministic automaton, description of the asymptotic behaviour [1].

It is noteworthy that the decision algorithm to decide whether there exists an equivalent automaton actually applies to unambiguous automata and that algorithms described for larger classes (finitely or polynomially ambiguous), consist indeed in deciding first whether the tropical automaton is equivalent to some unambiguous one [8, 7].

The above Theorem 1 belongs to a fascinating corpus of mathematical statements of the form 'if X belongs both to class C and to class D, then it belongs to class E', where E is structurally simpler than both C and D (often D is some form of dual of C). An archetypical example arises in descriptive set theory: Suslin's theorem states that

if a set is analytic and coanalytic, it is Borel.

Many other instances of this pattern exist. For instance in automata theory, if an infinite tree language is Büchi and its complement is Büchi, it is weak (Rabin's theorem [15]). This extends to cost-functions over infinite trees: if a cost-function over infinite trees is both B-Büchi and S-Büchi, it is quasi-weak ; over infinite words, it is even weak (Kuperberg and Vanden Boom [10, 11]). For languages of infinite words beyond regular, if a language is ωB and ωS definable, then it is ω-regular (Skrzypczak [18]). In language theory, a language which is both Σ_2 and Π_2 definable is definable in the two variables fragment (Thérien and Wilke [19]). Also, a language which is both the support and the complement of the support of a rational series over a field is regular [16]. This list continues on and on.

In many situations such statements arise in fact from a more general result of 'separation' (or of 'interpolation' in the logical terminology). For instance, Lusin's theorem is the separation version of Suslin's theorem: It states that

for $X \subseteq Y$ with X analytic and Y coanalytic, then $X \subseteq Z \subseteq Y$ for some Borel set Z.

Such separation results imply the characterization version. For instance, Suslin's result follows from Lusin's theorem: take $X = Y$ to be the set which is both analytic and coanalytic. Then $X \subseteq Z \subseteq Y = X$ for Z Borel; hence X is Borel. This relationship is general. The results of Rabin, Vanden Boom and Kuperberg, and Skrzypczak, for instance, exist in a 'separation variant'.

[1] Note that when a tropical automaton is unambiguous, it makes no difference whether it is a max-plus or a min-plus automaton: It computes the same function.

Contribution

The natural question that we answer in this work is thus:

Does there exist a separation version of Theorem 1 ?

In this paper, we provide a positive answer to this question. It takes the following form:

▶ **Theorem 2** (separation for tropical tree automata). *Given a max-plus automaton* $\mathcal{A}_{\max}$ *and a min-plus automaton* $\mathcal{A}_{\min}$ *such that*[2]

$$[\![\mathcal{A}_{\max}]\!] \leqslant [\![\mathcal{A}_{\min}]\!] ,$$

there exists effectively an unambiguous tropical automaton $\mathcal{A}_{\text{sep}}$ *such that*

$$[\![\mathcal{A}_{\max}]\!] \leqslant [\![\mathcal{A}_{\text{sep}}]\!] \leqslant [\![\mathcal{A}_{\min}]\!] .$$

Let us stress that the above theorem is established in the context of tropical automata over trees. Theorem 1 is now a corollary. Indeed, (a) tropical word automata are a particular case of tree automata over a ranked alphabet made of unary symbols only, plus a constant, and (b) assuming that f is both accepted by a min-plus and by a max-plus automaton, then by Theorem 2, there exists a function h accepted by an unambiguous tropical automaton such that $f \leqslant h \leqslant f$. Thus $f = h$ is accepted by an unambiguous tropical automaton.

Note that, though the result is a generalization, the proof of Theorem 2 is very different from the original one of Theorem 1.

Let us finally emphasize that particular care has been taken in order to obtain the result for real weights. Indeed, in the integer case (and as a consequence in the rational case), simpler techniques can be used that involve keeping in the finitely many states of the result automaton some explicit differences of partial weights up to a certain bound. Such a technique (as far as we know) cannot be used in the real case. Our results are effective for real weights as far as there exist an effective representation of the reals in the additive group generated by weights of automata $\mathcal{A}_{\max}$ and $\mathcal{A}_{\min}$, as well as algorithms that compute the addition, the subtraction, and the comparison on these representations.

Other Related Work

The class of unambiguous tropical automata form an interesting subclass of tropical automata. In particular, equivalence is decidable, while the problem for max-plus or min-plus automata is undecidable [9]. Given a tropical automaton, deciding unambiguity is an open problem. It has been solved when the input automaton is finitely ambiguous in [8], and when it is polynomially ambiguous in [7].

The approach used in this paper is completely different from the original result of [14]. Crossing reachable and productive states refers to technics that have been used since Hashiguchi's papers on limitedness of tropical automata [5], but the basement of our proof is the original pumping Lemma 11.

Structure of the Paper

This paper is organized as follows. In Section 2, we recall the standard definitions concerning trees, automata over trees, and tropical automata. In Section 3, we establish our main theorem of separation, Theorem 2. Section 4 concludes.

[2] In this statement, we assume that $\bot$ is incomparable with other elements, and thus $[\![\mathcal{A}_{\max}]\!]$ and $[\![\mathcal{A}_{\min}]\!]$ are equal to $\bot$ on the same words: they have same support.

2 Definitions

We review in this section classical notions concerning terms, automata, and tropical automata.

2.1 Terms and Contexts

A *ranked alphabet* is a set A, the elements of which are called *letters*, together with a map rank from A to $\mathbb{N}$. For $n \in \mathbb{N}$, let $\mathrm{Terms}(n)$ be the set of *terms of arity* n over the alphabet $A \cup \{1, \dots, n\}$ in which $1, \dots, n$ are seen as special letters of rank 0 that are used exactly once in each term. We call simply *terms* the terms of arity 0, and the set of terms is simply denoted Terms. We call *context* the terms of arity 1, and the set of contexts is simply denoted Contexts. Note that each letter a of rank n can naturally be seen as a term of arity n consisting solely of a root labelled a and children $1, \dots, n$. The *nodes* of a term of arity n, $\mathrm{Nodes}(t)$ is the set of positions of the letters in the term. The *root node* is denoted root. A node labelled i for $i = 1 \dots n$ is called the *ith-hole*. The nodes that are not holes are called *inner nodes*. Given a node $x \in \mathrm{Nodes}(t)$, $t(x)$ denotes the letter it carries. Given a letter of rank n and terms $t_0, \dots, t_{n-1}$, we denote by $a(t_0, \dots, t_{n-1})$ the term that has a as root, and as children from left to right $t_0, \dots, t_{n-1}$. The *height of a term* s, denoted $\mathrm{height}(s)$, is the longest length of a branch, for the standard meaning of a branch. The *size of a term* s, denoted $\mathrm{size}(t)$, is the number of nodes it has. Finally, given c a context and t a term (resp. t another context), we denote $c \circ t$ the term (resp. the context) obtained by plugging the root of t in the hole of c.

2.2 Automata

A *non-deterministic (tree) automaton* (or simply an automaton) has a finite set of *states* Q, an input ranked alphabet A, a *set of final states* F, and a *transition relation* Δ that consists of tuples of the form $(p_0, \dots, p_{n-1}, a, q)$ in which $a \in A$ is a letter of rank n, and $p_0, \dots, p_{k-1}, q$ are states from Q.

A *run of the automaton* over a term t of arity n is a map ρ from $\mathrm{Nodes}(t)$ to Q such that for all inner nodes $x \in \mathrm{Nodes}(t)$ of children $x_0, \dots, x_{n-1}$, $(\rho(x_0), \dots, \rho(x_{n-1}), t(x), \rho(x)) \in \Delta$. We shall write $\tilde{\rho}(x)$ for this transition. An *accepting run* is a run of the automaton such that $\rho(\mathrm{root}) \in F$. Given a term t, t is *accepted by the automaton* if there exists an accepting run of the automaton over t. The set of terms that are accepted is the *language accepted by the automaton*. We slightly refine the terminology for easier use. Over a term, a *run to state* q is a run that assumes state q at the root. Over a context, a *run from state* p *to state* q signifies that the state assumed in the hole is p, and the one assumed at the root is q. An *accepting run from* p is a run from p to q for a final state q.

An automaton is *unambiguous* if for all input terms t, there exists at most one accepting run over it. Said differently, for all input terms t, either there are no accepting runs over it, and the term is not accepted, or there is exactly one accepting run, and the term is accepted.

An *automaton with weights*[3] $\mathcal{A}$ is a non-deterministic automaton together with a real *weight* for all transitions and all final states, i.e. a map weight from $\Delta \uplus F$ to $\mathbb{R}$. Given a run ρ of the automaton, the *weight of the run* $\mathrm{weight}(\rho)$ is the sum of the weights of $\tilde{\rho}(x)$ for x

[3] This is not a weighted automaton, which is parametrized by a semiring and not a monoid. This definition serves here just for holding the structure of our tropical automata irrespective of whether these are min-plus or max-plus.

ranging over the inner nodes of t. Given an accepting run ρ of the automaton, the *weight of the accepting run* $\mathrm{weight}^{\mathrm{acc}}(\rho)$ is the sum of the weight of the run and $\mathrm{weight}(\rho(\mathrm{root}))$.

Tropical automata refer in this work to one of two forms of automata: min-plus automata and max-plus automata defined as follows. A *min-plus automaton* $\mathcal{A}$ is an automaton with weights that computes a function:

$$[\![\mathcal{A}]\!]_{\min} : \mathrm{Terms} \longrightarrow \mathbb{R} \uplus \{\bot\}$$
$$t \longmapsto \begin{cases} \bot & \text{if there are no accepting runs of } \mathcal{A} \text{ over } t, \\ \min\{\mathrm{weight}^{\mathrm{acc}}(\rho) \mid \rho \text{ accepting run of } \mathcal{A} \text{ over } t\} & \text{otherwise,} \end{cases}$$

in which $\bot$ is a symbol that we understand as 'undefined' (it appears classically as an absorbing element for $+$ which is larger than all $x \in \mathbb{R}$, i.e., the zero of the tropical semiring). A *max-plus automaton* is defined in an identical manner, but the semantics $[\![\mathcal{A}]\!]_{\max}$ is now defined using max instead of min. Since it is always clear from the context, we denote simply by $[\![\mathcal{A}]\!]$ either $[\![\mathcal{A}]\!]_{\min}$ or $[\![\mathcal{A}]\!]_{\max}$ depending on whether $\mathcal{A}$ has been declared as a min-plus or as a max-plus automaton.

An *unambiguous tropical automaton* $\mathcal{A}$ is a tropical automaton that has an unambiguous underlying automaton. Note that in this case, $[\![\mathcal{A}]\!]_{\max} = [\![\mathcal{A}]\!]_{\min}$, and hence we call it simply tropical automaton and do not have to specify whether it is min-plus or max-plus.

3 Separating Tropical Automata

3.1 Statement and Structure of the Proof

The goal of this section is to prove our main theorem:

▶ **Theorem 2** (separation for tropical tree automata). *Given a max-plus automaton $\mathcal{A}_{\max}$ and a min-plus automaton $\mathcal{A}_{\min}$ such that*[4]

$$[\![\mathcal{A}_{\max}]\!] \leqslant [\![\mathcal{A}_{\min}]\!] \ ,$$

there exists effectively an unambiguous tropical automaton $\mathcal{A}_{\mathrm{sep}}$ such that

$$[\![\mathcal{A}_{\max}]\!] \leqslant [\![\mathcal{A}_{\mathrm{sep}}]\!] \leqslant [\![\mathcal{A}_{\min}]\!] \ .$$

From now on, we fix the ranked alphabet A, a max-plus automaton $\mathcal{A}_{\max}$ and a min-plus automaton $\mathcal{A}_{\min}$:

$$\mathcal{A}_{\max} = (Q_{\max}, A, F_{\max}, \Delta_{\max}, \mathrm{weight}_{\max}) \ \text{ and } \ \mathcal{A}_{\min} = (Q_{\min}, A, F_{\min}, \Delta_{\min}, \mathrm{weight}_{\min})$$

such that

$$[\![\mathcal{A}_{\max}]\!] \leqslant [\![\mathcal{A}_{\min}]\!] \ .$$

It will be convenient in what follows to consider a single automaton with weights constructed as the disjoint union of $\mathcal{A}_{\max}$ and $\mathcal{A}_{\min}$ (of course, it should be neither seen as a min-plus

[4] In this statement, we assume that $\bot$ is incomparable with other elements, and thus $[\![\mathcal{A}_{\max}]\!]$ and $[\![\mathcal{A}_{\min}]\!]$ are equal to $\bot$ on the same words: they have same support.

automaton nor as a max-plus automaton). Formally, we assume without loss of generality that $Q_{\max}$ and $Q_{\min}$ are disjoint, and we set this automaton $\mathcal{A} = (Q, A, F, \Delta, \text{weight})$, with

$$Q = Q_{\min} \cup Q_{\max}, \qquad F = F_{\max} \cup F_{\min}, \qquad \Delta = \Delta_{\max} \cup \Delta_{\min},$$
$$\text{and} \quad \text{weight}(v) = \begin{cases} \text{weight}_{\max}(v) & \text{for } v \in \Delta_{\max} \uplus F_{\max} \\ \text{weight}_{\min}(v) & \text{otherwise.} \end{cases}$$

The rest of this section is devoted to the proof of Theorem 2, and is organized as follows. In Section 3.2, we use some classical automata constructions for accessing in an unambiguous manner the reachable and productive states (Lemma 3). The combinatorial core of the proof is contained in Section 3.3 in which we study how the values of the automata may evolve in a context (Lemma 9), and use it for showing how terms can be substituted for smaller ones while preserving separability (Corollary 12). We finally provide the construction of the automaton $\mathcal{A}_{\text{sep}}$ in Section 3.4, and establish its correctness (Lemma 15). This concludes the proof of Theorem 2.

3.2 Reachable and Productive States

An ingredient which is necessary in the proof is that the automaton we construct is always 'aware' of what are the states that may lead to an accepting run to the root. This section is concerned with this aspect, and involves only completely standard techniques for tree automata.

Given a term t, set $\text{Reach}(t) \subseteq Q$ to be the set of states p such that there is a run over t to p. We call such states *reachable in* t. Given a context c, set $\text{Prod}(c) \subseteq Q$ to be the set of states p such that there is an accepting run from p. We call such states *productive in* c. We finally set

$$\text{Reachable} = \{\text{Reach}(t) \mid t \in \text{Terms}\} \quad \text{and} \quad \text{Productive} = \{\text{Prod}(c) \mid c \in \text{Contexts}\}.$$

We describe the construction of an automaton $\mathcal{A}_{\text{pro}} = (Q_{\text{pro}}, A, F_{\text{pro}}, \Delta_{\text{pro}})$ that computes the productive states at each node of a term. The states are $Q_{\text{pro}} = \text{Reachable} \times \text{Productive}$. The final states $F_{\text{pro}} = \text{Reachable} \times \{F\}$, and for all letters a of rank n, the automaton has a transition of the form

$$((R_0, P_0), \dots, (R_{n-1}, P_{n-1}), a, (R, P)) \in \Delta_{\text{pro}}$$

whenever

- $R = \{r \in Q \mid (r_0, \dots, r_{n-1}, a, r) \in \Delta,\ r_j \in R_j \text{ for all } j\}$, and
- $P_i = \{r_i \in Q \mid (r_0, \dots, r_{n-1}, a, p) \in \Delta,\ r_j \in R_j \text{ for } j \neq i,\ p \in P\}$ for all $i = 0 \dots n-1$.

In the above definition, the constraint on R induces the computation in a bottom-up deterministic way of the set of states that are reachable from the term below. The constraint on P_i computes similarly in a top-down deterministic way the set of states that are productive in the context above. We do not prove the correctness of this construction further. The important aspects of this construction are summarized in the following lemma.

▶ **Lemma 3.** *For all* $P \in \text{Productive}$ *and all terms* t*, there exists one and one only run of* $\mathcal{A}_{\text{pro}}$ *over* t *to a state of the form* (R, P) *for some* $R \in \text{Reachable}$*. And furthermore,* $R = \text{Reach}(t)$*.*

For all $R \in \text{Reachable}$ *and all contexts* c*, there exists one and only one accepting run of* $\mathcal{A}_{\text{pro}}$ *over* c *from a state of the form* (R, P) *for some* $P \in \text{Productive}$*. And furthermore,* $P = \text{Prod}(c)$*.*

3.3 The Central Pumping Lemma

In this section, we establish the key Corollary 12. The central concept here is to understand what it does for the value computed by $\mathcal{A}_{\max}$ and by $\mathcal{A}_{\min}$ to substitute a subtree for another subtree. And more precisely, we devise sufficient conditions such that, after performing the substitution, the values of the two automata gets closer one to the other, up to some shifting. This property is expressed in Lemma 5.

The key definition involved is the one of refinement with shift as defined now.

▶ **Definition 4.** *Given two terms s, t, some set $P \subseteq Q$, and some real number x, then t* refines *s for P with shift x if*

- $\mathrm{Reach}(s) = \mathrm{Reach}(t)$,
- *for all runs ρ of $\mathcal{A}_{\max}$ over s to a state $p \in P$, there is a run ρ' over t to state p such that*

$$\mathrm{weight}(\rho) \leqslant \mathrm{weight}(\rho') + x$$

and

- *for all runs τ of $\mathcal{A}_{\min}$ over s to a state $q \in P$, there is a run τ' over t to state q such that*

$$\mathrm{weight}(\tau') + x \leqslant \mathrm{weight}(\tau)$$

The justification of this definition is given by the following lemma. It shows how substituting s for t in a context when t refines s with some shift is done while 'staying in the separation interval'.

▶ **Lemma 5.** *Let c be a context, and s, t be terms such that t refines s for $\mathrm{Prod}(c)$ with shift x, then*

$$[\![\mathcal{A}_{\max}]\!](c \circ s) \leqslant [\![\mathcal{A}_{\max}]\!](c \circ t) + x \leqslant [\![\mathcal{A}_{\min}]\!](c \circ t) + x \leqslant [\![\mathcal{A}_{\min}]\!](c \circ s) \ .$$

Proof. Let ρ be an accepting run of $\mathcal{A}_{\max}$ over $c \circ s$. It can be decomposed as an accepting run ρ_c over c from some state p and a run ρ_s over s to state p. The run ρ_c is a witness that $p \in \mathrm{Prod}(c) \cap Q_{\max}$. Hence, since t refines s for $\mathrm{Prod}(c)$ with shift x, there exists a run ρ_t over t to state p such that $\mathrm{weight}(\rho_s) \leqslant \mathrm{weight}(\rho_t) + x$. By gluing ρ_t with ρ_c, we obtain a new accepting run ρ' of $\mathcal{A}_{\max}$ over $c \circ t$, furthermore,

$$\begin{aligned}\mathrm{weight}^{\mathrm{acc}}(\rho) &= \mathrm{weight}^{\mathrm{acc}}(\rho_c) + \mathrm{weight}(\rho_s) \\ &\leqslant \mathrm{weight}^{\mathrm{acc}}(\rho_c) + \mathrm{weight}(\rho_t) + x = \mathrm{weight}^{\mathrm{acc}}(\rho') + x \ .\end{aligned}$$

Since for all ρ there exists such a ρ', we obtain

$$[\![\mathcal{A}_{\max}]\!](c \circ s) \leqslant [\![\mathcal{A}_{\max}]\!](c \circ t) + x \ .$$

The middle inequality simply comes from the key assumption $[\![\mathcal{A}_{\max}]\!] \leqslant [\![\mathcal{A}_{\min}]\!]$ in Theorem 2.

The third inequality is established as the first one (it is symmetric). ◀

The two following facts are straightforward to verify.

▶ **Fact 6** (reflexivity of refinement with shift)**.** *For all terms s, and all $P \subseteq Q$, s refines s for P with shift 0.*

▶ **Fact 7** (transitivity of refinement with shift)**.** *If t refines s for P with shift x, and u refines t for P with shift y, then u refines s for P with shift $x + y$.*

The next lemma is also purely mechanical.

▶ **Lemma 8** (refinement with shift is a congruence). *Let* $((P_0, R_0), \dots, (P_{n-1}, R_{n-1}), a, (P, R))$ *be in* Δ_{pro}*, and for all* $i = 0 \dots n-1$*, let* t_i, s_i *be terms such that*

- $\text{Reach}(t_i) = R_i$*, and*
- t_i *refines* s_i *for* P_i *with shift* x_i*,*

then $a(t_0, \dots, t_{n-1})$ *refines* $a(s_0, \dots, s_{n-1})$ *for* P *with shift* $x_0 + \dots + x_{n-1}$*.*

Proof. Let ρ be a run of $\mathcal{A}_{\max}$ over $a(t_0, \dots, t_{n-1})$ to state $p \in P$. The run ρ can be decomposed into a transition $(p_0, \dots, p_{n-1}, a, p)$ of weight x at the root, and a run ρ_i of $\mathcal{A}_{\max}$ over t_i to p_i for all $i = 0 \dots n-1$. For all $i = 0 \dots n-1$, $p_i \in \text{Reach}(t_i) = R_i$. Since furthermore $p \in P$ and $((P_0, R_0), \dots, (P_{n-1}, R_{n-1}), a, (P, R)) \in \Delta_{\text{pro}}$, we obtain that $p_i \in P_i$ for all $i = 0 \dots n-1$ (second item of the definition of Δ_{pro}). Thus, since t_i refines s_i for P_i with shift x_i, there exists a run ρ'_i of $\mathcal{A}_{\max}$ over t_i to p_i such that $\text{weight}(\rho_i) \leqslant \text{weight}(\rho'_i) + x_i$. We can combine all these runs ρ'_i together with the transition $(p_0, \dots, p_{n-1}, a, p)$ and obtain a new run ρ' of $\mathcal{A}_{\max}$ over $a(t_0, \dots, t_{n-1})$ to p such that

$$\begin{aligned}\text{weight}(\rho) &= \text{weight}(\rho_0) + \dots + \text{weight}(\rho_{n-1}) + x \\ &\leqslant \text{weight}(\rho'_0) + x_0 + \dots + \text{weight}(\rho'_{n-1}) + x_{n-1} + x \\ &= \text{weight}(\rho') + x_0 + \dots + x_{n-1} \ .\end{aligned}$$

This shows half of the fact that $a(t_0, \dots, t_{n-1})$ refines $a(s_0, \dots, s_{n-1})$ for P with shift $x_0 + \dots + x_{n-1}$. The other half is symmetric. ◀

We aim now at proving Corollary 12 which states that all sufficiently large term is 'shift refined' by another one of smaller size. Beforehand, we need a pumping argument to establish:

▶ **Lemma 9.** *Let* $P \in \text{Productive}$*,* $R \in \text{Reachable}$ *and* m *be a context, then there exists a real number* x *such that*

- *for every* p *in* $Q_{\max} \cap P \cap R$*, for all runs* ρ *of* $\mathcal{A}_{\max}$ *over* m *from* p *to* p*,* $\text{weight}(\rho) \leqslant x$*, and*
- *for every* q *in* $Q_{\min} \cap P \cap R$*, for all runs* τ *of* $\mathcal{A}_{\min}$ *over* m *from* q *to* q*,* $x \leqslant \text{weight}(\tau)$*.*

Proof. Let t be a term such that $\text{Reach}(t) = R$, and c be a context such that $\text{Prod}(c) = P$.

▷ Claim 10. We claim first that for all runs ρ of $\mathcal{A}_{\max}$ over m from p to p with $p \in P \cap R$ and all runs τ of $\mathcal{A}_{\min}$ over m from q to q with $q \in P \cap R$, $\text{weight}(\rho) \leqslant \text{weight}(\tau)$.
Otherwise, there would exist some runs ρ, τ as above such that $\text{weight}(\rho) > \text{weight}(\tau)$. I.e.

$$\text{weight}(\tau) - \text{weight}(\rho) < 0 \ . \qquad (\star)$$

Consider now for all $n > 0$ the term:

$$u_n = c \circ \overbrace{m \circ \dots \circ m}^{n\text{-times}} \circ t \ .$$

Let ρ' be some accepting run over c from p (this is possible since $p \in P = \text{Prod}(c)$). Let τ' be some accepting run over c from q (this is possible since $q \in P = \text{Prod}(c)$). Let ρ'' be some run over t to p (this is possible since $p \in R = \text{Reach}(t)$). Let τ'' be some run over t to q (this is possible since $q \in R = \text{Reach}(t)$).

By concatenating ρ', n-times ρ, and ρ'', we obtain an accepting run ρ_n over u_n of weight $\mathrm{weight}^{\mathrm{acc}}(\rho_n) = \mathrm{weight}^{\mathrm{acc}}(\rho') + n\,\mathrm{weight}(\rho) + \mathrm{weight}(\rho'')$. Similarly, by concatenating τ', n-times τ, and τ'', we obtain an accepting run τ_n over u_n of weight $\mathrm{weight}^{\mathrm{acc}}(\tau_n) = \mathrm{weight}^{\mathrm{acc}}(\tau') + n\,\mathrm{weight}(\tau) + \mathrm{weight}(\tau'')$.

Furthermore, since $[\![\mathcal{A}_{\max}]\!](u_n) \leqslant [\![\mathcal{A}_{\min}]\!](u_n)$, $\mathrm{weight}^{\mathrm{acc}}(\rho_n) \leqslant \mathrm{weight}^{\mathrm{acc}}(\tau_n)$. We obtain

$$\begin{aligned}0 &\leqslant \mathrm{weight}^{\mathrm{acc}}(\tau_n) - \mathrm{weight}^{\mathrm{acc}}(\rho_n)\\ &= \mathrm{weight}^{\mathrm{acc}}(\tau') + n\,\mathrm{weight}(\tau) + \mathrm{weight}(\tau'') - \mathrm{weight}^{\mathrm{acc}}(\rho') - n\,\mathrm{weight}(\rho) - \mathrm{weight}(\rho'')\\ &= \mathrm{weight}^{\mathrm{acc}}(\tau') + \mathrm{weight}(\tau'') - \mathrm{weight}^{\mathrm{acc}}(\rho') - \mathrm{weight}(\rho'') + n(\mathrm{weight}(\tau) - \mathrm{weight}(\rho)).\end{aligned}$$

However, using $(\star)$, this last quantity tends to $-\infty$ when n tends to ∞. It contradicts its non-negativeness. The claim is established.

We can now establish the lemma. Let Y be the set of weights $\mathrm{weight}(\rho)$ for ρ ranging over the runs of $\mathcal{A}_{\max}$ over m from p to p with $p \in P \cap R$. Similarly, let Z be the set of weights $\mathrm{weight}(\tau)$ for τ ranging over the runs of $\mathcal{A}_{\min}$ over m from q to q with $q \in P \cap R$. The above claim states that for all $y \in Y$ and all $z \in Z$, $y \leqslant z$. This implies the existence of some real number x such that for all $y \in Y$, $y \leqslant x$, and for all $z \in Z$, $x \leqslant z$ (note that proving it requires to treat the case of Y and/or Z being empty, and thus requires a case distinction). This is exactly the statement of the lemma. ◀

Notice that a fixed context m admits only a finite number of runs; hence, the weights of paths involved in Lemma 9 can be enumerated and the value x be effectively computed.

▶ **Lemma 11.** *There exists a computable $k \in \mathbb{N}$ such that for all $P_0 \in$ Reachable and all terms s of height more than k, there exists effectively a term t such that t refines s for P_0 with some shift and* $\mathrm{size}(t) < \mathrm{size}(s)$.

Proof. Let k be $(4|Q|)^{|Q|}$. Let us fix a context d such that $\mathrm{Prod}(d) = P_0$.

Consider now a term s of height larger than k and some $P_0 \in$ Reachable. We aim at removing some piece of this term while achieving the conclusions of the lemma.

For all states $p \in P_0$, set ρ_p to be an optimal run of $\mathcal{A}$ over s to p, i.e.,

- if $p \in Q_{\max}$, then for all runs τ of $\mathcal{A}_{\max}$ over s to p, $\mathrm{weight}(\tau) \leqslant \mathrm{weight}(\rho_p)$, and
- if $p \in Q_{\min}$, then for all runs τ of $\mathcal{A}_{\min}$ over s to p, $\mathrm{weight}(\rho_p) \leqslant \mathrm{weight}(\tau)$.

Since the longest branch of s has length greater than $2^{|Q|}2^{|Q|}|Q|^{|Q|}$, we can apply the pigeonhole principle to the various ways to split this branch in two, and get a factorisation of s into

$$s = c \circ m \circ s' \ ,$$

in which c is a context, m is a non-empty context, and s' is a term such that

- $\mathrm{Reach}(s') = \mathrm{Reach}(m \circ s')$; let R be this set;
- $\mathrm{Prod}(d \circ c) = \mathrm{Prod}(d \circ c \circ m)$; let P be this set;
- for all $p \in P_0$, there exists a state $q_p \in Q$ such that ρ_p is decomposed into a run τ_p over s' to q_p, a run τ'_p over m from q_p to q_p, and a run τ''_p over c from q_p to p.

Let us define now our term t as:

$$t = c \circ s' \ .$$

Since $s = c \circ m \circ s'$, our new term t is nothing but s in which the non-empty part corresponding to m has been removed. Hence $\mathrm{size}(t) < \mathrm{size}(s)$.

We shall prove now that t refines s for P_0 with shift x where x is obtained by applying Lemma 9 to P, R and m.

Let ρ be a run of $\mathcal{A}_{\max}$ over s to state p for some $p \in P_0$. We know that the run ρ_p as defined above is such that $\mathrm{weight}(\rho) \leqslant \mathrm{weight}(\rho_p)$. Finally, let ρ' be the run over $t = c \circ s'$ to p obtained by gluing τ_p and τ''_p together. We have:

$$\begin{aligned}
\mathrm{weight}(\rho) &\leqslant \mathrm{weight}(\rho_p) && \text{(by optimality of } \rho_p\text{)}\\
&\leqslant \mathrm{weight}(\tau_p) + \mathrm{weight}(\tau'_p) + \mathrm{weight}(\tau''_p) && \text{(decomposion of } \rho_p\text{)}\\
&\leqslant \mathrm{weight}(\tau_p) + x + \mathrm{weight}(\tau''_p) && \text{(by choice of } x \text{ and Lemma 9)}\\
&\leqslant \mathrm{weight}(\rho') + x\ . && \text{(definition of } \rho'\text{)}
\end{aligned}$$

Hence, we have proved the first half of the definition of 't refines s for P_0 with shift x'. The second half is symmetric. Overall, we conclude that t refines s for P_0 with shift x. ◀

Using iteratively the above Lemma 11, as long as the height of the term is larger than k, together with Fact 6 and 7, we obtain the following corollary.

▶ **Corollary 12.** *There exists a computable $k \in \mathbb{N}$ such that for all $P \in$ Reachable and all terms s there exists effectively a term t of height at most k which refines s for P with some shift.*

3.4 The Construction

We are now ready to construct our separating automaton $\mathcal{A}_{\mathrm{sep}}$. It is defined as follows:

$$\mathcal{A}_{\mathrm{sep}} = (Q_{\mathrm{sep}}, A, F_{\mathrm{sep}}, \Delta_{\mathrm{sep}}, \mathrm{weight}_{\mathrm{sep}})\ ,$$

in which the set of states is

$$Q_{\mathrm{sep}} = \{(R, P, t) \mid R \in \text{Reachable},\ P \in \text{Productive},\ t \in \text{Terms},\ \mathrm{Reach}(t) = R,\ \mathrm{height}(t) \leqslant k\}\ ,$$

(where k is the constant from Corollary 12), the final states, together with their weight, are

$$F_{\mathrm{sep}} = \{(R, P, t) \in Q_{\mathrm{sep}} \mid P = F,\ R \cap F \neq \emptyset\} \quad \text{with} \quad \mathrm{weight}_{\mathrm{sep}}(R, F, t) = [\![\mathcal{A}_{\max}]\!](t)\ ,$$

and the transition relation and the weights are defined as follows. For a letter a of rank n, there is a transition of the form

$$\delta = ((R_0, P_0, t_0), \ldots, (R_{n-1}, P_{n-1}, t_{n-1}), a, (R, P, t)) \in \Delta_{\mathrm{sep}} \quad \text{with} \quad \mathrm{weight}_{\mathrm{sep}}(\delta) = x\ ,$$

whenever

- $((R_0, P_0), \ldots, (R_{n-1}, P_{n-1}), a, (R, P))$ is a transition of Δ_{pro}.
- $(t, x) = \mathrm{sr}_P(a(t_0, \ldots, t_{n-1}))$, where sr is a map of the following form:

$$\begin{aligned}
\mathrm{sr}_P \colon \text{Terms} &\longrightarrow \text{Terms} \times \mathbb{R}\\
s &\longmapsto (t, x) \qquad \text{such that } t \text{ refines } s \text{ for } P \text{ with shift } x.
\end{aligned}$$

(Such a map exists thanks to Corollary 12.)

Notice first that R is a redondant information in the state (R, P, t), since $R = \mathrm{Reach}(t)$. The set P in the state ensures that the weights which are considered are really contributing to the run. If the constraint $\mathrm{height}(t) \leqslant k$ was removed from the definition of Q_{sep} and sr_P was defined as $sr_P(s) = (s, 0)$, the automaton $\mathcal{A}_{\mathrm{sep}}$ would be an infinite unambiguous automaton equivalent to $\mathcal{A}_{\max}$. Thanks to Lemma 11, one can bound the height of t in order to obtain an automaton that realizes a function which is larger than $[\![\mathcal{A}_{\max}]\!]$ but smaller than $[\![\mathcal{A}_{\min}]\!]$. The automaton is finite: the number of states is bounded by $2^{|Q|}a^{c^k}$, where a is the size of the alphabet, c is the maximal rank of letters, and k is the constant of Lemma 11, which is smaller than $(4|Q|)^{|Q|}$. This bound is obviously crude. In a practical implementation, an improvement can easily be made. It is not necessary to use all terms t of height up to k in Q_{sep}: it is sufficient to keep minimal ones for the shift refine relation for each $P \in \mathrm{Productive}$.

Let us first note:

▶ **Lemma 13.** *For all $P \in \mathrm{Productive}$ and all terms s, there exists exactly one run of $\mathcal{A}_{\mathrm{sep}}$ over s to a state of the form (R, P, t).*

Proof. Indeed, we have seen in Lemma 3 that $\mathcal{A}_{\mathrm{sep}}$ is unambiguous on its first two components. Then the third component is computed in a bottom-up deterministic manner. Furthermore, it is easy to show by induction that on every input term there is an accepting run. ◀

▶ **Lemma 14.** *Let ρ be a run of $\mathcal{A}_{\mathrm{sep}}$ over s to (R, P, t), then t refines s for P with shift $\mathrm{weight}_{\mathrm{sep}}(\rho)$.*

Proof. The proof is by induction on $\mathrm{height}(s)$. Assume s of the form $a(s_0, \ldots, s_{n-1})$. Let ρ be the run of $\mathcal{A}_{\mathrm{sep}}$ over s to (R, P, t), let $\delta = ((R_0, P_0, t_0), \ldots, (R_1, P_1, t_1), a, (R, P, t))$ be the transition assumed by ρ at the root. Let ρ_i be the run ρ restricted to the subterm s_i. By induction hypothesis, t_i refines s_i for P_i with shift $\mathrm{weight}_{\mathrm{sep}}(\rho_i)$. By Fact 7, $a(t_0, \ldots, t_{n-1})$ refines s for P with shift $\mathrm{weight}_{\mathrm{sep}}(\rho_0) + \cdots + \mathrm{weight}_{\mathrm{sep}}(\rho_{n-1})$. By definition of $\mathrm{weight}_{\mathrm{sep}}$, t refines $a(t_0, \ldots, t_{n-1})$ with shift $\mathrm{weight}_{\mathrm{sep}}(\delta)$. By Fact 7, we obtain that t refines s with shift $\mathrm{weight}_{\mathrm{sep}}(\rho_0) + \cdots + \mathrm{weight}_{\mathrm{sep}}(\rho_{n-1}) + \mathrm{weight}_{\mathrm{sep}}(\delta) = \mathrm{weight}_{\mathrm{sep}}(\rho)$. ◀

We can now provide the concluding lemma of the proof of Theorem 2.

▶ **Lemma 15.** $[\![\mathcal{A}_{\max}]\!] \leqslant [\![\mathcal{A}_{\mathrm{sep}}]\!] \leqslant [\![\mathcal{A}_{\min}]\!]$.

Proof. Let s be a term. By Lemma 13, there exists one and exactly one run ρ_{sep} of $\mathcal{A}_{\mathrm{sep}}$ over s to a state of the form (R, F, t). By Lemma 14, t refines s for F with shift $\mathrm{weight}_{\mathrm{sep}}(\rho_{\mathrm{sep}})$. Note that in this case $R = \mathrm{Reach}(s) = \mathrm{Reach}(t)$.

Two cases can occur. If (R, F, t) is not final. In this case, there is no accepting run of $\mathcal{A}_{\mathrm{sep}}$ over s, and $[\![\mathcal{A}_{\mathrm{sep}}]\!](s) = \bot$. However, $(R, F, t) \notin F_{\mathrm{sep}}$ means $\mathrm{Reach}(t) \cap F = \emptyset$, hence $\mathrm{Reach}(s) \cap F = \emptyset$. Thus $[\![\mathcal{A}_{\max}]\!](s) = [\![\mathcal{A}_{\min}]\!](s) = \bot$. We indeed have $[\![\mathcal{A}_{\max}]\!](s) \leqslant [\![\mathcal{A}_{\mathrm{sep}}]\!](s) \leqslant [\![\mathcal{A}_{\min}]\!](s)$.

Otherwise, (R, F, t) is final, i.e. $R \cap F \neq \emptyset$. Assume for instance that there is some $R \cap F \cap Q_{\max} \neq \emptyset$ (it would be the same for $Q_{\min}$). This means that $[\![\mathcal{A}_{\max}]\!](s) \neq \bot$. Since $[\![\mathcal{A}_{\min}]\!] \geqslant [\![\mathcal{A}_{\max}]\!]$, this implies also $[\![\mathcal{A}_{\min}]\!](s) \neq \bot$.

Let now ρ be an accepting run of $\mathcal{A}_{\max}$ over s of maximal value, and let p be its root state. Since t refines s for F with shift $\mathrm{weight}_{\mathrm{sep}}(\rho_{\mathrm{sep}})$, and $p \in F$, there exists a run ρ'

over t to state p such that $\mathrm{weight}(\rho) \leqslant \mathrm{weight}(\rho') + \mathrm{weight}_{\mathrm{sep}}(\rho_{\mathrm{sep}})$. Hence,

$$\begin{aligned}
[\![\mathcal{A}_{\max}]\!](s) &= \mathrm{weight}^{\mathrm{acc}}(\rho) \\
&= \mathrm{weight}(\rho) + \mathrm{weight}_{\max}(p) \\
&\leqslant \mathrm{weight}(\rho') + \mathrm{weight}_{\max}(p) + \mathrm{weight}(\rho_{\mathrm{sep}}) \\
&\leqslant [\![\mathcal{A}_{\max}]\!](t) + \mathrm{weight}(\rho_{\mathrm{sep}}) \\
&= [\![\mathcal{A}_{\mathrm{sep}}]\!](s)
\end{aligned}$$

In a symmetrical way, we obtain:

$$\begin{aligned}
[\![\mathcal{A}_{\mathrm{sep}}]\!](s) &= [\![\mathcal{A}_{\max}]\!](t) + \mathrm{weight}(\rho_{\mathrm{sep}}) \\
&\leqslant [\![\mathcal{A}_{\min}]\!](t) + \mathrm{weight}(\rho_{\mathrm{sep}}) && \text{(assumption } [\![\mathcal{A}_{\max}]\!] \leqslant [\![\mathcal{A}_{\min}]\!]\text{)} \\
&\leqslant [\![\mathcal{A}_{\min}]\!](s)\ . && \text{(as for the other inequality)}
\end{aligned}$$

Hence, we have established the expected $[\![\mathcal{A}_{\max}]\!](s) \leqslant [\![\mathcal{A}_{\mathrm{sep}}]\!](s) \leqslant [\![\mathcal{A}_{\min}]\!](s)$. ◀

4 Conclusion

We have established a separation result for tropical automata over trees.

All the results of this paper directly applies to automata on words. The proofs can be restated in this frameworks and are slightly easier, but their complexity actually comes from the fact that we want to encompass automata with (computable) real weights.

Our result is under the assumption that $[\![\mathcal{A}_{\max}]\!] \leqslant [\![\mathcal{A}_{\min}]\!]$. A natural variant is to invert the inequality and ask whether separation is possible when $[\![\mathcal{A}_{\min}]\!] \leqslant [\![\mathcal{A}_{\max}]\!]$. Some separation results exist in both variants (like interpolation results in logic), while some do not (separation of Büchi automata, or Lusin's theorem). For tropical automata, the assumption $[\![\mathcal{A}_{\min}]\!] \leqslant [\![\mathcal{A}_{\max}]\!]$ would be more complicated than the one in our theorem: it can be witnessed for instance by the fact that it is not decidable anymore [9].

Another interesting question is whether similar results hold for weights other than reals. For instance here, our proof requires for the weights of our automata to be equipped with a monoid structure, that it is commutative (otherwise weighted tree automata are not well defined), a total order (for the hypotheses of Theorem 2 to be meaningful), that the product be compatible with the order, and archimedianity (for the pumping argument in Lemma 9 to hold). The usefulness of each of these assumption could be studied. What if the monoid is not commutative (over words)? What if the order is not total (and be, for instance a lattice)? What if the operation is not archimedian (and what does it mean in these more general cases)? And in all these situations, do we capture interesting forms of automata?

More generally, these results of separation are fascinating, and it would be interesting to understand at high level what kind of abstract arguments may explain them, or at least some of them, uniformly.

References

1 Thomas Colcombet and Laure Daviaud. Approximate comparison of functions computed by distance automata. *Theory Comput. Syst.*, 58(4):579–613, 2016. doi:10.1007/s00224-015-9643-3.

2 Thomas Colcombet, Laure Daviaud, and Florian Zuleger. Size-change abstraction and max-plus automata. In *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, pages 208–219, 2014. doi:10.1007/978-3-662-44522-8_18.

3 Thomas Colcombet, Laure Daviaud, and Florian Zuleger. Automata and program analysis. In *Fundamentals of Computation Theory - 21st International Symposium, FCT 2017, Bordeaux, France, September 11-13, 2017, Proceedings*, pages 3–10, 2017. doi:10.1007/978-3-662-55751-8_1.

4 Stéphane Gaubert and Jean Mairesse. Idempotency. In Jeremy Gunawardena, editor, *Idempotency*, volume 11 of *Publications of the Newton Institute*, chapter Task resource models and (max,+) automata, pages 133–144. Cambridge University Press, 1998.

5 Kosaburo Hashiguchi. Limitedness theorem on finite automata with distance functions. *J. Comput. Syst. Sci.*, 24(2):233–244, 1982. doi:10.1016/0022-0000(82)90051-4.

6 Kosaburo Hashiguchi. Algorithms for determining relative star height and star height. *Inf. Comput.*, 78(2):124–169, 1988. doi:10.1016/0890-5401(88)90033-8.

7 Daniel Kirsten and Sylvain Lombardy. Deciding unambiguity and sequentiality of polynomially ambiguous min-plus automata. In *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009*, pages 589–600, 2009. doi:10.4230/LIPIcs.STACS.2009.1850.

8 Ines Klimann, Sylvain Lombardy, Jean Mairesse, and Christophe Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theor. Comput. Sci.*, 327(3):349–373, 2004. doi:10.1016/j.tcs.2004.02.049.

9 Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *IJAC*, 4(3):405–426, 1994. doi:10.1142/S0218196794000063.

10 Denis Kuperberg and Michael Vanden Boom. Quasi-weak cost automata: A new variant of weakness. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011*, pages 66–77, 2011. doi:10.4230/LIPIcs.FSTTCS.2011.66.

11 Denis Kuperberg and Michael Vanden Boom. On the expressive power of cost logics over infinite words. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, pages 287–298, 2012. doi:10.1007/978-3-642-31585-5_28.

12 Hing Leung. Limitedness theorem on finite automata with distance functions: An algebraic proof. *Theor. Comput. Sci.*, 81(1):137–145, 1991. doi:10.1016/0304-3975(91)90321-R.

13 Sylvain Lombardy and Jean Mairesse. Series which are both max-plus and min-plus rational are unambiguous. *ITA*, 40(1):1–14, 2006. doi:10.1051/ita:2005042.

14 Sylvain Lombardy and Jean Mairesse. Series which are both max-plus and min-plus rational are unambiguous. *arXiv e-prints*, page arXiv:0709.3257, September 2007. arXiv:0709.3257.

15 Michael O. Rabin. Weakly definable relations and special automata. *Mathematical Logic and Foundations of Set Theory*, pages 1–23, 1970.

16 Antonio Restivo and Christophe Reutenauer. On cancellation properties of languages which are support of rational series. *Journal of Computer System Sciences*, 29:153–159, 1984.

17 Marcel Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270, 1961. doi:10.1016/S0019-9958(61)80020-X.

18 Michał Skrzypczak. Separation property for wb- and ws-regular languages. *Logical Methods in Computer Science*, 10(1), 2014. doi:10.2168/LMCS-10(1:8)2014.

19 Denis Thérien and Thomas Wilke. Nesting until and since in linear temporal logic. *Theory Comput. Syst.*, 37(1):111–131, 2004. doi:10.1007/s00224-003-1109-3.

Asymptotic Quasi-Polynomial Time Approximation Scheme for Resource Minimization for Fire Containment

Mirmahdi Rahgoshay
Department of Computing Science, University of Alberta, Edmonton, Alberta, T6G 2E8, Canada
rahgosha@ualberta.ca

Mohammad R. Salavatipour
Department of Computing Science, University of Alberta, Edmonton, Alberta, T6G 2E8, Canada
mrs@ualberta.ca

Abstract

Resource Minimization Fire Containment (RMFC) is a natural model for optimal inhibition of harmful spreading phenomena on a graph. In the RMFC problem on trees, we are given an undirected tree G, and a vertex r where the fire starts at, called root. At each time step, the firefighters can protect up to B vertices of the graph while the fire spreads from burning vertices to all their neighbors that have not been protected so far. The task is to find the smallest B that allows for saving all the leaves of the tree. The problem is hard to approximate up to any factor better than 2 even on trees unless P = NP [11].

Chalermsook and Chuzhoy [6] presented a Linear Programming based $O(\log^* n)$ approximation for RMFC on trees that matches the integrality gap of the natural Linear Programming relaxation. This was recently improved by Adjiashvili, Baggio, and Zenklusen [1] to a 12-approximation through a combination of LP rounding along with several new techniques.

In this paper we present an asymptotic QPTAS for RMFC on trees. More specifically, let $\epsilon > 0$, and $\mathcal{I}$ be an instance of RMFC where the optimum number of firefighters to save all the leaves is $OPT(\mathcal{I})$. We present an algorithm which uses at most $\lceil (1+\epsilon)OPT(\mathcal{I}) \rceil$ many firefighters at each time step and runs in time $n^{O(\log\log n/\epsilon)}$. This suggests that the existence of an asymptotic PTAS is plausible especially since the exponent is $O(\log\log n)$, not $O(\log n)$.

Our result combines a more powerful height reduction lemma than the one in [1] with LP rounding and dynamic programming to find the solution. We also apply our height reduction lemma to the algorithm provided in [1] plus a more careful analysis to improve their 12-approximation and provide a polynomial time $(5+\epsilon)$-approximation.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Firefighter Problem, Resource Management, Fire Containment, Approximation Algorithm, Asymptotic Approximation Scheme

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.33

Funding Authors supported by organization NSERC.

1 Introduction

The Firefighter problem and a closely related problem named Resource Minimization Fire Containment (RMFC) are natural models for optimal inhibition of harmful spreading phenomena on a graph. The firefighter problem was formally introduced by Hartnell [9] and later Chalermsook and Chuzhoy [6] defined the RMFC problem. Since then, both problems have received a lot of attention in several research papers, even when the underlying graph is a spanning tree, which is one of the most-studied graph structures in this context and also the focus of this paper.

In both problems (when restricted to trees) we are given a graph $G = (V, E)$, which is a spanning tree, and a vertex $r \in V$, called root. The problem is defined over discretized time steps. At time 0, a fire starts at r and spreads step by step to neighboring vertices. During each time step $1, 2, \ldots$ any non-burning vertex u can be protected, preventing u from burning in any future time step.

In the RMFC problem the task is to determine the smallest number $B \in \mathbb{Z}_{\geq 1}$ such that there is a protection strategy which protects B vertices at each time step while saving all the leaves from catching fire. In this context, B is referred to as the number of firefighters (or budget at each step). In the firefighters problem, given a fixed number of firefighters (i.e. number of vertices that can be protected at each time step) the goal is to find a strategy to maximize the number of vertices saved from catching the fire.

For RMFC on trees, King and MacGillivray [11] showed that it is NP-hard to decide whether one firefighter is sufficient or not. This means that there is no (efficient) approximation algorithm with an approximation factor strictly better than 2, unless P=NP. On the positive side, Chalermsook and Chuzhoy [6] presented an $O(\log^* n)$-approximation, where n is the number of vertices. Their algorithm is based on a natural Linear Programming (LP) relaxation, which is a straightforward adaptation of the one previously used for the Firefighter problem on trees and essentially matches the integrality gap of the underlying LP (the integrality gap of the underlying LP is $\Theta(\log^* n)$ [6]). Recently, Adjiashvili et al. [1] presented a 12-approximation for RMFC, which is the first constant factor approximation for the problem. Their result is obtained through a combination of the known LPs with several new techniques, which allows for efficiently enumerating subsets of super-constant size of a good solution to obtain stronger LPs. They also present a $PTAS$ for the firefighter problem.

1.1 Our Results

In this paper our main focus is on RMFC problem. By using Linear Programming and dynamic programming techniques, we show how to approximate RMFC with a small additive error by presenting a quasi-polynomial time asymptotic approximation scheme (AQPTAS) for it. More specifically our main result is the following theorem:

▶ **Theorem 1.** *For RMFC on trees and for any $\epsilon > 0$ there is an algorithm that finds a solution using $\lceil (1 + O(\epsilon))B \rceil$ firefighters with running time $n^{O(\log \log n/\epsilon)}$, where B is the optimal number of firefighters.*

We will also show how applying our more powerful height reduction lemma to the algorithm used by Adjiashvili et al. [1], plus a more careful analysis, leads to a better constant factor. In particular, we obtain the following:

▶ **Theorem 2.** *For any $\epsilon > 0$, there is a polynomial time $(5 + \epsilon)$-approximation for the RMFC problem on trees.*

Recall that the RMFC problem on trees does not admit better than 2-approximation unless P = NP [11]. However, this does not rule out the possibility of a +1 approximation or an asymptotic PTAS. Our result is an indication that it is plausible that an asymptotic PTAS exists, especially since the exponent is $O(\log \log n)$, not $O(\log n)$ as we don't know any natural problem that admits $n^{O(\log \log n)}$ algorithm but not polynomial time.

We start by introducing a more powerful height reduction transformation than the one used in [1] that allows for transforming the RMFC problem into a more compact and better structured form, by only losing a $(1 + \epsilon)$ factor in terms of approximability. This transformation allows us to identify small substructures, over which we can optimize efficiently,

and having an optimal solution to these subproblems we can define a residual LP with small integrality gap. Then we will show how to apply dynamic programming on the transformed instance to obtain a strategy to protect the nodes at each step to successfully contain the fire and save all the leaves with using only $O(\epsilon B)$ more firefighters at each step. We will apply our more powerful height reduction lemma to the previous combinatorial approach [1] to reach a better constant factor approximation in polynomial time, which is presented in Theorem 2.

1.2 Further Related Work

The Firefighter problem and RMFC, both restricted to trees, are known to be computationally hard problems. More precisely, Finbow, King, MacGillivray and Rizzi [7] showed the NP-hardness for the Firefighter problem on trees even when the maximum degree is three. For RMFC on trees, it is NP-hard to decide whether one firefighter is sufficient or not [11], which implies that the problem is hard to approximate to a factor better than 2.

Several approximation algorithms have been proposed for both of these problems. Hartnell and Li [8] proved that a natural greedy algorithm is a $\frac{1}{2}$-approximation for the Firefighter problem. Later, Cai, Verbin and Yang [3] improved this result to $1 - \frac{1}{e}$, using a natural LP relaxation and dependent randomized rounding. Then Anshelevich, Chakrabarty, Hate and Swamy [2] showed that the Firefighter problem on trees can be interpreted as a monotone submodular function maximization (SFM) problem subject to a partition matroid constraint. This observation yields another $(1-\frac{1}{e})$-approximation by using a recent $(1-\frac{1}{e})$-approximation for monotone SFM subject to a matroid constraint [4, 13].

Chalermsook and Vaz [5] showed that, for any $\epsilon > 0$, the canonical LP used for the Firefighter problem on trees has an integrality gap of $1 - \frac{1}{e} + \epsilon$. This generalized a previous result by Cai, Verbin and Yang [3]. When restricted to some tree topologies this factor $1 - \frac{1}{e}$ was later improved (see [10]) but, for arbitrary trees, that was the best known approximation factor for a few years.

Recently, Adjiashvili, Baggio and Zenklusen [1] have filled the gap between previous approximation ratios and hardness results for the Firefighter problem. In particular, they present approximation ratios that nearly match the hardness results, thus showing that the Firefighter problem can be approximated to factors that are substantially better than the integrality gap of the natural LP. Their results are based on several new techniques, which may be of independent interest.

Assuming a variant of the Unique Games Conjecture (UGC), the RMFC problem in general graphs is hard to approximate within any constant factor, according to a recent work by Lee [12] which is based on a general method of converting an integrality gap instance to a length-control dictatorship test for variants of the s-t cut problem. For further results and related work we refer the reader to [1].

1.3 Organization of the Paper

In Section 2 we start by introducing some preliminaries including a (now standard) Linear Programming relaxation for the problem and then will provide a height reduction lemma. Section 3 will cover our main algorithm to obtain the asymptotic QPTAS. In Appendix A we will show how to apply our height reduction lemma to the previous combinatorial approach of [1] to improve their 12-approximation and provide a $(5 + \epsilon)$-approximation.

2 Preliminaries and Overview of the Algorithm

Recall that we are given a tree $G = (V, E)$ rooted at a vertex r, from which we assume the fire starts. We denote by $\Gamma \subseteq V$ the set of all leaves of the tree. Given an instance $\mathcal{I}$ for RMFC and an integer parameter $B \geq 1$, called the budget or the number of firefighters, at each time step we can "protect" up to B non-burning vertices. Such vertices are protected indefinitely. Our goal is to find the smallest B and a protection strategy such that all the leaves Γ are saved from catching the fire. Observe that we say a vertex u is protected, if we directly place a firefighter in u, and a vertex v is saved when the fire does not reach to u, because of protecting some u on the unique v-r path. This smallest value of B is denoted by $OPT(\mathcal{I})$.

Let $L \in \mathbb{Z}_{\geq 1}$ be the depth of the tree, i.e. the largest distance, in terms of the number of edges, between r and any other vertex in G. After at most L time steps, the fire spreading process will halt. For $\ell \in [L] := \{1, \ldots, L\}$, let $V_\ell \subseteq V$ be the set of all vertices of distance ℓ from r, which we call the ℓ-th level of the instance. We also use $V_{\leq \ell} = \cup_{k=1}^{\ell} V_k$, and we define $V_{\geq \ell}$, $V_{<\ell}$, and $V_{>\ell}$ in the same way. Moreover, for each $1 \leq \ell < L$ and each $u \in V_\ell$, $P_u \subseteq V_{\leq \ell} \setminus \{r\}$ denotes the set of all vertices on the unique u-r path except for the root r, and $T_u \subseteq V_{>\ell}$ denotes the subtree rooted at u, i.e. descendants of u.

2.1 Linear Programming Relaxation

We use the following (standard) Linear Programming (LP) relaxation for the problem that is used in both [6] and [1].

$$\begin{aligned} \min \quad & B & & & (1) \\ & x(P_u) & \geq & 1 \qquad \forall u \in \Gamma \\ & x(V_{\leq \ell}) & \leq & B \cdot \ell \qquad \forall \ell \in [L] \\ & x & \in & \mathbb{R}_{\geq 0}^{V \setminus \{r\}} \end{aligned}$$

Here $x(U) := \sum_{u \in U} x(u)$ for any $U \subseteq V \setminus \{r\}$. Note that with $x \in \{0,1\}^{V \setminus \{r\}}$ and $B \in \mathbb{Z}_{\geq 0}$ we get an exact description of RMFC where x is the characteristic vector of the vertices to be protected and B is the budget. The first constraint enforces that for each leaf u, one vertex between u and r will be protected, which makes sure that the fire will not reach u. The second constraint ensures that the number of vertices protected after each time step is at most $B \cdot \ell$ and makes sure that we are using no more than B firefighters per time step (see [6] for more details). Note that (as mentioned in [6]), there is an optimal solution to RMFC that protects, with the firefighters available at time step ℓ, only the vertices in V_ℓ. Hence, we can change the above relaxation to one with the same optimal objective value by replacing the constraints $x(V_{\leq \ell}) \leq B \cdot \ell$ by the constraints $x(V_\ell) \leq B$ for all $\ell \in [L]$.

$$\begin{aligned} \min \quad & B & & & (2) \\ & x(P_u) & \geq & 1 \qquad \forall u \in \Gamma \\ & x(V_\ell) & \leq & B \qquad \forall \ell \in [L] \\ & x & \in & \mathbb{R}_{\geq 0}^{V \setminus \{r\}} \end{aligned}$$

Throughout the paper we use a lemma of [1] which basically says that any basic feasible solution of LP(2) (and also LP(1)) is sparse. This is proved for the polytope of the firefighters problem, which has the same LP constraints (just different objective function). Consider any basic feasible solution x to LP(2). One can partition $supp(x) = \{v \in V \setminus \{r\} : x(v) > 0\}$ into

two parts: x-loose vertices and x-tight vertices. A vertex $v \in V \setminus \{r\}$ is x-loose or simply loose if $v \in supp(x)$ and $x(P_v) < 1$. All other vertices in $supp(x)$, which are not loose, will be x-tight or simply tight.

▶ **Lemma 3** (Lemma 6 in [1]). *Let x be a vertex solution to LP(2) for RMFC, then the number of x-loose vertices is at most L, the depth of the tree.*

We will use this property crucially in the design of our algorithm. Also, as noted in [1], we can work with a slightly more general version of the problem in which we have different numbers of budgets/firefighters at each time step: say $B_\ell = m_\ell B$ (for some $m_\ell \in \mathbb{Z}_{\geq 0}$) firefighters for each time step $\ell \in [L]$ while we are still minimizing B. Lemma 3 is valid for this generalization too.

2.2 Height Reduction

The technique of reducing the height of a tree at a small loss in cost (or approximation ratio) has been used in different settings and various problems (e.g. network design problems). For RMFC, Adjiashvili et al. [1] showed how one can reduce an instance of the problem to another instance where the height of the tree is only $O(\log n)$ at a loss of factor 2. In a sense, the tree will be compressed into a tree with only $O(\log n)$ levels. Here we introduce a more delicate version of that compression, which allows for transforming any instance to one on a tree with $O(\frac{\log n}{\epsilon})$ levels at a loss of $1+\epsilon$ in the approximation. Our compression is similar to that of [1] with an initial delay and ratio $1+\epsilon$. One key property we achieve with compression, is that we can later use techniques with running time exponential in the depth of the tree.

Suppose that the initial instance is a tree with L levels and each level ℓ has a budget B_ℓ. To compress the tree to a low height one, we will first do a sequence of what is called up-pushes. Each up-push acts on two levels $\ell_1, \ell_2 \in [L]$ with $\ell_1 < \ell_2$ of the tree, and moves the budget B_{ℓ_2} of level ℓ_2 up to ℓ_1. This means the new budget of level ℓ_1 will be $B_{\ell_1} + B_{\ell_2}$ and for level ℓ_2 it will be 0.

We will show that one can do a sequence of up-pushes such that: (i) the optimal objective value of the new instance is very close to the one of the original instance, and (ii) only $O(\log L/\epsilon)$ levels have non-zero budgets. Finally, 0-budget levels can easily be removed through a simple contraction operation, thus leading to a new instance with only $O(\log L/\epsilon)$ depth. The following theorem is a more powerful version of Theorem 5 in [1] with some improvements such as reducing the loss to only $1+\epsilon$ (instead of 2) and some differences in handling of the first levels.

▶ **Theorem 4.** *Let $G = (V, E)$ be a rooted tree of depth L. Then for some constants $c, d > 0$ (that only depend on ϵ) we can construct efficiently a rooted tree $G' = (V', E')$ with $|V'| \leq |V|$ and depth $L' = O(\frac{\log L}{\epsilon})$, such that:*

(i) If the RMFC problem on G has a solution with budget $B \in \mathbb{Z}_{\geq 0}$ at each level, then the RMFC problem on G' has a solution with non-uniform budgets of $B_\ell = B$ for each level $\ell < c$, and a budget of $B_\ell = m_\ell \cdot B$ for each level $\ell \geq c$, where $m_\ell = \left(\lceil (1+\epsilon)^{(\ell-d+1)} \rceil - \lceil (1+\epsilon)^{(\ell-d)} \rceil\right)$.

(ii) Any solution to the RMFC problem on G', where each level $\ell < c$ has a budget of $B_\ell = B$ and each level $\ell \geq c$ has a budget of $B_\ell = m_\ell \cdot B$ can be transformed efficiently into an RMFC solution for G with budget $\lceil (1+2\epsilon)B \rceil$.

Proof. We start by describing the construction of $G' = (V', E')$ from G. We first change the budget assignment of the instance and then contract all 0-budgets levels.

We set i^* to be the smallest integer such that $(1+\epsilon)^{i^*} \geq \frac{2(1+\epsilon)}{\epsilon^2}$ and we let $c = \lceil (1+\epsilon)^{i^*} \rceil$. The set of levels $\mathcal{L}$ in which the transformed instance will have non-zero budget contains the first $c-1$ levels of G and all the levels $\ell \geq c$ of G such that $\ell = \lceil (1+\epsilon)^i \rceil$ for some $i^* \leq i \leq \frac{\log L}{\log(1+\epsilon)} = O(\log L/\epsilon)$:

$$\mathcal{L} = \left\{ 1 \leq \ell \leq L \quad | \quad \ell < c \quad or \quad \ell = \lceil (1+\epsilon)^i \rceil \text{ for some } i^* \leq i \leq \left\lfloor \frac{\log L}{\log(1+\epsilon)} \right\rfloor \right\}$$

For all other levels $\ell \notin \mathcal{L}$ we first do up-pushes. More precisely, the budget of these levels $\ell \in [L] \setminus \mathcal{L}$ will be assigned to the closest level in $\mathcal{L}$ that is above ℓ (has smaller index than ℓ). We then remove all 0-budget levels by contraction. For each vertex v in a level $\ell_i = \lceil (1+\epsilon)^i \rceil \geq c$ we will remove all vertices in the levels $\ell_i < \ell < \ell_{i+1} = \lceil (1+\epsilon)^{i+1} \rceil$ from its sub-tree and connect all the vertices in level ℓ_{i+1} of its sub-tree to v directly. This leads to a new tree G' with a new set of leaves. Since our goal is to save all the leaves in the original instance, for each vertex $v \in G'$ such that $v \in G$ has some leaves in its contracted sub-tree, we will mark v as a leaf in G' and simply delete all its remaining subtree.

This finishes our construction of $G' = (V', E')$ and it remains to show that both (i) and (ii) hold. Note that the levels in G' correspond to levels of G in $\mathcal{L}$: the first c levels of G' are the same as the first c levels of G; for each $\ell > c$, level ℓ in G' is level $\lceil (1+\epsilon)^{\ell - c + i^*} \rceil$ of G.

Here we want to determine what will be the budget of each level of G'. For each $\ell < c = \lceil (1+\epsilon)^{i^*} \rceil$, the level ℓ of G' is the same as the level ℓ of G and has the same budget $B_\ell = B$, because these levels are not involved in up-pushes. For $\ell = c$, all the budgets from level $\lceil (1+\epsilon)^{i^*} \rceil$ to $\lceil (1+\epsilon)^{i^*+1} \rceil - 1$ in G are up-pushed to this level. This means that the budget for level c in G' is $B_c = \left(\lceil (1+\epsilon)^{i^*+1} \rceil - \lceil (1+\epsilon)^{i^*} \rceil \right) \cdot B$. Now for each $i^* < i \leq \lfloor \frac{\log L}{\log(1+\epsilon)} \rfloor$, all the budgets from levels $\lceil (1+\epsilon)^i \rceil$ to $\lceil (1+\epsilon)^{i+1} \rceil - 1$ in G are up-pushed to level $\lceil (1+\epsilon)^i \rceil$, which becomes level $i - i^* + c$ in G'; this means that the budget for this level of G' will be $\lceil (1+\epsilon)^{i+1} \rceil - \lceil (1+\epsilon)^i \rceil$. Setting $\ell = i - i^* + c$ and $d = c - i^*$, the budget of level ℓ in G', is $B_\ell = (\lceil (1+\epsilon)^{\ell - d + 1} \rceil - \lceil (1+\epsilon)^{\ell - d} \rceil) \cdot B$. To prove (ii), we use the following lemma:

▶ **Lemma 5.** *For any two consecutive levels $\ell \geq c$ and $\ell + 1$ in G', the difference between m_ℓ and $m_{\ell+1}$ is relatively small. More precisely: $m_\ell(1+2\epsilon) \geq m_{\ell+1}$*

Proof. Based on the definition of m_ℓ and $m_{\ell+1}$ we have:

$$\begin{aligned} m_\ell &= \lceil (1+\epsilon)^{(\ell-d+1)} \rceil - \lceil (1+\epsilon)^{(\ell-d)} \rceil \geq (1+\epsilon)^{(\ell-d+1)} - (1+\epsilon)^{(\ell-d)} - 1 \\ \Rightarrow m_\ell(1+\epsilon) &\geq (1+\epsilon)^{(\ell-d+2)} - (1+\epsilon)^{(\ell-d+1)} - (1+\epsilon). \end{aligned} \tag{3}$$

On the other hand:

$$\begin{aligned} m_{\ell+1} &= \lceil (1+\epsilon)^{(\ell-d+2)} \rceil - \lceil (1+\epsilon)^{(\ell-d+1)} \rceil \leq (1+\epsilon)^{(\ell-d+2)} - (1+\epsilon)^{(\ell-d+1)} + 1 \\ &\leq m_\ell(1+\epsilon) + 2 + \epsilon \qquad \text{using (3)} \end{aligned} \tag{4}$$

Also by our choice of c, d and $i^* = c - d$ we can conclude that:

$$\begin{aligned} m_\ell &= \lceil (1+\epsilon)^{(\ell-d+1)} \rceil - \lceil (1+\epsilon)^{(\ell-d)} \rceil \\ &\geq (1+\epsilon)^{(\ell-d+1)} - (1+\epsilon)^{(\ell-d)} - 1 = \epsilon(1+\epsilon)^{(\ell-d)} - 1 \\ &\geq \epsilon(1+\epsilon)^{(c-d)} - 1 = \epsilon \cdot (1+\epsilon)^{i^*} - 1 \geq \epsilon \frac{2(1+\epsilon)}{\epsilon^2} - 1 \\ \Rightarrow \quad m_\ell &\geq \frac{2+\epsilon}{\epsilon} \Rightarrow \quad \epsilon m_\ell \geq 2 + \epsilon. \end{aligned} \tag{5}$$

Combining (4) and (5) completes the proof. ◀

▶ **Corollary 6.** *For each $\ell \geq c$ and each budget $B > 0$:*

$$m_{\ell+1} \cdot B \leq m_\ell \cdot \lceil (1+2\epsilon)B \rceil$$

Notice that in the constructed graph G' for each level $\ell \geq c$, we have $B_\ell = m_\ell \cdot B$. Now consider the instance of the problem on graph G with budget $\lceil (1+2\epsilon)B \rceil$ at each level. We will show that by doing some down-pushes on G (i.e. move the budget of each level to some level down) we can construct G' again where the budget of each level ℓ is $m_\ell \cdot B$, and this means that if G' has a solution with budget $m_\ell \cdot B$ in each level, then G has a solution with uniform budget $\lceil (1+2\epsilon)B \rceil$.

Like before the set of levels $\mathcal{L}$ with non-zero budgets will be the same. Instead of up-pushes, we will down-push the budget from all levels $\ell \notin \mathcal{L}$ to the closest level in $\mathcal{L}$ which is below ℓ (i.e has larger index than ℓ). We will also down-push budget $\lceil 2\epsilon B \rceil$ from each level $\ell < c$ to level $\ell = c$.

By doing the same contraction, for each level $\ell < c$ we will have $B_\ell = B$ and for each level $\ell > c$ we will have $B_\ell = m_{\ell-1} \cdot \lceil (1+2\epsilon)B \rceil$, which is greater than $m_\ell \cdot B$ based on the above lemma.

The only remaining level to consider is level $\ell = c$. For this level, by doing down-pushes, we will have budget $B_c = B + \lceil 2\epsilon B \rceil \cdot c$. Our claim is that this is not less than $m_c \cdot B$, which is equal to $(\lceil (1+\epsilon)c \rceil - c) \cdot B$ (based on the definition of m_c):

$$\begin{aligned} B_c &= B + \lceil 2\epsilon B \rceil \cdot c \\ &\geq B + 2\epsilon B \cdot c = (1 + 2\epsilon c) \cdot B \\ &\geq \lceil 2\epsilon c \rceil \cdot B = \lceil (1+2\epsilon)c - c \rceil \cdot B \\ &\geq (\lceil (1+\epsilon)c \rceil - c) \cdot B = m_c \cdot B. \end{aligned}$$

This will complete the proof of the theorem, because by considering these down-pushes, any solution to the RMFC problem on G', where level $\ell \geq c$ has a budget of $B_\ell = m_\ell \cdot B$ and level $\ell < c$ has a budget of $B_\ell = B$, can be transformed efficiently into an RMFC solution for G with budget $\lceil (1+2\epsilon)B \rceil$. ◀

In the following we assume that the depth of the tree is not more than $\frac{\log n}{\log(1+\epsilon)} + \frac{2(1+\epsilon)}{\epsilon^2}$, so $L = O(\frac{\log n}{\epsilon})$. After finding a solution with budget B for a tree with this height, then we could apply the compression theorem and find a solution for the original tree by having $\lceil \epsilon B \rceil$ more firefighters at each level.

2.3 Overview of the Algorithm

Given an instance $\mathcal{I}$, our first step of the algorithm is to use Theorem 4 to reduce $\mathcal{I}$ to an instance $\mathcal{I}'$ with $L = O(\log n/\epsilon)$ levels. Note that when we use B to refer to *core* budget for instance $\mathcal{I}'$ we mean each level ℓ has budget $m_\ell \cdot B$ for $\ell \geq c$, and budget B for each level $\ell < c$. Also, by $OPT(\mathcal{I}')$ we mean the smallest value B such that $\mathcal{I}'$ has a feasible solution with core budget B as above. By Theorem 4, if we find a solution with *core* budget B for $\mathcal{I}'$ then it can be transformed to a solution for $\mathcal{I}$ with budget $\lceil (1+2\epsilon)B \rceil$. So we focus on the height reduced instance $\mathcal{I}'$ from now on. We present an algorithm such that if $B \geq OPT(\mathcal{I}')$ then it finds a feasible solution to $\mathcal{I}'$ with core budget at most $\lceil (1+\epsilon)B \rceil$. Then, using binary search, we find the smallest value of B_o (for B) for which the algorithm finds a feasible solution. This would give us a solution of budget at most $\lceil (1+\epsilon)OPT(\mathcal{I}') \rceil$, which in turn implies a solution for $\mathcal{I}$ of value at most $\lceil (1+O(\epsilon))OPT(\mathcal{I}) \rceil$.

So let us assume we have guessed a value $OPT(\mathcal{I}') \leq B_o$. We consider LP(2) (with fixed $B = B_o$) for $\mathcal{I}'$ with guessed core budget B_o. Let x^* be a basic feasible solution to this instance. Using Lemma 3 we know that there are at most L loose vertices. As we will see, when B_o is relatively large, i.e. $B_o > \frac{L}{\epsilon}$, then we can easily find an integer solution using core budget at most $\lceil (1+\epsilon)B_o \rceil$ and this yields the desired bound for the original instance.

The difficult case is when B_o is small compared to L. The difficulty lies in deciding which vertices are to be protected by the optimum solution in the top h levels of the tree for some $h = O(\log \log n)$; as if one has this information then we can obtain a good approximation as in [1].

One way to do this would be to guess all the possible subsets of vertices that could be protected by the optimal solution in the first h levels of the tree, but this approach would have a running time far greater than ours. Still, we can solve the problem on instance $\mathcal{I}'$ in quasi-polynomial time using a bottom-up dynamic programming approach. More precisely, starting with the leaves and moving up to the root, we compute for each vertex $u \in V$ the following table. Consider a subset of the available budgets, which can be represented as a vector $q \in [B_1] \times ... \times [B_L]$. For each such vector q and node v, we want to know whether or not using budgets described by q for the subtree T_v (subtree rooted at v) allows for disconnecting v from all the leaves below it, i.e. saving all the leaves in T_v. Since $L = O(\log n/\epsilon)$ and the size of each budget B_ℓ is at most the number of vertices, the table size is $n^{O(\log n/\epsilon)}$). Moreover, it is easy to show that this table can be constructed bottom-up in quasi-polynomial time using an auxiliary table and another dynamic programming, to fill each cell of the table.

This approach would have the total running time of $n^{O(\log n/\epsilon)}$, because of the size of the table. In order to reduce the running time to $n^{O(\log \log n/\epsilon)}$, we would consider each budget vector value rounded up to the nearest power of $(1+\frac{\epsilon^2}{(\log n)^2})$. So, instead of $O(n^L) = n^{O(\log n/\epsilon)}$ many options for budget vectors q, we will have $O((\log n/\epsilon)^{3L}) = n^{O(\log \log n/\epsilon)}$ many options and we will show how by being more careful in our dynamic programming on these budget vectors we can still compute the table in time $n^{O(\log \log n/\epsilon)}$; this leads to an approximation scheme (instead of the exact algorithm) for the instance $\mathcal{I}'$.

3 Asymptotic Approximation Scheme

As mentioned above, first we use the height reduction as discussed in the previous section to reduce the given instance $\mathcal{I}$ to a new one $\mathcal{I}'$ with $L = O(\frac{\log n}{\epsilon})$ levels. We assume we have guessed a value $B_o \geq OPT(\mathcal{I}')$. Recall that, as in the statement of Theorem 4, for some constants c, d (depending on ϵ) the budget of each level $\ell < c$ is $B_\ell = B_o$ and for each level $\ell \geq c$ the budget is $B_\ell = m_\ell \cdot B_o$ where $m_\ell = \left(\lceil (1+\epsilon)^{(\ell-d+1)} \rceil - \lceil (1+\epsilon)^{(\ell-d)} \rceil\right)$.

We consider two cases: (I) when $B_o > \frac{L}{\epsilon}$, and (II) when $B_o \leq \frac{L}{\epsilon}$. For the first case we show how we can find a solution with core budget at most $\lceil (1+\epsilon)B_o \rceil$ by rounding the standard Linear Programming relaxation. For the second case we show how we can use a bottom-up dynamic programming approach to find a quasi-polynomial time approximation scheme.

3.1 Easy Case: $B_o > \frac{L}{\epsilon}$

In this case we consider LP(2) (with fixed $B = B_o$) for this instance. If x^* is a feasible solution to this LP and $B_o > \frac{L}{\epsilon}$ then we add $L \leq \lceil \epsilon B_o \rceil$ extra budget (i.e. number of firefighters) to the first level which is enough to protect all the *loose* vertices. Since by using Lemma 3 we know that there are at most L loose vertices and we can protect them all in the first step using L extra firefighters.

It remains to show that by using a budget of $m_\ell \cdot B_o$ at every level ℓ, for $c \leq \ell \leq L$, and B_o for $\ell < c$, we can protect all the tight vertices and so all the leaves would be saved, by adding only L many extra firefighters to only the first level.

Observe that for each tight vertex v, either $x(v) < 1$, then we would have a loose vertex in P_v, or $x(v) = 1$. In the first case v is already saved by protecting the loose vertices in the first step. If we only consider vertices with $x(v) = 1$, we can see that the solution is integral itself for these vertices. So we have rounded a fractional solution with $B_o > \frac{L}{\epsilon}$ to an integral one by using only $\lceil \epsilon B_o \rceil$ more firefighters just in the first level. In this case we find a feasible solution with core budget $B_o + \lceil \epsilon B_o \rceil$ in polynomial time.

3.2 When $B_o \leq \frac{L}{\epsilon}$

Recall that we have a budget of $B_\ell = B_o < L/\epsilon$ for each level $\ell < c$ and $B_\ell = m_\ell \cdot B_o \leq m_\ell \cdot \frac{L}{\epsilon}$ for each $c \leq \ell \leq L$. We denote by q^* the L-dimensional total budget vector that has $q^*[\ell] = B_\ell$ for each $1 \leq \ell \leq L$. Also for each L-dimensional vector $q \in [B_1] \times [B_2] \times ... \times [B_L]$, we denote by $Q(q)$ the set of all vectors q' such that $q' \leq q$. Suppose that $|Q(q^*)| = m$. We first describe a simpler (and easier to explain) dynamic programming with running time $n^{O(\log n/\epsilon)}$. Then we change it to decrease the running time and have our final approximation scheme with running time $n^{O(\log \log n/\epsilon)}$.

3.2.1 First Algorithm

Our dynamic program (DP) consists of two DP's: an outer (main) DP and an inner DP. In our main DP table A we have an entry for each vertex v and each vector $q \in Q(q^*)$. This entry, denoted by $A[v, q]$, will store whether using budgets described by q for levels of T_v allows for disconnecting v from all leaves below it or not.

More formally, if we assume $v \in V_\ell$, then $A[v, q]$ would be true if and only if there is a strategy for T_v such that (i) all the leaves in T_v are saved, and (ii) the budget for levels of T_v are given by vector q in indices $\ell + 1, \ldots, L$, i.e. $q[\ell + 1]$ for the first level of T_v (direct children of v) , $q[\ell + 2]$ for the second level, and so on.

We compute the entry $A[.,.]$ in a bottom up manner, computing $A[v, q]$ after we have computed the entries for children of v. To compute cell $A[v, q]$, we would use another auxiliary table B. Suppose v has k children $u_1, \ldots, u_k$ and assume that we have already calculated $A[u_j, q']$ for every $1 \leq j \leq k$ and all vectors $q' \in Q(q)$. Then we define a cell in our auxiliary table $B[v, q', j]$ for each $1 \leq j \leq k$ and $q' \in Q(q)$, where $B[v, q', j]$ is supposed to determine if the budget vector q' is enough for the union of subtrees rooted at $u_1, \ldots, u_j$ to save all the leaves in $T_{u_1} \cup \ldots T_{u_j}$ or not, where the total budgets for union of those subtrees are given by q'. We can compute $B[v, q', j]$ having computed $A[u_j, q'']$ and $B[v, q' - q'', j - 1]$ for all $q'' \in Q(q')$. This means that we can compute each cell $A[v, q]$ using auxiliary table B and internal DPs and the running time is $O(n^2 \cdot m^3)$. We need to find $A[r, q^*]$. If this cell is true, then we can save all the leaves of the tree using q^* as the budget vector for each level and if it is false, B_o would not be enough.

The problem is that m_ℓ could be large ($m_L = O(n)$) and so the options we have for the budget of each level is $O(n)$. Recall that we can have $B_o \leq \frac{L}{\epsilon}$ many choices for $q[\ell]$ when $\ell < c$ and $m_\ell \cdot \frac{L}{\epsilon}$ many options when $c \leq \ell \leq L$. Using the definition of the m_ℓ: $m_\ell = O(\epsilon(1 + \epsilon)^{\ell - d})$, and so the total possible different budget vectors we could have is:

$$m = \left(\frac{L}{\epsilon}\right)^{c-1} \times \prod_{\ell=c}^{L} \left(m_\ell \cdot \frac{L}{\epsilon}\right) \quad = \quad \left(\frac{L}{\epsilon}\right)^{c-1} \times L^{L-c+1} \times \prod_{\ell=c}^{L} \left((1+\epsilon)^{\ell-d}\right) \quad = \quad O\left((nL)^L\right)$$

This means that the total running time will be $O(n^L) = n^{O(\log n/\epsilon)}$ and this is an exact algorithm to solve the RMFC problem on instance $\mathcal{I}'$.

3.2.2 Reducing Budget Possibilities

To reduce the running time, we only consider budget vectors where each entry of the vector is a power of $(1+(\epsilon/\log n)^2)$. In this case we have at most $O\big(\log(m_\ell \cdot L)\times(\frac{\log n}{\epsilon})^2\big) = O(\log^3 n)$ many options for ℓth entry of q for each $c \le \ell \le L$, and so $m = O((\log n)^L) = n^{O(\log\log n/\epsilon)}$. Also, we have to show how we can compute the entries of the table in time $n^{O(\log\log n/\epsilon)}$ and why this would give a $(1+\epsilon)$-approximation of the solution. For each real x, let $RU(x)$ denote the value obtained by rounding up x to the nearest power of $(1+(\epsilon/\log n)^2)$. The main idea is that if for each vector q we round up each entry q_i to $RU(q_i)$ and denote the new vector by $RU(q)$ then if $A[v,q] = true$ then $A[v, RU(q)]$ is also true. So we only try to fill in entries of the table that correspond to vectors q where each entry is a power of $(1+(\epsilon/\log n)^2)$. We show this can be done in time $n^{O(\log\log n/\epsilon)}$ and the total loss in approximation is at most $1+\epsilon$ at the root of the tree.

From now on, we assume each vector q has entries that are powers of $(1+(\epsilon/\log n)^2)$; and recall that $Q(q)$ is the set of all such vectors q' such that $q \le q'$ and assume we have already calculated $A[u_j, q']$ for every vector $q' \in Q(q)$ (again with all entries being powers of $(1+(\epsilon/\log n)^2)$).

If we try to compute $A[v,q]$ from $A[u_j,q']$'s the same way, we need to calculate $B[v,q',j]$ for each $1 \le j \le k$ and each time we round up the results of addition/subtractions (such as $q-q'$) to the nearest power of $(1+(\epsilon/\log n)^2)$.

3.2.3 Reducing Height of Inner Table

To compute cell $A[v,q]$ then this round-up operation could happen $k = O(n)$ times and the approximation loss blows up. Instead, we consider a hypothetical full binary tree with root v and leaves (at the lowest level) being $u_1,\dots,u_k$; this tree will have height $O(\log k) = O(\log n)$. Then we define a cell in our auxiliary table for each internal node of this tree. See Figure 1 for an illustration.

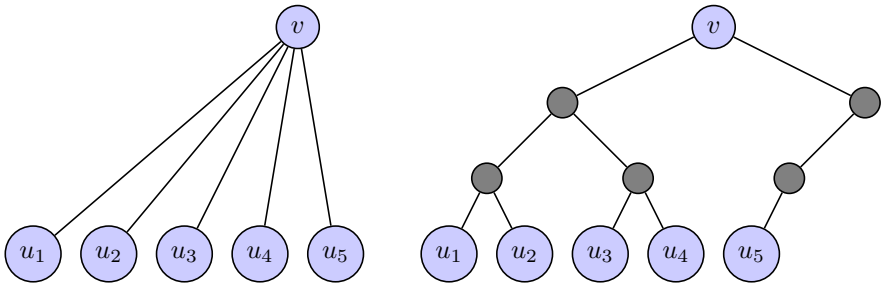


Figure 1 Illustration of the hypothetical full binary tree with root v and leaves $u_1,\dots,u_5$.

More formally we would define a cell in our auxiliary table $B[v,q',j,j']$ for each $0 \le j \le \lceil\log k\rceil$, $1 \le j' \le \lceil\frac{k}{2^j}\rceil$ and $q' \in Q(q)$ with all entries being powers of $(1+(\epsilon/\log n)^2)$, where $B[v,q',j,j']$ is supposed to determine if the budget vector q' is enough for the subtrees rooted at $u_{j_1},\dots,u_{j_2}$, where $j_1 = 2^j\cdot(j'-1)+1$ and $j_2 = min\{2^j\cdot j', k\}$, to save all the leaves in those subtrees, where the total budgets for the union of those subtrees is given by q'.

Similar to what we did before, we can compute $B[v,q',j,j']$ having computed $B[v,q'',j-1,2j'-1]$ and $B[v,RU(q'-q''),j-1,2j']$ (if it exists) for all $q'' \in Q(q')$. At each step we are computing a cell in table B a round-up will be applied to make the result of vector subtraction to be a vector with entries being powers of $(1+(\epsilon/\log n)^2)$. If we can find a q'' such that both $B[v,q'',j-1,2j'-1]$ and $B[v,RU(q'-q''),j-1,2j']$ are true, then $B[v,q',j,j']$ would be true too. Also we can fill $A[v,q]$ by checking the value of $B[v,q_i,\lceil\log k\rceil,1]$.

In the way we construct our auxiliary tables, while computing $A[v,q]$, when v has k children, $\log k$ many round up operations have happened (going up the auxiliary tree with root v) to the solution we found for T_v only in this step. This means that $O(\log k) \leq O(\log n)$ many round-ups could happen to compute entry $A[v,q]$ and the total number of round-ups starting from the values of $A[.,.]$ at a leaf level to $A[r,q]$ (for any q) would be at most $L \times \log n \leq \frac{\log^2 n}{\epsilon}$ and at each round-up we increase our budget by a factor of $(1+(\epsilon/\log n)^2)$. So the total approximation increase while computing the entries for $A[r,.]$ would be at most:

$$\big(1+\frac{\epsilon^2}{(\log n)^2}\big)^{\frac{\log^2 n}{\epsilon}} \quad = \quad 1+O(\epsilon)$$

Observe that for every node v and subtree T_v if there is a solution with budget vectors q then there is a solution with budget vector $RU(q)$ as well. Using this fact we can find a solution with budget vector at most $(1+O(\epsilon))q^*$ if there exists a solution with budget vector q^*. This completes the proof of Theorem 1.

4 Conclusion

In this paper we presented an asymptotic QPTAS for RMFC on trees. More specifically, let $\epsilon > 0$, and $\mathcal{I}$ be an instance of RMFC where the optimum number of firefighters is $OPT(\mathcal{I})$. We presented an algorithm that uses at most $\lceil (1+\epsilon)OPT(\mathcal{I}) \rceil$ many firefighters at each step and runs in time $n^{O(\log\log n/\epsilon)}$. Our result combines a more powerful height reduction lemma than the one in [1] by using dynamic programming to find the solution. We also provide a polynomial time $(5+\epsilon)$-approximation for the problem by applying our height reduction lemma to the algorithm provided in [1] as well as some minor changes to improve the best previously known 12-approximation (Appendix A).

We believe that it should be possible to have an asymptotic PTAS for the RMFC problem. Perhaps one way is to somehow guess the upper part of the optimal solution in polynomial time and then use the LP to round the solution for the height reduced instance for which we initially applied the height reduction lemma.

References

1. David Adjiashvili, Andrea Baggio, and Rico Zenklusen. Firefighting on trees beyond integrality gaps. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 2364–2383, Philadelphia, PA, USA, 2017. Society for Industrial and Applied Mathematics. URL: `http://dl.acm.org/citation.cfm?id=3039686.3039842`.
2. E. Anshelevich, D. Chakrabarty, A. Hate, and C. Swamy. Approximability of the firefighter problem. In *Algorithmica*, pages 520–536, 2012.
3. L. Cai, E. Verbin, and L. Yang. Firefighting on trees:$(1-\frac{1}{e})$–approximation, fixed parameter tractability and a subexponential algorithm. In *Proceedings of the 19th International Symposium on Algorithms and Computation*, ISAAC '08, pages 258–269. Springer-Verlag, 2008.
4. G. Calinescu, C. Chekuri, M. Pal, and J. Vondrak. Maximizing a monotone submodular function subject to a matroid constraint. In *SIAM Journal on Computing*, pages 1740–1766, 2011.
5. P. Chalermsook and D. Vaz. New integrality gap results for the firefighters problem on trees. In *Approximation and Online Algorithms, Cham, Springer International Publishing*, pages 65–77, 2017.
6. Parinya Chalermsook and Julia Chuzhoy. Resource minimization for fire containment. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 1334–1349, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics. URL: `http://dl.acm.org/citation.cfm?id=1873601.1873709`.

7 S. Finbo, A. King, G. MacGillivray, and R. Rizzi. The firefighter problem for graphs of maximum degree three. In *Discrete Mathematics*, pages 2094–2105, 2007.

8 B. Hartnell and Q. Li. Firefighting on trees: how bad is the greedy algorithm? In *Proceedings of Congressus Numerantium*, pages 187–192, 2000.

9 B.L. Hartnell. Firefighter! an application of domination. In *24th Manitoba Conference on Combinatorial Mathematics and Computing*, 1995.

10 Y. Iwaikawa, N. Kamiyama, and T. Matsui. Improved approximation algorithms for firefighter problem on trees. In *IEICE Transactions on Information and Systems*, pages 196–199, 2011.

11 A. King and G. MacGillivray. The firefighter problem for cubic graphs. In *Discrete Mathematics*, pages 614–621, 2010.

12 Euiwoong Lee. Improved hardness for cut, interdiction, and firefighter problems. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 92:1–92:14, 2017. URL: `https://doi.org/10.4230/LIPIcs.ICALP.2017.92`.

13 J. Vondrak. Optimal approximation for the submodular welfare problem in the value oracle model. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 67–74, 2008.

A Polynomial $(5+\epsilon)$-Approximation for RMFC

In this section we show how the approach introduced in [1] can be adapted so that along with our height reduction lemma gives a $(5+\epsilon)$-approximation. We largely follow the proof of [1] only pointing out the main steps that need slight adjustments. We assume the reader is familiar with that proof and terminology used there.

Let x be a fractional solution to LP(2). We define W_x as the set of leaves that are (fractionally) cut off from r largely on low levels, i.e. there is high x-value on P_u on vertices far away from the root. We first start by recalling Theorem 12 from [1] which basically says that we can round an LP solution to an integral one by increasing the core budget B by a small constant such that W_x can be saved.

▶ **Theorem 7** (modified version of Theorem 12 in [1]). *Let $B \in \mathbb{R}_{\geq 1}$, $\mu \in (0,1]$, and $h = \lfloor \log_{1+\epsilon} L \rfloor$. Let $x \in$ LP(2) with value B and $supp(x) \subseteq V_{>h}$, and we define $W = \{u \in \Gamma | x(P_u) \geq \mu\}$. Then one can efficiently compute a set $R \subseteq V_{>h}$ such that:*

- *$R \cap P_u \neq \emptyset \quad \forall u \in W$, and*
- *There is an integral solution $z = y_1 + y_2$ to LP(2), which is a combination of two integral solutions y_1 and y_2 with value $B' = \frac{1}{\mu}B$ and 1 respectively such that $supp(y_1) \subseteq V_{>h}$ and $supp(y_2) \subseteq V_{\leq h}$.*

Proof. The proof would be very similar to the proof of Theorem 12 in [1], and the only difference is in providing the extra budget for protectecting the loose vertices in $V_{>h}$. They changed B to $B+1$ at level $h+1$ to provide this required budget. It that was enough, because the budget in the reduced instance is $B_{h+1} = 2^{h+1} \cdot B$ at this level, and so by this change $2^h = L$ many more firefighters are available and they are enough to protect all the loose vertices. But we need to change B to $B+1$ on all levels 1 to h, to have L many more firefighters for protecting all the loose vertices. This is because our budget in the reduced instance is $B_\ell = B$ when $1 \leq \ell < c$ and $B_\ell = m_\ell \cdot B$ when $c \leq \ell \leq L$. So by this change, we should have $c-1$ more firefighters in total for the first $c-1$ levels and $\sum_{\ell=c}^{h} m_\ell$ many more firefighters for levels c to h and the total would be $(1+\epsilon)^h = L$, which is enough to protect all the loose vertices. But the difference in our integral solution is that all the added budgets are from levels 1 to h (one for each level), and the remaining integral solution, which is $\frac{1}{\mu}$ feasible, is the subset of $V_{>h}$. This completes the proof of this theorem. ◀

Similar to [1], we consider two cases based on how B compares to $\log L$. When $B \geq \log L$, we will have a 3-approximation for the reduced instance, by first solving the LP(2). This is similar to Theorem 13 in [1] and consistent with our height reduction lemma:

▶ **Theorem 8** (modified version of Theorem 13 in [1]). *There is an efficient algorithm that computes a feasible solution to a compressed instance of RMFC with budget at most* $3B_{OPT}$ *when* $B \geq \log L$.

Proof. Assume x is a fractional LP(2) solution with value B. Then we use Theorem 7 and set $\mu = 1/2$ to obtain an integral solution z, which saves $W = \{u \in \Gamma | x(P_u) \geq \mu\}$, by core budget 1 at each level $1 \leq \ell \leq h$ and $2B$ at each level $h+1 \leq \ell \leq L$. Note that we can now transfer the 1 unit of budget from the very first level $\ell = 1$ to level $h+1$ and change the core budget $2B$ to $2B+1$ on this level and remove that extra budget from the very first level. This is because these extra firefighters from levels 1 to h are supposed to protect the loose vertices, which are in $V_{>h}$. By doing so we have an integral solution z such that the core budget is 0 in the first level, 1 in levels 2 to h, $2B+1$ at level $h+1$, and $2B$ at level $h+2$ to L. Now consider leaves $\Gamma \setminus W$. If we write another LP similar to LP(2), but specifically to save only these leaves by only protecting the vertices in $V_{\leq h}$, this LP would be feasible. Because all these vertices had $x(P_u) \cap V_{\leq h} \geq 0.5$, and so, $2x$ restricted to the vertices in $V_{\leq h}$, would be a feasible solution to this LP. Hence, we can find the optimal solution to this LP call it y. Based on Lemma 3, there would be at most $h = \log L$ many loose vertices all in $V_{\leq h}$, and so by adding $B > \log L = h$ many firefighters in the first level we would be able to protect all these y-loose vertices. Then all other remaining vertices could be saved by core budget $2B$. Putting these two solutions together (for saving W and $\Gamma \setminus W$) we have found an integral solution to save all the leaves, by having core budget $3B$ in the first level, $2B+1$ in levels 2 to $h+1$, and $2B$ at the remaining levels. This completes the proof of this theorem. ◀

We use the same terminology defined before Lemma 14 in [1], in particular for clean set pairs of vertices A, D. Suppose (A, D) is a clean pair compatible with OPT, i.e. $A \cup D \subseteq V_{\leq h}$, $A \subseteq OPT$ and $D \cap OPT = \emptyset$, for $h = \log \log L$ and $LP(A, D)$ by adding two sets of constraints to LP(2) to force the solution to pick all vertices in A and not picking all vertices in D as well as the vertices in their path to the root. Also for each fractional solution to this LP let $W_x = \left\{u \in \Gamma \middle| x(P_u \cap V_{>h}) \geq \frac{1}{1+\epsilon}\right\}$ to be the set of leaves cut off from the root by an x-load of at least $\mu = \frac{1}{1+\epsilon}$ within bottom levels (we changed $\frac{2}{3}$ to $1/(1+\epsilon)$ from [1]). For each $u \in \Gamma \setminus W_x$, let $f_u \in V_{\leq h}$ be the vertex closest to the root among all vertices in $(P_u \cap V_{\leq h}) \setminus D$, then define $F_x = \{f_u | u \in \Gamma \setminus W_x\} \setminus A$. It follows that no two vertices of F_x lie on the same leaf-root path. Furthermore, every leaf $u \in \Gamma \setminus W_x$ is part of the subtree T_f for precisely one $f \in F_x$. Also lets define $Q_x = V_{\leq h} \cap (\cup_{f \in F_x} T_f)$.

Now we are ready to provide our modification of Lemma 14 in [1] when $B < \log L$:

▶ **Lemma 9** (modified version of Lemma 14 in [1]). *Let* (A, D) *be a clean pair of vertices* (A, D)*, which is compatible with* OPT*, and let* x *and* y *be optimal solutions to* $LP(A, D)$ *and* $LP(A, V_{\leq h} \setminus A)$ *with objective function* B *and* $\hat{B}$ *respectively. Then, if* $OPT \cap Q_x = \emptyset$*, we have* $\hat{B} \leq (2+\epsilon)B_{OPT}$.

Proof. The proof is similar to the proof of Lemma 14 in [1] and the first difference is that we changed $\frac{2}{3}$ to $\frac{1}{1+\epsilon}$ in the definition of W_x. First of all we can have a fractional solution that saves W_x with picking only vertices from $V_{>h}$. This is because $(1+\epsilon)x$ restricted to levels $h+1$ to L would save W_x. Now partition $\Gamma \setminus W_x$ into two groups. The leaves that OPT cut them from the root by protecting a vertex in $V_{\leq h}$, denote them by W_1, and W_2

are the leaves that OPT is cutting them in levels $h+1$ to L. By finding such (A, D), we are actually saving W_1. and for W_2 there is an integral solution with core budget B_{OPT}, which is restricted to levels $h+1$ to L. So the optimum solution to $LP(A, V_{\leq h} \setminus A)$ would not use more than $(1+\epsilon)B_{OPT} + B_{OPT}$ as the core budget in levels $h+1$ to L. This completes the first part of lemma. To round this fractional solution to an integral one which saves W_x and W_2 (note that W_1 is saved already by the choice of A and D), we use the same technique as Theorem 7.

We need to first find an integral solution restricted to levels $h_1 = \log L$ to L that saves the leaves with $y(P_u \cap V_{>h_1}) \geq \frac{1}{2(1+\epsilon)}$ by adding one core budget to levels 1 to h_1 and then write another LP restricted to levels h to h_1. Then we find another integral solution restricted to levels h to h_1 by adding another core budget to levels 1 to h that saves all the remaining leaves, which for sure has $y(P_u \cap V_{>h} \cap V_{\leq h_1}) \geq \frac{1}{2(1+\epsilon)}$. Finally we would have an integral solution with core budget $B_{OPT} + 2$ for the first h levels, $2(2+\epsilon)B_{OPT} + 1$ for levels $h+1$ to h_1 and $2(2+\epsilon)B_{OPT}$ for levels h_1 to L. This completes the proof of this lemma. ◀

The only remaining thing is to show how we can find such (A, D) pair of vertices in polynomial time that follows in the exact same way of Lemma 15 in [1]. and the only difference is the running time, which is still polynomial. In their proof they have used the fact that for each leaf $u \in \Gamma \setminus W_x$, we have $x(P_u \cap V< h) > \frac{1}{3}$, and here we can say $x(P_u \cap V< h) > 1 - \frac{1}{1+\epsilon} > \epsilon$ that would only change the constant factor in the actual running time of $O(\log L)^{O(\log L)}$. So the total running time would be still polynomial. This means that we are able to find a $(5+\epsilon)$-approximation for the reduced instance of the RMFC problem, and then it leads to the $(5+\epsilon)$-approximation for the RMFC problem.

Streaming Complexity of Spanning Tree Computation

Yi-Jun Chang
ETH Zürich, Switzerland
yi-jun.chang@eth-its.ethz.ch

Martín Farach-Colton
Rutgers University, USA
farach@cs.rutgers.edu

Tsan-Sheng Hsu
Academia Sinica, Taipei City, Taiwan
tshsu@iis.sinica.edu.tw

Meng-Tsung Tsai
National Chiao Tung University, Hsinchu, Taiwan
mtsai@cs.nctu.edu.tw

Abstract

The semi-streaming model is a variant of the streaming model frequently used for the computation of graph problems. It allows the edges of an n-node input graph to be read sequentially in p passes using $\tilde{O}(n)$ space. If the list of edges includes deletions, then the model is called the turnstile model; otherwise it is called the insertion-only model. In both models, some graph problems, such as spanning trees, k-connectivity, densest subgraph, degeneracy, cut-sparsifier, and $(\Delta+1)$-coloring, can be exactly solved or $(1+\varepsilon)$-approximated in a single pass; while other graph problems, such as triangle detection and unweighted all-pairs shortest paths, are known to require $\tilde{\Omega}(n)$ passes to compute. For many fundamental graph problems, the tractability in these models is open. In this paper, we study the tractability of computing some standard spanning trees, including BFS, DFS, and maximum-leaf spanning trees.

Our results, in both the insertion-only and the turnstile models, are as follows.

Maximum-Leaf Spanning Trees: This problem is known to be APX-complete with inapproximability constant $\rho \in [245/244, 2)$. By constructing an *ε-MLST sparsifier*, we show that for every constant $\varepsilon > 0$, MLST can be approximated in a single pass to within a factor of $1+\varepsilon$ w.h.p. (albeit in super-polynomial time for $\varepsilon \leq \rho - 1$ assuming P $\neq$ NP) and can be approximated in polynomial time in a single pass to within a factor of $\rho_n + \varepsilon$ w.h.p., where ρ_n is the supremum constant that MLST cannot be approximated to within using polynomial time and $\tilde{O}(n)$ space. In the insertion-only model, these algorithms can be deterministic.

BFS Trees: It is known that BFS trees require $\omega(1)$ passes to compute, but the naïve approach needs $O(n)$ passes. We devise a new randomized algorithm that reduces the pass complexity to $O(\sqrt{n})$, and it offers a smooth tradeoff between pass complexity and space usage. This gives a polynomial separation between single-source and all-pairs shortest paths for unweighted graphs.

DFS Trees: It is unknown whether DFS trees require more than one pass. The current best algorithm by Khan and Mehta [STACS 2019] takes $\tilde{O}(h)$ passes, where h is the height of computed DFS trees. Note that h can be as large as $\Omega(m/n)$ for n-node m-edge graphs. Our contribution is twofold. First, we provide a simple alternative proof of this result, via a new connection to sparse certificates for k-node-connectivity. Second, we present a randomized algorithm that reduces the pass complexity to $O(\sqrt{n})$, and it also offers a smooth tradeoff between pass complexity and space usage.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Max-Leaf Spanning Trees, BFS Trees, DFS Trees

37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020).
Editors: Christophe Paul and Markus Bläser; Article No. 34; pp. 34:1–34:19

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

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.34

Related Version A full version of the paper is available at `https://arxiv.org/abs/2001.07672`.

Funding *Martín Farach-Colton*: This research was supported in part by NFS grants CSR-1938180, CCF-1715777, and CCF-1724745.
Tsan-Sheng Hsu: This research was supported in part by the Ministry of Science and Technology of Taiwan under contract MOST Grant 108-2221-E-001-011-MY3.
Meng-Tsung Tsai: This research was supported in part by the Ministry of Science and Technology of Taiwan under contract MOST grant 107-2218-E-009-026-MY3.

Acknowledgements We thank the anonymous reviewers for their helpful comments, and Eric Allender and Meng Li for their insightful discussions.

1 Introduction

Spanning trees are critical components of graph algorithms, from depth-first search trees (DFS) for finding articulation points and bridges [44], computing *st*-numbering [13], chain decomposition [41], and coloring signed graphs [18], to breadth-first search trees (BFS) for finding separators [33], computing sparse certificates of k-node-connectivity [8, 12], approximating diameters [10, 40], and characterizing AT-free graphs [5], and to maximum-leaf spanning trees (MLST) for connected dominating sets [35, 42] and connected maximum cuts [26, 21].

In the semi-streaming model, the tractability of spanning tree computation, except arbitrary spanning trees [3, 43, 39], is less studied. The *semi-streaming* model [37, 3] is a variation of streaming model frequently used for the computation of graph problems. It allows the edges of an n-node input graph to be read sequentially in p passes using $\tilde{O}(n)$[1] space. If the list of edges includes deletions, then the model is called the turnstile model; otherwise it is called the insertion-only model. In both models, some graph problems, such as spanning trees [3], k-connectivity [25], densest subgraph [36], degeneracy [15], cut-sparsifier [29], and $(\Delta+1)$-coloring [4], can be exactly solved or $(1+\varepsilon)$-approximated in a single pass, while other graph problems, such as triangle detection and unweighted all-pairs shortest paths [7], are known to require $\tilde{\Omega}(n)$ passes to compute. For many fundamental graph problems, e.g., standard spanning trees, the tractability in these models is open. BFS computation is known to require $\omega(1)$ passes [17], but only the naive $O(n)$-pass algorithm is known. It is unknown whether DFS computation requires more than one passes [14, 30], but the current best algorithm needs $\tilde{O}(h)$ passes [30] where h is the height of the computed DFS trees, so $h = O(n)$ for dense graphs. The tractability of maximum-leaf spanning trees (MLST) is unknown even allowing $O(n^2)$ space, since it is APX-complete [34, 20].

Due to the lack of efficient streaming algorithms for spanning tree computation, for some graph problems that are traditionally solved using spanning trees, such as finding articulation points and bridges, people had to look for alternative methods when designing streaming algorithms for these problems [16, 14]. The alternative methods, even if they are based on known results in graph theory, may still involve the design of new streaming algorithms. For the problems mentioned above, the alternative methods use newly-designed sparse connectivity certificates [12, 25] that are easily computable in the semi-streaming

[1] We write $\tilde{O}(k)$ to denote $O(k \operatorname{poly}\log n)$ or $O(k/\operatorname{poly}\log n)$ where n is the number of nodes in the input graph. Similarly, $\tilde{\Omega}(k)$ denotes $\Omega(k \operatorname{poly}\log n)$ or $\Omega(k/\operatorname{poly}\log n)$.

model, rather than the classical one due to Nagamochi and Ibaraki [38]. Hence establishing the hardness of spanning tree computation helps to explain the need of the alternative methods.

In this paper, we study the tractability of computing standard spanning trees for connected simple undirected graphs, including BFS trees, DFS trees, and MLST. Unless otherwise stated, our upper bounds work in the turnstile model (and hence also in the insertion-only model), and our lower bounds hold for the insertion-only model (and hence also in the turnstile model). The space upper and lower bounds are in bits. Our results are as follows.

Maximum-Leaf Spanning Trees: We show, by constructing an *ε-MLST sparsifier* (Theorem 6), that for every constant $\varepsilon > 0$, MLST can be approximated in a single pass to within a factor of $1+\varepsilon$ w.h.p.[2] (albeit in super-polynomial time for $\varepsilon \leq \rho - 1$ since it is APX-complete [34, 20] with inapproximability constant $\rho \in [245/244, 2)$ [9]) and can be approximated in polynomial time in a single pass to within a factor of $\rho_n + \varepsilon$ w.h.p., where ρ_n is the supremum constant that MLST cannot be approximated to within using polynomial time and $\tilde{O}(n)$ space. In the insertion-only model, these algorithms are deterministic. We also show a complementary hardness result (Theorem 17) that for every $k \in [1, (n-5)/4]$, to approximate MLST to within an additive error k, any single-pass randomized streaming algorithm that succeeds with probability at least $2/3$ requires $\Omega(n^2/k^2)$ bits. This hardness result excludes the possibility to have a single-pass semi-streaming algorithm to approximate MLST to within an additive error $n^{1/2-\Omega(1)}$. Our results for MLST shows that intractability in the sequential computation model (i.e., Turing machine) does not imply intractability in the semi-streaming model.

Our algorithms rely on a new sparse certificate, the *ε-MLST sparsifier*, defined as follows. Let G be an n-node m-edge connected simple undirected graph. Then for any given constant $\varepsilon > 0$, H is an *ε-MLST sparsifier* if it is a connected spanning subgraph of G with $|E(H)| \leq f(\varepsilon)|V(G)|$ and $\mathrm{leaf}(H) \geq (1-\varepsilon)\,\mathrm{leaf}(G)$, where $\mathrm{leaf}(G)$ denotes the maximum number of leaves (i.e. nodes of degree one) that any spanning tree of G can have and f is some function independent of n. We show that an ε-MLST sparsifier can be constructed efficiently in the semi-streaming model.

▶ **Theorem 1.** *In the turnstile model, for every constant $\varepsilon > 0$, there exists a randomized algorithm that can find an ε-MLST sparsifier with probability $1 - 1/n^{\Omega(1)}$ using a single pass, $\tilde{O}(f(\varepsilon)n)$ space, and $\tilde{O}(n+m)$ time, and in the insertion-only model a deterministic algorithm that uses a single pass, $\tilde{O}(f(\varepsilon)n)$ space, and $O(n+m)$ time.*

Combining Theorem 1 with any polynomial-time RAM algorithms for MLST that uses $\tilde{O}(n+m)$ space, e.g, [34, 35, 42], we obtain the following result.

▶ **Corollary 2.** *In the turnstile model, for every constant $\varepsilon > 0$, there exists a randomized algorithm that can approximate MLST for any n-node connected simple undirected graph with probability $1 - 1/n^{\Omega(1)}$ to within a factor of $\rho_n + \varepsilon$ using a single pass, $\tilde{O}(f(\varepsilon)n)$ space, and polynomial time, where ρ_n is the supremum constant that MLST cannot be approximated to within using polynomial time and $\tilde{O}(n)$ space, and in the insertion-only model a deterministic algorithm that uses a single pass, $\tilde{O}(f(\varepsilon)n)$ space, and polynomial time.*

Using Corollary 2, we show that approximate connected maximum cut can be computed in a single pass using $\tilde{O}(n)$ space for unweighted regular graphs (Corollary 7).

[2] W.h.p. means with probability $1 - 1/n^{\Omega(1)}$.

BFS Trees: It is known that BFS trees require $\omega(1)$ passes to compute [17], but the naive approach needs $O(n)$ passes. We devise a randomized algorithm that reduces the pass complexity to $O(\sqrt{n})$ w.h.p., and give a smooth tradeoff between pass complexity and space usage.

▶ **Theorem 3.** *In the turnstile model, for each $p \in [1, \sqrt{n}]$, there exists a randomized algorithm that can compute a BFS tree for any n-node connected simple undirected graph with probability $1 - 1/n^{\Omega(1)}$ in p passes using $\tilde{O}((n/p)^2)$ space, and in the insertion-only model a deterministic algorithm that uses $\tilde{O}(n^2/p)$ space.*

This gives a polynomial separation between single-source and all-pairs shortest paths for unweighted graphs because any randomized semi-streaming algorithm that computes unweighted all-pairs shortest paths with probability at least $2/3$ requires $\tilde{\Omega}(n)$ passes.

We extend Theorem 3 and obtain that multiple BFS trees, each starting from a unique source node, can be computed more efficiently in pass complexity in a batch than individually (see Theorem 13). We show that this batched BFS has applications to computing a 1.5-approximation of diameters for unweighted graphs (Theorem 15) and a 2-approximation of Steiner trees for unweighted graphs (Corollary 14).

DFS Trees: It is unknown whether DFS trees require more than one passes [14, 30], but the current best algorithm needs $\tilde{O}(h)$ passes due to Khan and Mehta [30], where h is the height of computed DFS trees. We devise a randomized algorithm that has pass complexity $O(\sqrt{n})$ w.h.p., and give a smooth tradeoff between pass complexity and space usage.

▶ **Theorem 4.** *In the turnstile model, for each $p \in [1, \sqrt{n}]$, there exists a randomized algorithm that can compute a DFS tree for any n-node connected simple undirected graph with probability $1 - 1/n^{\Omega(1)}$ in p passes that uses $\tilde{O}(n^3/p^4)$ space, and in the insertion-only model a deterministic algorithm that uses $\tilde{O}(n^2/p^2)$ space.*

For dense graphs, our algorithms improves upon the current best algorithms for DFS due to Khan and Mehta [30] which needs $\Omega(m/n)$ passes for n-node m-edge graphs in the worst case because of the existence of (m/n)-cores, where a k-core is a maximal connected subgraph in which every node has at least k neighboring nodes in the subgraph.

1.1 Technical Overview

Maximum-Leaf Spanning Trees: We construct an ε-MLST sparsifier by a new result that complements Kleitman and West's lower bounds on the maximum number of leaves for graphs with minimum degree $\delta \geq 3$ [31]. The lower bounds are: if a connected simple undirected graph G has minimum degree δ for some sufficiently large δ, then $\mathrm{leaf}(G) \geq (1 - (2.5 \ln \delta)/\delta)|V(G)|$ and the leading constant can be larger for $\delta \in \{3, 4\}$. Our complementary result (Lemma 5), without the restriction on the minimum degree, is: any connected simple undirected graph G, except the singleton graph, has

$$\mathrm{leaf}(G) \geq \frac{1}{10}(|V(G)| - \mathrm{inode}(G)), \tag{1}$$

where $\mathrm{inode}(G)$ denotes the number of nodes whose degree is two and whose neighbors both have degree two. Equation (1) implies that, if one can find a connected spanning subgraph H of G so that $|\mathrm{leaf}(G) - \mathrm{leaf}(H)| \leq \varepsilon(V(G) - \mathrm{inode}(G))$, then one gets an (10ε)-MLST sparsifier.

Our sparsification technique is general enough to obtain a $(t+\varepsilon)$-approximation for MLST in a single pass using $\tilde{O}(n)$ space by combining any t-approximation $\tilde{O}(n)$-space RAM algorithm for MLST with our ε-MLST sparsifier. On the other hand, since in linear time one can find an ε-MLST sparsifier of $O(n)$ edges, any t-approximation RAM algorithm for MLST with time complexity $O(f(n,m))$ can be reduced to $O(f(n,n)+n+m)$ if a small sacrifice on approximation ratio is allowed. This reduces the time complexity of RAM algorithms for MLST that need superlinear time on the number of edges, such as the local search approach from $O(m^k n^{k+2})$ for $k \geq 1$ to $O(n^{2k+2})$ and the leafy forest approach from $O((m+n)\alpha(n))$ to $O(m+n\alpha(n))$, both due to Lu and Ravi [34, 35].

BFS Trees: We present a simple deterministic algorithm attaining a smooth tradeoff between pass complexity and space usage. In particular, in the insertion-only semi-streaming model, the algorithm finishes in $O(n/\text{poly}\log n)$ passes. The algorithm is based on an observation that the sum of degrees of nodes in any root-to-leaf path of a BFS tree is bounded by $O(n)$ (Lemma 8).

Our more efficient randomized algorithm (Theorem 3) constructs a BFS tree by combining the results of multiple instances of bounded-radius BFS. To reduce the space usage, the simulation of these bounded-radius BFS are assigned random starting times, and the algorithm only maintains the last three layers of each BFS tree. These ideas are borrowed from results on shortest paths computation in the parallel and the distributed settings [11, 22, 27, 45].

DFS Trees: We present a simple alternative proof of the result of Khan and Mehta [30] that a DFS tree can be constructed in $\lceil h/k \rceil$ passes using $\tilde{O}(nk)$ space, for any given parameter k, where h is the height of the computed DFS tree. The new proof is based on the following connection between the DFS computation and the sparse certificates for k-node-connectivity. We show in Lemma 16 that the first k layers of *any* DFS tree of a such a certificate H can be extended to a DFS tree of the original graph G.

The proof of Theorem 4 is based on the parallel DFS algorithm of Aggarwal and Anderson [2]. In this paper we provide an efficient implementation of their algorithm in the streaming model, also via the sparse certificates for k-node-connectivity, which allows us to reduce the number of passes by batch processing.

We note that in a related work, Ghaffari and Parter [23] showed that the parallel DFS algorithm of Aggarwal and Anderson can be adapted to distributed setting. Specifically, they showed that DFS can be computed in the CONGEST model in $\tilde{O}(\sqrt{Dn}+n^{3/4})$ rounds, where D is the diameter of the graph.

1.2 Paper Organization

In Section 2, we present how to construct an ε-MLST sparsifier and apply it to devise single-pass semi-streaming algorithms to approximate MLST to within a factor of $(1+\varepsilon)$ for every constant $\varepsilon > 0$. Then, in Section 3, we show how to compute a BFS tree rooted at a given node by an $O(\sqrt{n})$-pass $\tilde{O}(n)$-space algorithm w.h.p. and its applications to computing approximate diameters and approximate Steiner trees. In Section 4, we have a similar result for computing DFS trees; that is, $O(\sqrt{n})$-pass $\tilde{O}(n)$-space algorithm that succeeds w.h.p. Lastly, we prove the claimed single-pass lower bound in Section 5.

2 Maximum-Leaf Spanning Trees

In this section, we will show how to construct an ε-MLST sparsifier in the semi-streaming model; that is, proving Theorem 1. We recall the notions defined in Section 1 before proceeding to the results. By *ignorable node*, we denote a node x whose degree is two and whose neighbors u and v have degree two as well. Note that $u \neq v$ for simple graphs. Let $\mathrm{leaf}(G)$ be the maximum number of leaves (i.e. nodes of degree one) that a spanning tree of G can have. Let $\mathrm{inode}(G)$ denote the number of ignorable nodes in G. Let $\deg_G(x)$ denote the degree of node x in graph G. Let $S_k(G)$ denote any subgraph of G so that $S_k(G)$ contains all nodes in G and every node x in $S_k(G)$ has degree $\deg_{S_k}(x) \geq \min\{\deg_G(x), k\}$. Let $T(G)$ be any spanning tree of a connected graph G.

We begin with a result that complements Kleitman and West's lower bounds on the number of leaves for graphs with minimum degree δ for any $\delta \geq 3$. Our lower bound does not rely on the degree constraint. The constant $1/10$ in Lemma 5 may be improved, but the subsequent lemmata and theorems only require it to be $\Omega(1)$.

▶ **Lemma 5.** *Every connected simple undirected graph G, except the singleton graph, has*

$$\mathrm{leaf}(G) \geq \frac{1}{10}(|V(G)| - \mathrm{inode}(G)).$$

Proof. Our proof is a generalization of the dead leaf argument due to Kleitman and West [31]. Let T be a tree rooted at s with $N(s)$ as leaves for some arbitrary node $s \in G$ initially, where $N(s)$ denotes the neighbors of s, and then grow T iteratively by a node expansion order, defined below. By expanding T at node x, we mean to select a leaf node x of T and add all of x's neighbors in $G \backslash T$, say $y_1, y_2, \ldots, y_d$, and their connecting edges, $(x, y_1), (x, y_2), \ldots, (x, y_d)$, to T. In this way, every node outside T cannot be a neighbor of any non-leaf node in T. We say a leaf node in T is *dead* if it has no neighbor in $G \setminus T$. Let $(\Delta n)_i$ denote the number of non-ignorable nodes in G that joins T while the i-th operation is applied. Let $(\Delta \ell)_i$ denote the change of the number of leaf nodes in T while the i-th operation is applied. Let $(\Delta m)_i$ denote the change of the number of dead leaf nodes in T while the i-th operation is applied. The subscript i may be removed when the context is clear. We need to secure that $\Delta \ell + \Delta m \geq \Delta n/5$ holds for each of the following operations and the initial operation.

Operation 1: If T has a leaf node x that has $d \geq 2$ neighbors outside T, then expand T at x. In this case, $\Delta n \leq d$, $\Delta \ell \geq d - 1$, and $\Delta m \geq 0$.

Operation 2: If every leaf node in T has at most one neighbor outside T and some node $x \notin T$ has $d \geq 2$ neighbors in T, then expand T at one of x's neighbors in T. In this case, $\Delta n \leq 1$, $\Delta \ell = 0$, and $\Delta m = d - 1$.

Operation 3: This operation is used only when the previous two operations do not apply. Let x_0 be some leaf in T that has exactly one neighbor x_1 not in T. For each $i \geq 1$, if x_i is defined and all neighbors of x_i other than x_{i-1} are outside T and x_i has degree two in G, then define x_{i+1} to be the neighbor of x_i other than x_{i-1}. Suppose that x_i for $i \leq k$ are defined and x_{k+1} is not defined, then we expand T at x_i for each $i \leq k$ in order. Though k can be arbitrarily large, $\Delta n \leq 2 + \deg_G(x_k)$. If x_{k+1} is not defined and x_k has $d > 0$ neighbors other than x_{k-1} in T (thus $k \geq 2$ in this case otherwise Operation 2 applies), then we discuss in subcases:

Subcase 1 ($\deg_G(x_k) = 1$): It is impossible to have $\deg_G(x_k) = 1$ for this case.
Subcase 2 ($\deg_G(x_k) = 2$): Then $\Delta \ell = 0$ and $\Delta m = 2$.
Subcase 3 ($\deg_G(x_k) \geq 3$): Then $\Delta \ell = \deg_G(x_k) - d - 2$ and $\Delta m \geq d$.

If x_{k+1} is not defined and x_k has 0 neighbor other than x_{k-1} in T, then $\deg_G(x_k)$ is either 1 or ≥ 3. For $\deg_G(x_k) = 1$, $\Delta\ell = 0$ and $\Delta m = 1$. For $\deg_G(x_k) \geq 3$, $\Delta\ell = \deg_G(x_k) - 2$ and $\Delta m \geq 0$.

It is clear that one can expand T to get a spanning tree of G by a sequence of the above operations. Because all leaves are eventually dead, $\sum \Delta m = \sum \Delta\ell$. Consequently, $2\,\mathrm{leaf}(G) \geq 2\sum \Delta\ell = \sum \Delta\ell + \Delta m \geq (\sum \Delta n)/5 = (V(G) - \mathrm{inode}\,G)/5$, as desired. ◀

Given Lemma 5, our goal is, for every constant $\varepsilon > 0$, find a sparse subgraph H of the input graph G so that:

1. The nodes incident to the edges in $T^* \setminus H$ can be *dominated* by a small set S of at most $\varepsilon(|V(G)| - \mathrm{inode}(G))$ nodes, i.e. either in S or has at least one neighbor node in S using the edges in H, where T^* is any optimal MLST of G.
2. H is connected.

Because of the existence of the small dominating set S, one can obtain a forest F from $T^* \cap H$ by adding some edges in H so that the number of leaves in F is no less than that in T^* by $|S|$ and the number of connected components in F is no more than that in T^* by $|S|$. Since H is connected, one can further obtain a spanning tree T from F by adding at most $|S|$ edges in H, so the number of leaves in T is no less than that in F by $2|S|$. Pick an H associated with a sufficiently small ε, by Equation (1) H is an ε-MLST sparsifier. A formal proof is given below.

▶ **Theorem 6.** *For every integer $k \geq 186$, every connected simple undirected graph G has*

$$\mathrm{leaf}(S_k(G) \cup T(G)) \geq \left(1 - 30\left(\frac{1 + \ln(k+1)}{k+1}\right)\right)\mathrm{leaf}(G).$$

Proof. Let T^* be a spanning tree of G that has $\mathrm{leaf}(G)$ leaves. Let k be some fixed integer at least 3 and let $H = S_k(G) \cup T(G)$. Let $L = \{x \in V(G) : x \text{ is incident to some } e \in T^* \setminus H\}$. Note that every node $x \in L$ has $\deg_G(x) > k$, so x and all neighbors of x are not ignorable nodes in G.

First, we show that L can be dominated by a small set S of size at most $\varepsilon(|V(G)| - \mathrm{inode}(G))$ using some edges in H. We obtain S from two parts, S_1 and S_2. S_1 is a random node subset sampled from the non-ignorable nodes in G, in which each node is included in S_1 with probability p independently, for some $p \in (0, 1)$ to be determined later. Thus, $E[|S_1|] = p(|V(G)| - \mathrm{inode}(G))$. Since every node $x \in L$ is adjacent only to the non-ignorable nodes in G, the probability that $x \in L$ is not dominated by any node in S_1 is

$$\Pr[x \text{ is not dominated}] = (1-p)^{1+\deg_H(x)} \leq (1-p)^{k+1}.$$

Let S_2 be the set of nodes in L that are not dominated by any node in S_1 using the edges in H. Thus,

$$E[|S|] = E[|S_1| + |S_2|] \leq \left(p + (1-p)^{k+1}\right)(|V(G)| - \mathrm{inode}(G)).$$

Then, we obtain a forest F from $T^* \cap H$ by adding some edges in H as follows. Initially, $F = T^* \cap H$.

Operation 1: For each $x \in L$, if x is an isolated node in $T^* \cap H$ and $x \notin S$, then add an edge e from x to some node in S to F. Such an edge e must exist because S dominates L.

Operation 2: For each $x \in L$, if x is not an isolated node in $T^* \cap H$ and the connected component that contains x has an empty intersection with S, then add an edge e from x to some node in S to F. Again, such an edge e must exist because S dominates L.

For each leaf $\ell \in T^*$, if $\deg_G(\ell) \leq k$, then ℓ is a leaf in $T^* \cap H$ (also in F unless $\ell \in S$); otherwise $\deg_G(\ell) > k$, if ℓ is not a leaf in $T^* \cap H$, then ℓ must be an isolated node in $T^* \cap H$, and by Operation 1 ℓ is connected to some node in S unless $\ell \in S$. Hence, except those in S, every ℓ is a leaf node in F, so the number of leaves in F is no less than that in T^* by $|S|$. By Operation 2, the number of connected component is at most $|S|$.

Lastly, since H is connected, one can obtain a spanning tree T from F by connecting the components in F by some edges in H. Thus, the number of leaves in T is no less than that in T^* by $3|S|$. To obtain an ε-MLST sparsifier, by Lemma 5 we need:

$$\frac{3|S|}{\frac{1}{10}(|V(G)| - \mathrm{inode}(G))} \leq 30\left(p + (1-p)^{k+1}\right) \leq 30\left(p + e^{-p(k+1)}\right) \leq \varepsilon$$

Setting $p = (\ln(k+1))/(k+1)$ gives the desired bound, and the leading constant is positive for $k \geq 186$. ◀

To find such a subgraph H, fetching a spanning tree of the input graph G and grabbing k edges for each node in G suffices. Thus, we get a single-pass $\tilde{O}(n)$-space algorithm for the insertion-only model. As for the turnstile model, we use $\tilde{O}(k)$ ℓ_0-samplers [28] for each node in G to fetch at least k neighbors of x w.h.p., and fetch a spanning tree by appealing to the single-pass $\tilde{O}(n)$-space algorithm for spanning trees in dynamic streams [3]. This gives a proof of Theorem 1.

Applications. In [21], Gandhi et al. show a connection between the maximum-leaf spanning trees and connected maximum cut. Their results imply that, for any unweighted regular graph G, the connected maximum cut can be found by the following two steps:

Step 1: Find a spanning tree T whose $\mathrm{leaf}(T) \geq (1/2 - \varepsilon)\,\mathrm{leaf}(G)$ for some constant $\varepsilon > 0$.
Step 2: Randomly partition the leaves in T into two parts L and R so that each leaf is included in L with probability $1/2$ independently.

Then, outputting L and $V(G) \setminus L$ yields an $8 + \varepsilon$-approximation for connected maximum cut. Step 1 is the bottleneck and can be implemented by combining our ε-MLST sparsifier (Theorem 1) with the 2-approximation algorithm for MLST due to Solis-Oba, Bonsma, and Lowski [42]. This gives Corollary 7.

▶ **Corollary 7.** *In the turnstile model, for every constant $\varepsilon > 0$, there exists a randomized algorithm that can approximate the connected maximum cut for n-node unweighted regular graphs to within a factor of $8 + \varepsilon$ with probability $1 - 1/n^{\Omega(1)}$ in a single pass using $\tilde{O}(f(\varepsilon)n)$ space.*

3 Breadth-First Search Trees

A BFS tree of an n-node connected simple undirected graph can be constructed in $O(n)$ passes using $\tilde{O}(n)$ space by simulating the standard BFS algorithm layer by layer. By storing the entire graph, a BFS tree can be computed in a single pass using $O(n^2)$ space. In Section 3.1, we show that it is possible to have a smooth tradeoff between pass complexity and space usage. In Section 3.2, we prove Theorem 3, which shows that the above tradeoff can be

improved when randomness is allowed, even in the turnstile model. Then, in Section 3.3, we show that multiple BFS trees, each starting from a distinct source node, can be computed more efficiently in a batch than individually. Lastly, we demonstrate an application to diameter approximation in Section 3.4.

In the BFS problem, we are given an n-node connected simple undirected graph $G = (V, E)$ and a distinguished node s, and it suffices to compute the distance $\mathrm{dist}(s, v)$ for each node $v \in V \setminus \{s\}$. To infer a BFS tree from the distance information $\{\mathrm{dist}(s, v) \colon v \in V\}$, it suffices to assign a parent to each node $v \in V \setminus \{s\}$ the smallest-identifier node from the set $\{u \in N(v) \colon \mathrm{dist}(s, u) = \mathrm{dist}(s, v) - 1\}$ where $N(v)$ is the set of v's neighbors. This can be done with one additional pass using $\tilde{O}(n)$ space in the insertion-only model. In the turnstile model, for p-pass streaming algorithms with $p > \log n$, this can be done with $O(\log n / \log\log n)$ additional passes w.h.p. using $O(\log n)$ ℓ_0-samplers [28] for each node $v \in V \setminus \{s\}$, and this costs $\tilde{O}(n)$ space. For $p \leq \log n$, the space bound is $\tilde{O}(n^2)$ and one can use $\tilde{O}(n)$ ℓ_0-samplers for each node, so this step can be done in one additional pass. Hence in the subsequent discussion we focus on computing the distance from s to each node $v \in V \setminus \{s\}$.

3.1 A Simple Deterministic Algorithm

We present a simple deterministic p-pass $\tilde{O}(n^2/p)$-space algorithm in the insertion-only model by an observation that every root-to-leaf path in a BFS tree cannot visit too many high-degree nodes (Lemma 8). Then, one can simulate the standard BFS algorithm efficiently layer-by-layer over high-degree nodes (Theorem 9).

▶ **Lemma 8.** *Let P be a root-to-leaf path in some BFS tree of an n-node connected simple undirected graph G. Then*

$$\sum_{x \in P} \deg_G(x) \leq 3n = O(n)$$

where $\deg_G(x)$ denotes the degree of x in G.

Proof. Suppose $P = x_1 x_2 \cdots x_k$ comprises k nodes. Observe that if x_i and x_j have $i \equiv j \pmod 3$, then x_i and x_j cannot share any neighbor node; otherwise P can be shorten, a contradiction. Thus, for each $c \in \{0, 1, 2\}$ the total contribution of all x_i's whose $i \equiv c \pmod 3$ to $\sum_{x_i \in P} \deg_G(x_i)$ is $O(n)$. Summing over all possible c gives the bound. ◀

We note that Lemma 8 is near-optimal. To see why, let $H = (V, E)$ where V is the union of disjoint sets $V_0, V_1, \ldots, V_k$ and $E = \{(x, y) : x \in V_i \text{ and } y \in V_j \text{ for any } i, j \text{ that } |i - j| \leq 1\}$. By setting $k = \lceil (n-1)/t \rceil$ for some parameter t, $|V_0| = 1$, $|V_i| = t$ for every $i \in [1, k-1]$, and $1 \leq |V_k| \leq t$, any BFS tree rooted at the node in V_0 has a root-to-leaf path Q of length k, and each node in $Q \cap (V_2 \cup V_3 \cup \ldots \cup V_{k-2})$ has degree $3t - 1$. Pick any t such that $k = \omega(1)$ and $t = \omega(1)$. We have $\sum_{x \in Q} \deg_H(x) = (3 - o(1))n$.

▶ **Theorem 9.** *Given an n-node connected simple undirected graph G with a distinguished node s, a BFS tree rooted at s can be found deterministically in p passes using $\tilde{O}(n^2/p)$ space for every $p \in [1, n]$ in the insertion-only model.*

Proof. Given a parameter k, our algorithm goes as follows. In the first pass, keep arbitrary n/k neighbors for each node $v \in G$ in memory and then use the in-memory edges to update the distance $\mathrm{dist}(s, v)$ for each $v \in G$ by any single-source shortest path algorithm. The set of the in-memory edges is an invariant after the first pass. Hence, the memory usage is

$\tilde{O}(n^2/k)$. Then, in each of the subsequent passes, processing the edges (u, v) in the stream one by one, without keeping them in memory after the processing, if $\mathrm{dist}(s, u) + 1 < \mathrm{dist}(s, v)$ (resp. if $\mathrm{dist}(s, v) + 1 < \mathrm{dist}(s, u)$), then update $\mathrm{dist}(s, v)$ (resp. $\mathrm{dist}(s, u)$). After the edges in the stream are all processed, use the in-memory edges to update the distance $\mathrm{dist}(s, v)$ for each $v \in G$ again by any single-source shortest path algorithm but with initial distances. Our algorithm repeats until no distance has been updated in a single pass.

Observe a root-to-leaf path $P = sz_1z_2 \cdots z_t$ in some BFS tree rooted at s. Suppose P contains exactly ℓ edges that appears only on tape, let them be $(z_{x_1}, z_{y_1}), \ldots, (z_{x_\ell}, z_{y_\ell})$ where $1 \leq x_i < y_i \leq x_{i+1} < y_{i+1} \leq t$ for every $i \in [1, \ell - 1]$. Let $\mathrm{pred}_P(z_i)$ be the predecessor of z_i on P that is closest to z_i among nodes in $\{s\} \cup \{z_{y_j} : y_j < i\}$. By the definition of the above construction, it is assured that $\deg(z_{x_i}) \geq n/k$ for each $i \in [1, \ell]$. Thus by Lemma 8, $\ell = O(k)$. Then we appeal to the argument used for the analysis of Bellman-Ford algorithm [19, 6]. For every $i \in [1, t]$, if $i \notin \{y_1, y_2, \ldots, y_\ell\}$, $\mathrm{dist}(s, z_i)$ attains the minimum possible value at the same pass when $\mathrm{dist}(s, \mathrm{pred}_P(z_i))$ attains; otherwise $i = y_j$ for some $j \in [1, \ell]$, $\mathrm{dist}(s, y_j)$ attains the minimum possible value at most one pass after $\mathrm{dist}(s, x_j)$ attains. Hence, $O(k)$ passes suffices to compute $\mathrm{dist}(s, z_i)$ for all $i \in [1, t]$ and this argument applies to all root-to-leaf paths. Setting $k = p$ yields the desired bound. ◀

3.2 A More Efficient Randomized Algorithm

In this section, we prove Theorem 3. Our BFS algorithm is based on the following generic framework, which has been applied to finding shortest paths in the parallel and the distributed settings [11, 22, 27, 45]. Sample a set U of approximately k distinguished nodes such that each node $v \neq s$ joins U independently with probability k/n, and $s \in U$ with probability 1. By a Chernoff bound, $|U| = \tilde{\Theta}(k)$ with high probability. We will grow a local BFS tree of radius $\tilde{O}(n/k)$ from each node in U, and then we will construct the final BFS tree by combining them. We will rely on the following lemma, which first appeared in [45].

▶ **Lemma 10** ([45]). *Let s be a specified source node. Let U be a subset of nodes such that each node $v \neq s$ joins U with probability k/n, and s joins U with probability 1. For any given parameter $C \geq 1$, the following holds with probability $1 - n^{-\Omega(C)}$. For each node $t \neq s$, there is an s-t shortest path $P_{s,t}$ such that each of its $C(n \log n)/k$-node subpath P' satisfies $P' \cap U \neq \emptyset$.*

For notational simplicity, in subsequent discussion we write $h = C(n \log n)/k - 1 = \tilde{O}(n/k)$. Lemma 10 shows that for each node $t \in V \setminus \{s\}$,

$$\mathrm{dist}(s, t) = \min_{u \in U \cap N^h(t)} \mathrm{dist}(s, u) + \mathrm{dist}(u, t) \tag{2}$$

with probability $1 - n^{-\Omega(C)}$ where $N^h(v) = \{u \colon \mathrm{dist}(u, v) \leq h\}$.

To see this, consider the s-t shortest path $P_{s,t}$ specified in Lemma 10. If the number of nodes in $P_{s,t}$ is less than h, then the above claim holds because $s \in U \cap N^h(t)$. Otherwise, Lemma 10 guarantees that there is a node $u \in P_{s,t} \cap U \cap N^h(t)$ with probability $1 - n^{-\Omega(C)}$. Using Equation (2), a BFS tree can be computed using the following steps.

1. Compute $\mathrm{dist}(u, v)$ for each $u \in U$ and $v \in U \cap N^h(u)$. Using this information, we can infer $\mathrm{dist}(s, u)$ for each $u \in U$.
2. Compute $\mathrm{dist}(s, t)$ for each $t \in V \setminus \{s\}$ by the formula $\mathrm{dist}(s, t) = \min_{u \in U \cap N^h(t)} \mathrm{dist}(s, u) + \mathrm{dist}(u, t)$.

In what follows, we show how to implement the above two steps in the streaming model, using $\tilde{O}(n + k^2)$ space and $\tilde{O}(n/k)$ passes. By a change of parameter $p = \tilde{O}(n/k)$, we obtain Theorem 3.

Step 1. To compute $\text{dist}(u,v)$ for each $u \in U$ and $v \in U \cap N^h(u)$, we let each $u \in U$ initiate a radius-h local BFS rooted at u. A straightforward implementation of this approach in the streaming model costs $h = \tilde{O}(n/k)$ passes and $O(n \cdot |U|) = \tilde{O}(nk)$ space, since we need to maintain $|U|$ search trees simultaneously.

We show that the space requirement can be improved to $\tilde{O}(n + k^2)$. Since we only need to learn the distances between nodes in U, we are allowed to forget distance information associated with nodes $v \notin U$ when it is no longer needed. Specifically, suppose we start the BFS computation rooted at $u \in U$ at the τ_uth pass, where τ_u is some number to be determined. For each $0 \le i \le h-1$, the induction hypothesis specifies that at the beginning of the $(\tau_u + i)$th pass, all nodes in $L_i(u) = \{v \in V \colon \text{dist}(u,v) = i\}$ have learned that $\text{dist}(u,v) = i$. During the $(\tau_u + i)$th pass, for each node $v \in V$ with $\text{dist}(u,v) > i$, we check if v has a neighbor in $L_i(u)$. If so, then we learn that $\text{dist}(u,v) = i+1$.

In the above BFS algorithm, if $\text{dist}(u,v) = i$ for some $0 \le i \le h-1$, then we learn the fact that $\text{dist}(u,v) = i$ during the $(\tau_u + i - 1)$th pass. Observe that such information is only needed during the next two passes. After the end of the $(\tau_u + i + 1)$th pass, for each $v \in V$ with $\text{dist}(u,v) = i$, we are allowed to forget that $\text{dist}(u,v) = i$. That is, v only needs to participate in the BFS computation rooted at u during these three passes $\{\tau_u + i - 1,\ \tau_u + i,\ \tau_u + i + 1\}$.

For each $u \in U$, we assign the starting time τ_u independently and uniformly at random from $\{1, 2, \ldots, h\}$. Lemma 11 shows that for each node $v \in V$ and for each pass $1 \le t \le 2h-1$, the number of BFS computations that involve v is $\tilde{O}(1)$. The idea of using random starting time to schedule multiple algorithms to minimize congestion can be traced back from [32]. Note that $\tau_u + \text{dist}(u,v) - 1 \le t \le \tau_u + \text{dist}(u,v) + 1$ is the criterion for v to participate in the BFS rooted at u during the tth pass.

▶ **Lemma 11.** *For each node v, and for each integer $1 \le t \le 2h-1$, with high probability, the number of nodes $u \in U$ such that $\tau_u + \text{dist}(u,v) - 1 \le t \le \tau_u + \text{dist}(u,v) + 1$ is at most $O(\max\{\log n, |U|/h\})$.*

Proof. Given two nodes $u \in U$ and $v \in V$, and a fixed number t, the probability that $\tau_u + \text{dist}(u,v) - 1 \le t \le \tau_u + \text{dist}(u,v) + 1$ is at most $3/h$. Let X be the total number of $u \in U$ such that $\tau_u + \text{dist}(u,v) - 1 \le t \le \tau_u + \text{dist}(u,v) + 1$. The expected value of X can be upper bounded by $\mu = |U| \cdot (3/h)$. By a Chernoff bound, with high probability, $X = O(\max\{\log n, |U|/h\})$. ◀

Recall that $|U| = \tilde{O}(k)$ with high probability, and $h = \tilde{O}(n/k)$. By Lemma 11, we only need $\lceil k^2/n \rceil \cdot \tilde{O}(1)$ space per each $v \in V$ to do the radius-h BFS computation from all nodes $u \in U$. That is, the space complexity is $\tilde{O}(n + k^2)$. To store the distance information $\text{dist}(u,v)$ for each $u \in U$ and $v \in U \cap N^h(u)$, we need $\tilde{O}(k^2)$ space. Thus, the algorithm for Step 1 costs $\tilde{O}(n + k^2)$ space. The number of passes is $2h - 1 = \tilde{O}(k)$.

In the insertion-only model, the implementation is straightforward. In the turnstile model, care has to be taken when implementing the above algorithm. We write $x = O(\max\{\log n, |U|/h\})$ to be the high probability upper bound on the number of BFS computation that a node participates in a single pass. We write $y = O(x \log n)$. Let $U_1, U_2, \ldots, U_y$ be random subsets of U such that each $u \in U$ joins each U_j with probability $1/x$, independently. Consider a node $v \in V$ and consider the rth pass. Let S be the subset of U such that $u \in S$ if $r = \tau_u + \text{dist}(u,v) - 1$, i.e., the BFS computation rooted at u hits v during the rth pass. We know that with high probability $|S| \le x$. By our choice of $U_1, U_2, \ldots, U_y$, we can infer that with high probability for each $u \in S$ there is at least one index j such that $S \cap U_j = \{u\}$.

To implement the rth pass in the turnstile model, each node $v \in V$ virtually maintains y edge set $Z_1, Z_2, \ldots, Z_y$. For each insertion (resp., deletion) of an edge $e = \{w, v\}$ satisfying $r = \tau_u + \mathrm{dist}(u, w) - 2$ for some $u \in U_j$, we add (resp., remove) the edge from the set Z_j. After processing the entire data stream, we take one edge out of each edge set $Z_1, Z_2, \ldots, Z_y$. In view of the above discussion, it suffices to only consider these edges when we grow the BFS trees. This can be implemented using y ℓ_0-samplers per each node $v \in V$, and the space complexity is still $\tilde{O}(ny) = \tilde{O}(n + k^2)$.

Step 2. At this moment we have computed $\mathrm{dist}(s, u)$ for each $u \in U$. Now we need to compute $\mathrm{dist}(s, t)$ for each $t \in V \setminus \{s\}$ by the formula $\mathrm{dist}(s, t) = \min_{u \in U \cap N^h(t)} \mathrm{dist}(s, u) + \mathrm{dist}(u, t)$.

In the insertion-only model, this task can be solved using h iterations of Bellman-Ford steps. Initially, $d_0(v) = \mathrm{dist}(s, v)$ for each $v \in U$, and $d_0(v) = \infty$ for each $v \in V \setminus U$. During the ith pass, we do the update $d_i(v) \leftarrow \min\{d_{i-1}(v), 1 + \min_{u \in N(v)} d_{i-1}(u)\}$. By Equation (2), we can infer that $d_h(t) = \mathrm{dist}(s, t)$ for each $t \in V$. A straightforward implementation of this procedure costs $\tilde{O}(n)$ space and $h = \tilde{O}(n/k)$ passes.

In the turnstile model, we can solve this task by growing a radius-h BFS tree rooted at u, for each $u \in U$, as in Step 1. During the process, each node $v \in V$ maintains a variable $d(v)$ which serves as the estimate of $\mathrm{dist}(s, v)$. Initially, $d(v) \leftarrow \infty$. When the partial BFS tree rooted at $u \in U$ hits v, we update $d(v)$ to be the minimum of the current value of $d(v)$ and $\mathrm{dist}(s, u) + \mathrm{dist}(u, v)$. At the end of the process, we have $d(v) = \min_{u \in U \cap N^h(t)} \mathrm{dist}(s, u) + \mathrm{dist}(u, v) = \mathrm{dist}(s, v)$ for each $v \in V$. This costs $\tilde{O}(n + k^2)$ space and $\tilde{O}(n/k)$ passes in view of the analysis of Step 1.

3.3 Extensions

In this section we consider the problem of solving c instances of BFS simultaneously for some $c \leq n$ and a simpler problem of computing the pairwise distance between the c given nodes. Both of these problems can be solved via a black box application of Theorem 3. In this section we show that it is possible to obtain better upper bounds.

▶ **Theorem 12.** *Given an n-node undirected graph G, for any given parameters $1 \leq c \leq k \leq n$, the pairwise distances between all pairs of nodes in a given set of c nodes in G can be computed with probability $1 - 1/n^{\Omega(1)}$ using $\tilde{O}(n/k)$ passes and $\tilde{O}(n + k^2)$ space in the turnstile model.*

Proof. Let S be the input node set of size c. Consider the modified Step 1 of our algorithm where each $s \in S$ is included in U with probability 1. Since $|S| = c \leq k$, we still have $|U| = \tilde{O}(k)$ with high probability. Recall that Step 1 of our algorithm calculates $\mathrm{dist}(u, v)$ for each $u \in U$ and $v \in U \cap N^h(u)$ in $\tilde{O}(n + k^2)$ space and $\tilde{O}(n/k)$ passes. Applying Equation (2) for each $s \in U$, we obtain the pairwise distances between all pairs of nodes in U, which includes S as a subset. There is no need to do Step 2. ◀

For example, if $c = n^{1/2}$, then Theorem 12 implies that we can compute the pairwise distances between all pairs of nodes in a given set of c nodes in $\tilde{O}(n)$ space and $\tilde{O}(n^{1/2})$ passes.

▶ **Theorem 13.** *Given an n-node undirected graph G, for any given parameters $1 \leq c \leq k \leq n$, one can solve c instances of BFS with probability $1 - 1/n^{\Omega(1)}$ using $\tilde{O}(n/k)$ passes and $\tilde{O}(cn + k^2)$ space in the turnstile model.*

Proof. Let S be the node set of size c corresponding to the roots of the c BFS instances. Consider the following modifications to our BFS algorithm.

Same as the proof of Theorem 12, in Step 1, include each $s \in S$ in U with probability 1. The modified Step 1 still takes $\tilde{O}(n + k^2)$ space and $\tilde{O}(n/k)$ passes, and it outputs the pairwise distances between all pairs of nodes in U.

Now consider Step 2. In the insertion-only model, remember that a BFS tree rooted at a node $s \in S$ can be constructed in $O(n)$ space and $h = \tilde{O}(n/k)$ passes using h iterations of Bellman-Ford steps. The cost of constructing all c BFS trees is then $O(cn)$ space and $\tilde{O}(n/k)$ passes.

In the turnstile model, we can also use the strategy of growing a radius-h BFS tree rooted at u, for each $u \in U$. During the process, each node $v \in V$ maintains c variables serving as the estimates of $\text{dist}(s, v)$, for all $s \in S$. The complexity of growing radius-h BFS trees is still $\tilde{O}(n + k^2)$ space and $\tilde{O}(n/k)$ passes. The extra space cost for maintaining these cn variables is $O(cn)$. ◀

For example, if $c = n^{1/3}$, then Theorem 13 implies that we can solve c instances of BFS in $\tilde{O}(n^{4/3})$ space and $\tilde{O}(n^{1/3})$ passes. Note that the space complexity of $\tilde{O}(n^{4/3})$ is necessary to output $c = n^{1/3}$ BFS trees.

Theorem 13 immediately gives the following corollary.

▶ **Corollary 14.** *Given an n-node connected undirected graph G with unweighted edges and a c-node subset S of G, for any given parameters $1 \le c \le k \le n$, finding a Steiner tree in G that spans S can be approximated to within a factor of 2 with probability $1 - 1/n^{\Omega(1)}$ using $\tilde{O}(n/k)$ passes and $\tilde{O}(cn + k^2)$ space in the turnstile model.*

Note that if we do not need to construct a Steiner tree, and only need to approximate the size of an optimal Steiner tree, then Theorem 12 can be used in place of Theorem 13.

3.4 Diameter Approximation

It is well-known that the maximum distance label in a BFS tree gives a 2-approximation of diameter. We show that it is possible to improve the approximation ratio to nearly 1.5 without sacrificing the space and pass complexities.

Roditty and Williams [40] showed that a nearly 1.5-approximation of diameter can be computed with high probability as follows.

1. Let S_1 be a node set chosen by including each node $v \in V$ to S_1 with probability $p = (\log n)/\sqrt{n}$ independently. Perform a BFS from each node $v \in S_1$.
2. Let $v^\star$ be a node chosen to maximize $\text{dist}(v^\star, S_1)$. Break the tie arbitrarily. Perform a BFS from $v^\star$.
3. Let S_2 be the node set consisting of the $\sqrt{n}$ nodes closest to $v^\star$, where ties are broken arbitrarily. Perform a BFS from each node $v \in S_2$.

Let D^* be the maximum distance label ever computed during the BFS computations in the above procedure. Roditty and Williams [40] proved that D^* satisfies that $\lfloor 2D/3 \rfloor \le D^* \le D$, where D is the diameter of G.

The algorithm of Roditty and Williams [40] can be implemented in the streaming model by applying Theorem 13 with $c = \tilde{O}(\sqrt{n})$, but we can do better. Note that when we perform BFS from the nodes in S_1 and S_2, it is not necessary to store the entire BFS trees. For example, in order to select v^*, we only need to let each node v know $\text{dist}(v, S_1)$, which is the maximum distance label of v in all BFS trees computed in Step 1. Therefore, the $O(cn)$ term in the space complexity of Theorem 13 can be improved to $O(n)$. That is, the space and pass complexities are the same as the cost for computing a *single* BFS tree using Theorem 3. We conclude the following theorem.

▶ **Theorem 15.** *Given an n-node connected undirected graph G, a diameter approximation D^* satisfying $\lfloor 2D/3 \rfloor \leq D^* \leq D$, where D is the diameter of G, can be computed with probability $1 - 1/n^{\Omega(1)}$ in p passes using $\tilde{O}((n/p)^2)$ space, for each $1 \leq p \leq \tilde{O}(\sqrt{n})$ in the turnstile model.*

4 Depth-First Search

A straightforward implementation of the naive DFS algorithm in the streaming model costs either $n-1$ passes with $\tilde{O}(n)$ space or a single pass with $O(n^2)$ space. Khan and Mehta [30] recently showed that it is possible to obtain a smooth tradeoff between the two extremes. Specifically, they designed an algorithm that requires at most $\lceil n/k \rceil$ passes using $\tilde{O}(nk)$ space, where k is any positive integer. Furthermore, for the case the height h of the computed DFS tree is small, they further decrease the number of passes to $\lceil h/k \rceil$. In Section 4.1, we will provide a very simple alternative proof of this result, via sparse certificates for k-node-connectivity.

In the worst case, the "space $\times$ number of passes" of the algorithms of Khan and Mehta [30] is still $\tilde{O}(n^2)$. In Section 4.2, we will show that it is possible to improve this upper bound asymptotically when the number of passes p is super-constant. Specifically, for any parameters $1 \leq s \leq k \leq n$, we obtain the following DFS algorithms.

- A deterministic algorithm using $\tilde{O}((n/k)+(k/s))$ passes and $\tilde{O}(ns)$ space in the insertion-only model. After balancing the parameters, the space complexity is $\tilde{O}(n^2/p^2)$ for p-pass algorithms, for each $1 \leq p \leq \tilde{O}(\sqrt{n})$.
- A randomized algorithm using $\tilde{O}((n/k)+(k/s))$ passes and $\tilde{O}(ns^2)$ space in the turnstile model. After balancing the parameters, the space complexity is $\tilde{O}(n^3/p^4)$ for p-pass algorithms, for each $1 \leq p \leq \tilde{O}(\sqrt{n})$.

4.1 A Simple DFS Algorithm

In this section, we present a simple alternative proof of the result of Khan and Mehta [30] that a DFS tree can be constructed in $\lceil h/k \rceil$ passes using $\tilde{O}(nk)$ space, for any given parameter k, where h is the height of the computed DFS tree.

Sparse Certificate for s-Node-Connectivity. A *strong s-VC certificate* of a graph H is its subgraph K such that for any supergraph G of H, for every pair of nodes $s^*, t^* \in G$, if they are c-node-connected in G, then they are c'-node-connected for some $c' \geq \min\{s, c\}$ in the graph obtained from G by replacing its subgraph H with K. A *sparse* strong s-VC certificate of the graph G is exactly what we need here. Eppstein, Galil, Italiano, and Nissenzweig [12] showed that such a sparse subgraph of $O(ns)$ edges can be computed in a single pass with $\tilde{O}(ns)$ space *deterministically* in the insertion-only model. In the turnstile model, Guha, McGregor, and Tench [25] showed that such a sparse subgraph of $\tilde{O}(ns^2)$ edges can be computed with high probability in a single pass using $\tilde{O}(ns^2)$ space. This result can be inferred from Theorem 8 of [25] with $\epsilon = \Theta(1/s)$. In [25] the analysis only considers the case $G = H$, but it is straightforward to extend the analysis to incorporate any supergraph G of H.

If the subgraph K of the graph H satisfies the above requirement for the special case of $G = H$, then K is said to be a *s-VC certificate* of H. Our simple DFS algorithm relies on this tool.

▶ **Lemma 16.** *Suppose K is a $(k+1)$-VC certificate of H. Let T be any DFS tree of K. Consider any two nodes u and v such that the least common ancestor w of u and v are within the top k layers of T. If $w \neq u$ and $w \neq v$, then u and v are not adjacent in H.*

Proof. Suppose u, v, and w violate the statement of the lemma. That is, u and v are adjacent in H. Since T is a DFS tree, u and v are not adjacent in K, and each path connecting u and v must pass through a node that is a common ancestor of u and v. Let c_H (resp., c_K) be the u-v node-connectivity in H (resp., K). The above discussion implies that $c_K \leq k$ and $c_H \geq c_K + 1$, contradicting the assumption that K is a $(k+1)$-VC certificate of H. ◀

Algorithm. Using Lemma 16, we can construct a DFS tree of G recursively as follows. Pick K as a $(k+1)$-VC certificate of G. Compute a DFS tree T of K. Let T' be the tree induced by the top $k+1$ layers of of T. Let $v_1, v_2, \ldots, v_z$ be the leaves of T'. Denote S_i as the set of descendants of v_i in T, including v_i. By Lemma 16, there exists no edge in G that crosses two distinct sets S_i and S_j. For each $1 \leq i \leq z$, we recursively find a DFS tree T_i of the subgraph of G induced by S_i rooted at v_i. By the above observation, we can obtain a valid DFS tree of G by appending $T_1, T_2, \ldots, T_z$ to T'.

Analysis. If the height of the final DFS tree is h, then the depth of the recursion is at most $\lceil h/k \rceil$. The cost for computing a $(k+1)$-VC certificate is 1 pass and $\tilde{O}(nk)$ space, and the resulting subgraph K has $O(nk)$ edges. Therefore, the total number of passes is at most $\lceil h/k \rceil$, and the overall space complexity is $\tilde{O}(nk)$.

4.2 Streaming Implementation of the Algorithm of Aggarwal and Anderson

The bounds of Theorem 4 are attained via an implementation of the parallel DFS algorithm of Aggarwal and Anderson [2] in the streaming model, with the help of various tools, including the strong sparse certificates for s-node-connectivity described above. Due to the page limit, we only provide a sketch of the proof. The complete proof will be presented in the full version of this paper.

At a high level, the DFS algorithm of Aggarwal and Anderson [2] works as follows. Start with a maximal matching, and then merge these length-1 paths iteratively into a constant number of node-disjoint paths such that the number of nodes not in any path is at most $|V|/2$. The algorithm then constructs the initial segment of the DFS tree from these paths. Each remaining connected component is solved recursively. The final DFS tree is formed by appending the DFS trees of recursive calls to the initial segment.

The bottleneck of this DFS algorithm is a task called MaximalPaths which is a variant of the maximal node-disjoint paths problem between a set of source nodes S and a set of sink nodes T. In this variant, each member of S is a path instead of a node. Goldberg, Plotkin, and Vaidya [24] gave a parallel algorithm for this problem. Their algorithm has two phases. For any given parameter k, they showed that after k iterations of the algorithm of the first phase, the number of sources in S that are still *active* is at most n/k. These remaining active sources are processed one-by-one in the second phase. Using this approach with $k = \sqrt{n}$, MaximalPaths can be solved in the streaming model with $\tilde{O}(\sqrt{n})$ passes and $\tilde{O}(n)$ space. To further reduce the pass complexity, we apply the sparse certificates for s-node-connectivity of Eppstein, Galil, Italiano, and Nissenzweig [12] and Guha, McGregor, and Tench [25], which allow us to process the remaining active sources in batches. In the insertion-only model, we obtain a deterministic p-pass algorithm with space complexity $\tilde{O}(n^2/p^2)$, for each $1 \leq p \leq \tilde{O}(\sqrt{n})$. For the more challenging turnstile model, we obtain a randomized algorithm with a somewhat worse space complexity of $\tilde{O}(n^3/p^4)$.

5 Single-Pass Lower Bounds

In this section, we use the lower bound of the 1-way randomized communication complexity for the INDEX problem [1] to show the single-pass space lower bound for computing approximate MLST to within an additive error k. This gives a complementary result for Theorem 1.

▶ **Theorem 17.** *In the insertion-only model, given a connected n-node simple undirected graph G, computing a spanning tree of G that has at least $\mathrm{leaf}(G)-k$ leaves for any $k \in [1,(n-5)/4]$ requires $\Omega(n^2/k^2)$ bits for any single-pass randomized streaming algorithm that can succeed with probability at least $2/3$.*

Proof. We begin with a reduction from an n^2-bit instance of the INDEX problem to computing a spanning tree of $(2n+3)$-node graph G with $\mathrm{leaf}(G)$ leaves for any $n \geq 1$. Given Alice's input in the INDEX problem, i.e. a bit-array of length n^2, we construct an n by n bipartite graph H, as part of G, in which edge (x_i, y_j) for every $i, j \in [1, n]$ corresponds to the $((i-1)n+j)$-th bit in Alice's array. Then, given Bob's input, a tuple (i, j) for some $i, j \in [1, n]$, we construct the remaining part of G by adding three additional nodes s, t, and ℓ, and

- connecting an edge from s to z for every node $z \neq y_j$ in H, and
- adding edge (ℓ, x_i), (s, t), and (t, y_j).

It clear that G is connected and has

$$\mathrm{leaf}(G) = \begin{cases} 2n+1 & \text{if } (x_i, y_j) \in H \\ 2n & \text{otherwise} \end{cases}$$

Thus, having a single-pass streaming algorithm to compute $\mathrm{leaf}(G)$ suffices to decide the n^2-bit instance of the INDEX problem, i.e. for Bob to tell what the $((i-1)n+j)$-th bit in Alice's array is. This requires $\Omega(n^2)$ bits. To obtain the hardness result for MLST with additive error k for any $k \geq 1$, one can duplicate $H \cup \{\ell, t\}$ into $(k+1)$ copies and let the copies share the same s, so G is connected, has $(k+1)(2n+2)+1$ nodes, and has

$$\mathrm{leaf}(G) = \begin{cases} (2n+1)(k+1) & \text{if } (x_i, y_j) \in H \\ 2n(k+1) & \text{otherwise} \end{cases}$$

Hence, having a single-pass streaming algorithm to compute $\mathrm{leaf}(G)$ for G of $(k+1)(2n+2)+1$ nodes to within an additive error k suffices to decide the n^2-bit INDEX problem. Replace $(k+1)(2n+2)+1 = n'$ and $n^2 = \Omega((n'/k)^2)$ yields the desired bound. ◀

6 Conclusion

In this paper, we devised semi-streaming algorithms for spanning tree computations, including max-leaf spanning trees, BFS trees, and DFS trees. For max-leaf spanning trees, despite that any streaming algorithm requires $\Omega(n^2)$ space to compute the exact solution, we show how to compute a $(1+\varepsilon)$-approximation using a single pass and $\tilde{O}(n)$ space, albeit in super-polynomial time. For BFS trees and DFS trees, we show how to compute them using $O(\sqrt{n})$ passes and $\tilde{O}(n)$ space, and offer a smooth tradeoff between pass complexity and space usage.

The pass complexities of our algorithms for BFS trees and DFS trees are still far from the known lower bounds, $\omega(1)$ passes for BFS trees [17] and the trivial 1 pass for DFS trees. It is unclear whether our upper bounds can be further reduced or the known lower bounds can be improved. We leave closing the gap to future work.

References

1 Farid M. Ablayev. Lower bounds for one-way probabilistic communication complexity and their application to space complexity. *Theor. Comput. Sci.*, 157(2):139–159, 1996.

2 Alok Aggarwal and Richard J. Anderson. A random NC algorithm for depth first search. *Combinatorica*, 8(1):1–12, March 1988.

3 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 459–467, 2012.

4 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for $(\Delta + 1)$ vertex coloring. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 767–786, 2019.

5 Jesse Beisegel. Characterising AT-free graphs with BFS. In *Graph-Theoretic Concepts in Computer Science - 44th International Workshop, WG 2018, Cottbus, Germany, June 27-29, 2018, Proceedings*, pages 15–26, 2018.

6 Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

7 Vladimir Braverman, Rafail Ostrovsky, and Dan Vilenchik. How hard is counting triangles in the streaming model? In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, pages 244–254, 2013.

8 Joseph Cheriyan and Ramakrishna Thurimella. Algorithms for parallel k-vertex connectivity and sparse certificates. In *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing*, STOC '91, pages 391–401. ACM, 1991.

9 Miroslav Chlebík and Janka Chlebíková. Approximation hardness of dominating set problems in bounded degree graphs. *Inf. Comput.*, 206(11):1264–1275, 2008.

10 Derek G. Corneil, Feodor F. Dragan, and Ekkehard Köhler. On the power of BFS to determine a graph's diameter. *Networks*, 42(4):209–222, 2003.

11 M. Elkin. Distributed exact shortest paths in sublinear time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 757–770, New York, NY, USA, 2017. ACM.

12 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification – a technique for speeding up dynamic graph algorithms. *JACM*, 44(5):669–696, 1997.

13 Shimon Even and Robert Endre Tarjan. Computing an st-numbering. *Theoretical Computer Science*, 2(3):339–344, 1976.

14 Martin Farach-Colton, Tsan-sheng Hsu, Meng Li, and Meng-Tsung Tsai. Finding articulation points of large graphs in linear time. In *Algorithms and Data Structures - 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings*, pages 363–372, 2015.

15 Martin Farach-Colton and Meng-Tsung Tsai. Tight approximations of degeneracy in large graphs. In *LATIN 2016: Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico, April 11-15, 2016, Proceedings*, pages 429–440, 2016.

16 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005.

17 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the data-stream model. *SIAM J. Comput.*, 38(5):1709–1727, 2008.

18 T. Fleiner and G. Wiener. Coloring signed graphs using DFS. *Optimization Letters*, 10(4):865–869, April 2016.

19 L.R. Ford. *Network Flow Theory*. Paper P. Rand Corporation, 1956.

20 Giulia Galbiati, Francesco Maffioli, and Angelo Morzenti. A short note on the approximability of the maximum leaves spanning tree problem. *Inf. Process. Lett.*, 52(1):45–49, 1994.

21 Rajiv Gandhi, Mohammad Taghi Hajiaghayi, Guy Kortsarz, Manish Purohit, and Kanthi K. Sarpatwar. On maximum leaf trees and connections to connected maximum cut problems. *Inf. Process. Lett.*, 129:31–34, 2018.

22 Mohsen. Ghaffari and Jason. Li. Improved distributed algorithms for exact shortest paths. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2018.

23 Mohsen Ghaffari and Merav Parter. Near-optimal distributed DFS in planar graphs. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 21:1–21:16, 2017.

24 A. V. Goldberg, S. A. Plotkin, and P. M. Vaidya. Sublinear-time parallel algorithms for matching and related problems. *JALG*, 14(2):180–213, 1993.

25 Sudipto Guha, Andrew McGregor, and David Tench. Vertex and hyperedge connectivity in dynamic graph streams. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 241–247, 2015.

26 Mohammad Taghi Hajiaghayi, Guy Kortsarz, Robert MacDavid, Manish Purohit, and Kanthi K. Sarpatwar. Approximation algorithms for connected maximum cut and related problems. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 693–704, 2015.

27 Chien-Chung Huang, Danupon Nanongkai, and Thatchaphol Saranurak. Distributed exact weighted all-pairs shortest paths in $\tilde{O}(n^{5/4})$ rounds. In *IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 168–179, 2017.

28 Hossein Jowhari, Mert Sağlam, and Gábor Tardos. Tight bounds for ℓ_p samplers, finding duplicates in streams, and related problems. In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '11, pages 49–58, New York, NY, USA, 2011. ACM.

29 Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. *SIAM J. Comput.*, 46(1):456–477, 2017.

30 Shahbaz Khan and Shashank Mehta. Depth first search in the semi-streaming model. In *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

31 Daniel J. Kleitman and Douglas B. West. Spanning trees with many leaves. *SIAM J. Discrete Math.*, 4(1):99–106, 1991.

32 F. T. Leighton, B. M. Maggs, and S. B. Rao. Packet routing and job-shop scheduling in $O(\text{Congestion} + \text{Dilation})$ steps. *Combinatorica*, 14(2):167–186, 1994.

33 R. Lipton and R. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

34 Hsueh-I Lu and R. Ravi. The power of local optimization: Approximation algorithms for maximum-leaf spanning tree. In *In Proceedings, Thirtieth Annual Allerton Conference on Communication, Control and Computing*, pages 533–542, 1992.

35 Hsueh-I Lu and R. Ravi. Approximating maximum leaf spanning trees in almost linear time. *J. Algorithms*, 29(1):132–141, 1998.

36 Andrew McGregor, David Tench, Sofya Vorotnikova, and Hoa T. Vu. Densest subgraph in dynamic graph streams. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, pages 472–482, 2015.

37 S. Muthukrishnan. Data streams: Algorithms and applications. *Found. Trends Theor. Comput. Sci.*, 1(2):117–236, August 2005.

38 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992.

39 Jelani Nelson and Huacheng Yu. Optimal lower bounds for distributed and streaming spanning forest computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete*

Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019, pages 1844–1860, 2019.

40 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings 45th ACM Symposium on Theory of Computing (STOC)*, pages 515–524, 2013.

41 Jens M. Schmidt. A simple test on 2-vertex- and 2-edge-connectivity. *Inf. Process. Lett.*, 113(7):241–244, 2013.

42 Roberto Solis-Oba, Paul S. Bonsma, and Stefanie Lowski. A 2-approximation algorithm for finding a spanning tree with maximum number of leaves. *Algorithmica*, 77(2):374–388, 2017.

43 Xiaoming Sun and David P. Woodruff. Tight bounds for graph problems in insertion streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA*, pages 435–448, 2015.

44 Robert Endre Tarjan. A note on finding the bridges of a graph. *Inf. Process. Lett.*, 2(6):160–161, 1974.

45 J. D. Ullman and M. Yannakakis. High-probability parallel transitive-closure algorithms. *SIAM Journal on Computing*, 20(1):100–125, 1991.

Shortest Reconfiguration of Colorings Under Kempe Changes

Marthe Bonamy
CNRS, LaBRI, Université de Bordeaux,
Talence, France
marthe.bonamy@u-bordeaux.fr

Marc Heinrich
Université Lyon 1, LIRIS,
UMR5205, France
marc.heinrich@univ-lyon1.fr

Takehiro Ito
Graduate School of Information Sciences,
Tohoku University, Japan
takehiro@ecei.tohoku.ac.jp

Yusuke Kobayashi
Research Institute for Mathematical Sciences,
Kyoto University, Japan
yusuke@kurims.kyoto-u.ac.jp

Haruka Mizuta
Graduate School of Information Sciences,
Tohoku University, Japan
haruka.mizuta.s4@dc.tohoku.ac.jp

Moritz Mühlenthaler
Laboratoire G-SCOP, Grenoble INP,
Université Grenoble Alpes, France
moritz.muhlenthaler@grenoble-inp.fr

Akira Suzuki
Graduate School of Information Sciences,
Tohoku University, Japan
a.suzuki@ecei.tohoku.ac.jp

Kunihiro Wasa
National Institute of Informatics,
Tokyo, Japan
wasa@nii.ac.jp

Abstract

A k-coloring of a graph maps each vertex of the graph to a color in $\{1, 2, \ldots, k\}$, such that no two adjacent vertices receive the same color. Given a k-coloring of a graph, a Kempe change produces a new k-coloring by swapping the colors in a bicolored connected component. We investigate the complexity of finding the smallest number of Kempe changes needed to transform a given k-coloring into another given k-coloring. We show that this problem admits a polynomial-time dynamic programming algorithm on path graphs, which turns out to be highly non-trivial. Furthermore, the problem is NP-hard even on star graphs and we show that on such graphs it admits a constant-factor approximation algorithm and is fixed-parameter tractable when parameterized by the number k of colors. The hardness result as well as the algorithmic results are based on the notion of a canonical transformation.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Combinatorial Reconfiguration, Graph Algorithms, Graph Coloring, Kempe Equivalence

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.35

Funding *Takehiro Ito*: Partially supported by JST CREST Grant Number JPMJCR1402, and JSPS KAKENHI Grant Numbers JP18H04091 and JP19K11814, Japan.
Yusuke Kobayashi: Supported by JST ACT-I Grant Number JPMJPR17UB, and JSPS KAKENHI Grant Numbers JP16K16010 and JP18H05291, Japan.
Akira Suzuki: Partially supported by JST CREST Grant Number JPMJCR1402, and JSPS KAKENHI Grant Numbers JP17K12636 and JP18H04091, Japan.
Kunihiro Wasa: Partially supported by JST CREST Grant Numbers JPMJCR18K3 and JPMJCR1401, and JSPS KAKENHI Grant Number 19K20350, Japan.

Acknowledgements This work is partially supported by JSPS and MEAE-MESRI under the Japan-France Integrated Action Program (SAKURA).

1 Introduction

Reconfiguration problems ask for the existence of a transformation between two solutions of an instance of a combinatorial problem, such as the satifiability problem, the stable set problem, and so forth. Our reference problem is the k-coloring problem. Recall that a *k-coloring* of a graph partitions the vertex set into at most k stable sets, called *color classes*. A classical technique to obtain a new k-coloring from a given one is the Kempe change: Given a k-coloring, a *Kempe change* swaps the colors in a connected component of a subgraph induced by two color classes. We say that two k-colorings α and β of a graph admit a *Kempe-sequence of length ℓ* if there is a sequence $\gamma_0, \gamma_1, \ldots, \gamma_\ell$ of k-colorings, such that $\gamma_0 = \alpha$, $\gamma_\ell = \beta$, and for $0 \leq i < \ell$, the coloring γ_{i+1} can be obtained from γ_i by performing a single Kempe change. Two k-colorings that admit a Kempe-sequence (of any length) are called *Kempe-equivalent*. Kempe-equivalence has been of great interest in graph theory [15, 17, 19], sampling of colorings [8, 25], and statistical physics [2, 20, 21]. We consider a natural optimization variant of Kempe-equivalence: Given a number ℓ and two k-colorings of a graph, do the two colorings admit a Kempe-sequence of length at most ℓ?

One of the first algorithms that exhibits Kempe-sequences is due to Las Vergnas and Meyniel [15]. From their analysis it follows that for a d-degenerate graph and any $k > d$, any two k-colorings of the graph are Kempe-equivalent. More recently, several results of a similar flavor were obtained [2, 3, 10, 19]. Interestingly, no non-trivial subexponential upper bounds on the length of the Kempe-sequences computed by the algorithm of Las Vergnas and Meyniel are known, even for small values of k. Using different techniques, Bousquet and Heinrich showed recently that for $k \geq d + 2$, any two k-colorings of a d-degenerate graph admit a Kempe-sequence of polynomial length in d [6]. Note that none of the algorithms in the references mentioned above is known to exhibit Kempe-sequences of minimal length. Furthermore, to the best of our knowledge, the complexity of determining the minimal length of a Kempe-sequence between two k-colorings is open. The aim of this paper is to settle the complexity of this task and to provide efficient exact and approximate algorithms for it.

We show that a Kempe-sequence of minimal length for two k-colorings of a path graph can be found in polynomial time by a non-trivial dynamic programming algorithm. On the other hand, it is unlikely that this positive result can be generalized significantly, since we also show that the same problem on star graphs is NP-hard. Note that since star graphs and path graphs are bipartite, a Kempe-sequence of linear length in the input size can be found efficiently [3, 19].

In order to illustrate why finding Kempe-sequences of minimal length is non-trivial even on path graphs, let us consider the example shown in Figure 1. It shows two Kempe-sequences $\alpha, \gamma_1, \gamma_2, \beta$ and $\alpha, \gamma'_1, \gamma'_2, \gamma'_3, \beta$ between two 3-colorings α and β of a path on seven vertices. The latter Kempe-sequence changes only the colors of vertices that receive differents colors in α and β. On the other hand, the Kempe-sequence $\alpha, \gamma_1, \gamma_2, \beta$, which is optimal, changes also twice the color of the middle vertex v, which receives the same color in α and β. The purpose of changing the color of v is to build a large connected component consisting of the vertices with colors 1 and 2. As a result, with a single Kempe change, all vertices can be recolored to their target colors. We conclude from the example that a key difficulty is to build up suitable bicolored components in order to obtain a short Kempe-sequence.

Related Work

Finding shortest transformations between configurations has recently received much attention in different domains, ranging from the triangulations of point sets and polygons [16, 1] to satisfying assignments of Boolean formulas [22] to puzzles [9, 12, 24]; please see [13, 23]

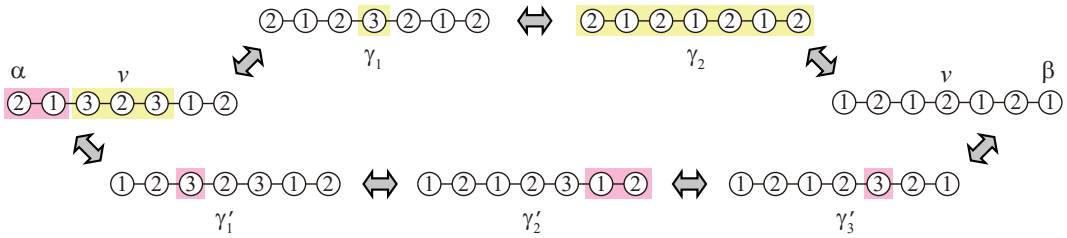


Figure 1 Two Kempe-sequences between 3-colorings α and β of a path on seven vertices. The upper sequence $\langle \alpha, \gamma_1, \gamma_2, \beta \rangle$ is shortest even though it recolors the middle vertex v twice; notice that v receives the same color 2 in α and β. On the other hand, the lower sequence $\langle \alpha, \gamma'_1, \gamma'_2, \gamma'_3, \beta \rangle$ is not shortest although it recolors only the vertices receiving different colors in α and β.

for recent suveys. Such problems are often NP-hard, even if the existence of a suitable transformation between two configurations can be decided in polynomial time. A recent example is the NP-hardness of solving the n-Rubik's cube using a minimal number of steps [9].

We say that two k-colorings are *equivalent under elementary recolorings* if they admit a Kempe-sequence, such that any two consecutive k-colorings of the sequence differ with respect to the color of a single vertex. In contrast to Kempe-equivalence, the complexity of deciding equivalence under elementary recolorings is quite well understood [5, 7]. Furthermore, Johnson et al. showed that deciding the existence of a transformation of length ℓ between two k-colorings in this setting admits a polynomial-time algorithm for $k = 3$ and is FPT when parameterized by $k + \ell$ [14]. It may seem quite surprising that there is a polynomial-time algorithm for $k = 3$, since deciding if a graph admits a 3-coloring is NP-complete. This polynomial-time algorithm is based on a potential argument: Each vertex is assigned a non-negative potential value and in each step of a shortest transformation, the potential is decreased by a positive constant value. When the target coloring is reached the potential becomes zero. Due to long-range effects of general Kempe changes it is unlikely that a similar potential argument can be used in our setting; hence our techniques are very different.

We can think of a k-coloring in terms of placing labeled tokens the vertices of a graph. Transformations between labelled token configurations on graphs have been considered in a slightly different setting called *token swapping*: from a given token configuration a new configuration can be obtained by swapping two tokens on adjacent vertices. The goal is again to decide whether there is a transformation of length at most ℓ between two given token configurations. It is not required that adjacent vertices have tokens with different labels. The problem is related to the design of efficient sorting networks. It is NP-complete on graphs of treewidth two and constant diameter [4] and admits a polynomial-time 4-factor approximation algorithm [18]. Several variations of the problems have been studied, for example colored token swapping [26, 27], where tokens of the same color are indistinguishable.

Our Contribution

Let $\mathcal{G}$ be a class of graphs. We consider the following problem.

Kempe Distance on $\mathcal{G}$
input: Graph $G \in \mathcal{G}$, numbers $k, \ell \in \mathbb{N}$, two k-colorings α, β of G
question: Do α and β admit a Kempe-sequence of length at most ℓ?

We show that Kempe Distance on paths admits a $O(kn)$-time dynamic programming algorithm, where n is the number of vertices of the input graph. The algorithm can easily be modified to output a shortest Kempe-sequence. The analysis of the algorithm is highly

non-trivial. Roughly speaking, given a path on vertices $v_1, v_2, \ldots, v_n$ and two k-colorings α and β of the input graph, the algorithm computes for $1 \leq i \leq n$ two kinds of quantities on the sub-path $v_1, v_2, \ldots, v_i$ according to eight rules (please refer to Section 2 for an overview). It computes i) the length of a shortest Kempe-sequence between the two colorings restricted to the current subpath and ii) for each color a that is different from the source and target color of v_i, the algorithm checks whether there is a shortest Kempe-sequence on the subpath such that v_i has color a at some point. To establish the correctness of the algorithm, we prove several interesting properties of Kempe-sequences between k-colorings of a path graph. For instance, we show that in a shortest Kempe-sequence, the color of each vertex changes at most twice.

We complement the above result by showing that the problem Kempe Distance on stars is NP-complete by a reduction from the problem Hamiltonian Cycle. In contrast, we show that a Kempe-sequence of minimal length that certifies the equivalence of two k-colorings of a star graph under elementary recolorings can be found efficiently. On the positive side, we show that there is a polynomial-time algorithm that computes a Kempe-sequence of length at most $4\,\mathrm{OPT}(I) + 1$, where $\mathrm{OPT}(I)$ denotes the length of a shortest Kempe-sequence for an instance I of Kempe Distance on stars, and give an almost matching lower bound. Furthermore, we show that Kempe Distance on stars parameterized by the number k of colors is fixed-parameter tractable. The algorithmic results as well as the hardness result are based on the notion of a *canonical transformation*. We would like to remark that many algorithms for reconfiguration problems make use of a *canonical configuration*, that is, the existence of a transformation is established by showing that any configuration can be transformed into a certain canonical one. Here, we show that for any Kempe-sequence between two k-colorings of a star there is a canonical Kempe-sequence of at most the same length. Hence, it is sufficient to consider canonical shortest Kempe-sequences.

Notation

A *star graph* on n vertices consists of a *center vertex* of degree $n-1$ and $n-1$ *leaf vertices*, each of degree one. For a k-coloring α of a graph, a maximal monochromatic set of vertices is called a *color class*. Let G be a graph and let α and β be two k-colorings of G. Suppose that α and β admit a Kempe-sequence $s := (\gamma_0, \gamma_1, \ldots, \gamma_l)$. We say that s is a Kempe-sequence *from* α *to* β and denote its length by $|s|$. For a Kempe sequence $s = (\gamma_0, \gamma_1, \ldots, \gamma_l)$ and for a vertex $v \in V$, the *color transition of* v *in* s is a sequence of colors $c_0 \to c_1 \to \cdots \to c_p$ with $c_i \neq c_{i-1}$ for each i that represents the transition of colors of v in s. In other words, $c_0, c_1, \ldots, c_p$ is obtained from $\gamma_0(v), \gamma_1(v), \ldots, \gamma_l(v)$ by removing duplicates if a color appears consecutively in the sequence. When the color transition of v is $c_0 \to c_1 \to \cdots \to c_p$, we say that v is *recolored* p *times* in the Kempe sequence.

Proofs marked with $(*)$ have been omitted due to space limitations.

2 Kempe Distance on Paths: a polynomial-time algorithm

In this section we present a polynomial-time algorithm for the problem Kempe Distance on paths. In the following, let P be a path graph and we denote its vertex set by $V = \{v_1, v_2, \ldots, v_n\}$ and its edge set by $E = \{v_1v_2, v_2v_3, \ldots, v_{n-1}v_n\}$. Let $C = \{1, 2, \ldots, k\}$ be the color set and let $\alpha : V \to C$ and $\beta : V \to C$ be the initial and target k-colorings of P, respectively. For $i = 1, 2, \ldots, n$, the colors $\alpha(v_i)$ and $\beta(v_i)$ are denoted by α_i and β_i, respectively.

For $i \in \{1, \dots, n\}$, let P_i be the subpath of P induced by $V_i := \{v_1, \dots, v_i\}$. The restriction of α (resp. β) to $\{v_1, \dots, v_i\}$ is also denoted by α (resp. β) for notational convenience. For $i \in \{1, \dots, n\}$, let $l(P_i, \alpha, \beta) \in \mathbb{Z}_+$ be the length of a shortest Kempe sequence for the instance (P_i, α, β). We simply denote $l_i := l(P_i, \alpha, \beta)$ if α and β are clear. For each $i \in \{1, \dots, n\}$ and for a color $a \in C$, we define $f(i, a)$ as follows: $f(i, a) = \mathsf{yes}$ if $a \notin \{\alpha_i, \beta_i\}$ and there exists a Kempe sequence s of length l_i such that the color transition of v_i is $\alpha_i \to a \to \beta_i$, and $f(i, a) = \mathsf{no}$ otherwise.

Our algorithm adopts a dynamic programming approach that computes l_i and $f(i, a)$ for each $i \in \{1, \dots, n\}$ and for each $a \in C$. For $i \geq 2$, the update formula is as follows.

(U1) If $\alpha_i = \beta_i$, then $l_i = l_{i-1}$ and $f(i, a) = \mathsf{no}$ for any $a \in C$.
(U2) If $\alpha_i = \beta_{i-1}$ and $\alpha_{i-1} = \beta_i$, then $l_i = l_{i-1}$ and $f(i, a) = \mathsf{no}$ for any $a \in C$.
(U3) If $\alpha_i = \beta_{i-1}$, $\alpha_{i-1} \neq \beta_i$, and $f(i-1, \beta_i) = \mathsf{yes}$, then $l_i = l_{i-1}$ and $f(i, a) = \mathsf{no}$ for any $a \in C$.
(U4) If $\alpha_i \neq \beta_{i-1}$, $\alpha_{i-1} = \beta_i$, and $f(i-1, \alpha_i) = \mathsf{yes}$, then $l_i = l_{i-1}$ and $f(i, a) = \mathsf{no}$ for any $a \in C$.
(U5) If $\alpha_i = \beta_{i-1}$, $\alpha_{i-1} \neq \beta_i$, and $f(i-1, \beta_i) = \mathsf{no}$, then $l_i = l_{i-1} + 1$ and

$$f(i, a) = \begin{cases} \mathsf{yes} & \text{if } a = \alpha_{i-1} \text{ or } f(i-1, a) = \mathsf{yes}, \\ \mathsf{no} & \text{otherwise} \end{cases}$$

for $a \in C$.
(U6) If $\alpha_i \neq \beta_{i-1}$, $\alpha_{i-1} = \beta_i$, and $f(i-1, \alpha_i) = \mathsf{no}$, then $l_i = l_{i-1} + 1$ and

$$f(i, a) = \begin{cases} \mathsf{yes} & \text{if } a = \beta_{i-1} \text{ or } f(i-1, a) = \mathsf{yes}, \\ \mathsf{no} & \text{otherwise} \end{cases}$$

for $a \in C$.
(U7) If $\alpha_i, \beta_i, \alpha_{i-1}$, and β_{i-1} are distinct, then $l_i = l_{i-1} + 1$ and

$$f(i, a) = \begin{cases} \mathsf{yes} & \text{if } a = \alpha_{i-1} \text{ and } f(i-1, \alpha_i) = \mathsf{yes}, \\ \mathsf{yes} & \text{if } a = \beta_{i-1} \text{ and } f(i-1, \beta_i) = \mathsf{yes}, \\ \mathsf{no} & \text{otherwise} \end{cases}$$

for $a \in C$.
(U8) If $\alpha_i \neq \beta_i$ and $\alpha_{i-1} = \beta_{i-1}$, then $l_i = l_{i-1} + 1$ and

$$f(i, a) = \begin{cases} \mathsf{yes} & \text{if } i \geq 3,\ a = \alpha_{i-1},\ \alpha_i = \alpha_{i-2},\ \beta_i = \beta_{i-2}, \text{ and } f(i-2, a) = \mathsf{yes}, \\ \mathsf{no} & \text{otherwise} \end{cases}$$

for $a \in C$.

In order to show the validity of (U1)–(U8), we introduce the following properties (P1) and (P2), and show simultaneously that (P1), (P2), and (U1)–(U8) hold for any $i \in \{1, \dots, n\}$ by induction.

(P1) There exists a Kempe sequence of length l_i such that v_i is recolored at most once in it, that is, v_i is recolored directly from α_i to β_i if $\alpha_i \neq \beta_i$, and v_i is never recolored if $\alpha_i = \beta_i$.
(P2) For any Kempe sequence s for (P_i, α, β), v_i is recolored at most $2|s| - 2l_i + 2$ times in s. In particular, v_i is recolored at most twice in any shortest Kempe sequence.

▶ **Proposition 1** (∗). *For any pair of colorings $\alpha : V \to C$ and $\beta : V \to C$, (U1)–(U8) hold for $i \geq 2$, and (P1) and (P2) hold for any $i \geq 1$.*

By using this proposition, we can obtain a polynomial-time algorithm for Kempe Distance on paths.

▶ **Theorem 2.** *Kempe Distance on paths can be solved in $O(nk)$ time.*

Proof. We can easily compute l_1 and $f(1, a)$ for each $a \in C$. For $i = 2, 3, \ldots, n$ in this order, we compute l_i and $f(i, a)$ for each $a \in C$ by using (U1)–(U8). Since each value can be computed in constant time, we can compute l_n in $O(nk)$ time. ◀

The algorithm returns l_n, the length of a shortest sequence. We note that we can easily modify our algorithm so that it outputs a sequence of Kempe changes of length l_n.

3 Kempe Distance on Stars

In this section we show that Kempe Distance on stars admits a constant-factor approximation algorithm and the same problem parameterized by the number k of colors is FPT. Furthermore, we give a lower bound instance for the approximation algorithm and show that Kempe Distance on stars is NP-complete. The key concept common to all our results in this section is the notion of a *sorted Kempe-sequence*.

3.1 Sorted Kempe-Sequences

In the following, let $S = (V, E)$ be a star graph with center vertex $c \in V$ and leaves $L = V \setminus \{c\}$. Any Kempe change performed on a k-coloring of S either changes the color of the center vertex or it does not. This observation motivates the following notion of a *sorted Kempe-sequence*. We consider a Kempe-sequence to be sorted, if there is some intermediate coloring γ, such that the color of the center vertex is constant up to γ and after γ it changes in each step. Formally, a Kempe-sequence $\gamma_0, \gamma_1, \ldots, \gamma_\ell$ is *sorted* if there is some index $j \in \{1, 2, \ldots, \ell\}$ and a color $a \in \{1, 2, \ldots, k\}$, such that for $1 \leq i \leq j$, we have $\gamma_i(c) = a$ and for $j \leq i < \ell$, we have $\gamma_i(c) \neq \gamma_{i+1}(c)$. In the following we let $\gamma := \gamma_j$ and call γ the *intermediate coloring*. We first show that we may restrict our attention to sorted Kempe-sequences. We then provide tight bounds for shortest Kempe-sequences from α to γ and γ to β, respectively, which will imply our algorithmic results and our hardness result.

According to the next lemma, for any Kempe-sequence between two k-colorings of S there is a sorted one of at most the same length.

▶ **Lemma 3.** *Let $s := \gamma_0, \gamma_1, \ldots, \gamma_\ell$ be a Kempe-sequence of length ℓ of k-colorings of S. Then there is a sorted Kempe-sequence from γ_0 to γ_ℓ of length at most ℓ.*

Proof. We assume without loss of generality that no two consecutive colorings in s are identical, since if this is the case, we may remove one of the two from s and thus obtain a shorter Kempe-sequence. Suppose furthermore that s is not sorted, that is, there is some index $i \in \{1, 2, \ldots, \ell - 2\}$, such that $\gamma_i(c) \neq \gamma_{i+1}(c)$ and $\gamma_{i+1}(c) = \gamma_{i+2}(c)$. To keep our notation concise, let $a := \gamma_i(c)$ and $a' := \gamma_{i+1}(c)$ be the color of c in γ_i and γ_{i+1}, respectively. Since $\gamma_{i+1} \neq \gamma_{i+2}$, there is a unique leaf $v \in L$, such that $\gamma_{i+1}(v) \neq \gamma_{i+2}(v)$. We let $b := \gamma_{i+1}(v)$ and $b' := \gamma_{i+2}(v)$ be the color of the leaf v in γ_{i+1} and γ_{i+2}, respectively. By assumption we have $a \neq a'$ and $b \neq b'$. Furthermore, since $\gamma_{i+1}(c) = \gamma_{i+2}(c)$, we have that $b' \neq a'$. Finally, since γ_{i+1} is a k-coloring, we have $a' \neq b$.

We show that there is a Kempe-sequence $\gamma_0, \gamma_1, \ldots, \gamma_i, \gamma', \gamma_{i+2}, \ldots, \gamma_\ell$, such that $\gamma_i(c) = \gamma'(c)$. By applying this argument iteratively, we obtain a sorted Kempe-sequence between γ_0 and γ_ℓ of length at most ℓ. We consider two main cases.

Case 1: $\gamma_i(v) = \gamma_{i+1}(v)$

It remains to consider the two subcases $a = b'$ and $a \neq b'$. In the first subcase we have $\gamma_i(c) = a$ and $\gamma_{i+1}(c) = \gamma_{i+2}(c) = a'$, as well as $\gamma_i(v) = \gamma_{i+1}(v) = b$ and $\gamma_{i+2}(v) = a = b'$. So in this case we may first recolor v to color a' and then c to b' using Kempe changes. That is, we let the k-coloring γ' be given by $\gamma'(u) := \gamma_i(u)$ for each vertex $u \in V \setminus \{v\}$ and $\gamma'(v) := a'$. On the other hand, if $a \neq b'$ then a, a', b, b' are distinct, so we may choose $\gamma'(u) := \gamma_i(u)$ for $u \in V \setminus \{v\}$ and $\gamma'(v) := b'$. That is, we can reach γ' from γ_i by changing the color of v to b' and γ_{i+2} from γ' by changing the color of the center vertex c from a to a'.

Case 2: $\gamma_i(v) \neq \gamma_{i+1}(v)$

Recall that c receives different colors in γ_i and γ_{i+1}. Therefore, if v receives different colors in γ_i and γ_{i+1}, then we must have $\gamma_i(v) = a'$ and $\gamma_{i+1}(v) = a = b$. Furthermore, if $a = b'$ then v receives the same color in γ_{i+1} and γ_{i+2}, which contradicts our assumptions. Hence, it remains to consider the case that $a \neq b'$. We let $\gamma'(u) := \gamma_i(u)$ for $u \in V \setminus \{v\}$ and $\gamma'(v) := b'$ and observe that γ' is a k-coloring of S. By construction, the k-coloring γ' can be obtained from γ_i by performing a Kempe change that alters the color of v to b'. Then, γ_{i+2} can be obtained from γ' by a Kempe change that alters the color of c from a to a'. ◀

Note that a Kempe change that alters the color of the center vertex c, say, from color a to color b, also changes the color of every leaf of color b to color a. Let us fix two k-colorings α and β of the star S and let γ be the intermediate coloring of a sorted Kempe-sequence from α to β. The next lemma establishes that the color classes of an intermediate coloring γ agree with those of β.

▶ **Lemma 4** (∗). *Let $u, v \in L$ be two leaves of S. Then $\beta(u) = \beta(v)$ if and only if $\gamma(u) = \gamma(v)$.*

We will show below that, given some coloring γ whose color classes agree with β, we can find efficiently a corresponding shortest sorted Kempe-sequence from α to β via γ. Hence, in the light of Lemma 3, the task of finding a shortest Kempe-sequence from α to β reduces to the task of finding among such colorings one whose corresponding sorted Kempe-sequence is shortest. For this purpose, from the two k-colorings α and β of S, we construct an edge-weighted bipartite auxiliary graph and show that any sorted Kempe-sequence from α to β corresponds to a one-sided perfect matching in this graph. Furthermore, we determine the bounds on the length of a sorted Kempe-sequence from α to β in terms of the weight of the corresponding matching and structure of the graph it induces on the set of colors. Let $G_{\alpha\beta}$ be a complete bipartite graph on the vertex sets $([k] \setminus \{\alpha(c)\}, \beta(L))$. The weight w_{ij} of an edge $(i, j) \in E(G_{\alpha\beta})$ is given by

$$w_{ij} := |\alpha^{-1}(i) \cap \beta^{-1}(j)| \ .$$

In the following, let γ be a k-coloring of S whose color classes agree with those of β. That is the color classes of γ and β induce the same partition of the vertex set of S, but the parts may receive different colors in β and γ. Let $M := \{(i, j) \in E(G_{\alpha\beta}) \mid \gamma^{-1}(i) = \beta^{-1}(j)\}$. By Lemma 4, the set M is a $\beta(L)$-perfect matching of $G_{\alpha\beta}$. Hence, any sorted Kempe-sequence

from α to β gives a $\beta(L)$-perfect matching of $G_{\alpha\beta}$. Also, each $\beta(L)$-perfect matching of $G_{\alpha\beta}$ corresponds to an intermediate coloring of a sorted Kempe-sequence from α to β.

We now show that, given γ, we can compute in polynomial time a shortest Kempe-sequence from α to β via γ.

▶ **Proposition 5.** *Let γ be a k-coloring of S whose color classes agree with those of β. Then there is a polynomial-time algorithm that computes a shortest sorted Kempe-sequence from α to β via γ.*

The next two lemmas give tight bounds for i) the length of a Kempe-sequence from α to γ that does not alter the color of the center vertex and ii) the length of a Kempe-sequence from γ to β such that the color of the center vertex is altered in each step. Since the proofs of the two lemmas are constructive and lead to polynomial-time algorithms, they imply Proposition 5.

▶ **Lemma 6** (∗). *The length of a shortest Kempe-sequence from α to γ such that the color of the center vertex is constant is $|L| - w(M)$.*

Proof (sketch). Observe that the number of leaves $u \in L$ such that $\alpha(u) \neq \gamma(u)$ is a lower bound on the length of a Kempe-sequence from α to γ that does not alter the color of the center vertex. By performing for each leaf u such that $\alpha(u) \neq \gamma(u)$ a Kempe change that assigns to u the color $\gamma(u)$, we obtain a shortest Kempe-sequence from α to γ of the claimed length that does not alter the color of the center vertex. ◀

It remains to bound the length of a shortest Kempe-sequence from γ to β such that the color of the center vertex changes in each step. Note that matching M in $G_{\alpha\beta}$ obtained from γ defines at most one *successor* $M(u) \in \beta(L)$ for each color $u \in [k] \setminus \{\gamma(c)\}$, where c is the center vertex of the star S. By the construction of $G_{\alpha\beta}$, this partial successor map gives rise to a set $\mathcal{C}$ of cycles and a set $\mathcal{P}$ paths on the set $\{1, 2, \ldots, k\}$ of colors. To obtain a desired Kempe-sequence from γ to β, for each item $Z \in \mathcal{P} \cup \mathcal{C}$, the color classes corresponding to the vertices $V(Z)$ need to be altered one-by-one by changing the color of the center vertex.

▶ **Lemma 7** (∗). *The length of a shortest Kempe-sequence from γ to β such that the color of the center vertex changes in each step is*

$$\left(\sum_{Z \in \mathcal{C} \cup \mathcal{P}} |E(Z)| + 1\right) - |\{\gamma(c)\} \cap \beta(L)\}| + 1 - |\{\beta(c)\} \cap \gamma(L)\}| \ .$$

Proof (sketch). By Lemma 4 it remains to assign the correct colors (given by β) to the color classes of γ. We observe that for any cycle $C \in \mathcal{C}$, since the color of the center vertex is not in $V(C)$, at least $|E(C)| + 1$ Kempe changes altering the color of the center vertex are required in order to assign to each leaf $u \in L$ with $\gamma(u) \in V(C)$ the color $\beta(u)$. By a similar observation, each path $P \in \mathcal{P}$ requires at $|E(P)| + 1$ Kempe changes if the center vertex has $\gamma(c) \notin V(P)$ and $|E(P)|$ Kempe changes otherwise. ◀

3.2 Fixed-parameter and Approximation Algorithms

Based on the insights about sorted Kempe-sequences from Section 3.1 we show that Kempe Distance on stars is fixed-parameter tractable, where the parameter is the number k of available colors. The correspondence between sorted Kempe-sequences and matchings in the auxiliary graph $G_{\alpha\beta}$ defined in Section 3.1, together with Proposition 5, yields the following FPT result.

▶ **Theorem 8** (∗). *Kempe Distance on stars can be decided in time* $O(k! \cdot \mathsf{poly}(k) \cdot n)$*, where* k *is the number of available colors and* n *is the size of the instance.*

Proof (sketch). Let $(S, k, \ell, \alpha, \beta)$ be an instance of Kempe Distance on stars. Let L be the set of leaves of the star graph S. Each intermediate coloring of a sorted Kempe-sequence corresponds to a $\beta(L)$-perfect matching in the graph $G_{\alpha\beta}$ from Section 3.1. This graph can be constructed in time $O(\mathsf{poly}(k) \cdot n)$ by iterating over the leaves of G and keeping track of the source and target colors. For each $\beta(L)$-perfect matching M of $G_{\alpha\beta}$ we obtain in time $O(n)$ the length of a shortest Kempe-sequences according to the proofs of lemmas 6 and 7. If the sum of the two lengths is at most ℓ for some matching output Yes, otherwise output No. ◀

Note that for a given instance of Kempe Distance, a shortest Kempe-sequence can be obtained in a straight-forward way by turning the constructive proofs of Lemmas 6 and 7 into an algorithm. For two k-colorings α and β of a graph, we let $\alpha \triangle \beta := \{i \in \{1, 2, \ldots, k\} \mid \alpha^{-1}(i) \neq \beta^{-1}(i)\}$ be the set of colors on which the color classes of α and β are different. Since changing the color of any leaf u that has the same source and target color results in at least two additional Kempe changes, any shortest Kempe-sequence from α to β only involves colors in $\alpha \triangle \beta$. This observation implies the following corollary.

▶ **Corollary 9.** *Kempe Distance on stars is FPT in the number of color classes on which the two colorings differ.*

For an instance I of Kempe Distance on stars, we denote by $\mathrm{OPT}(I)$ the smallest value t, such that α and β admit a Kempe-sequence of length at most t. Note that $\mathrm{OPT}(I)$ is at most the order of input graph. We show that for a maximum-weight matching M of $G_{\alpha\beta}$, the length of a sorted Kempe-sequence with intermediate coloring γ_M gives a constant-factor approximation.

▶ **Theorem 10.** *There is a polynomial-time algorithm that, given an instance* I *of Kempe Distance on stars with* k*-colorings* α *and* β*, computes a Kempe-sequence from* α *to* β *of length at most* $4 \cdot \mathrm{OPT}(I) + 1$*.*

Proof. Let $I = (S, \alpha, \beta, k, \ell)$ be an instance of Kempe Distance on stars and let $G_{\alpha\beta} = (A + B, E)$ be the bipartite graph obtained from the instance as in Section 3.1. We denote by L the set of leaves of the graph S. Let τ^* be a shortest Kempe-sequence from α to β of length $\mathrm{OPT}(I)$. By Lemma 3, we may assume that τ^* is sorted, so there is a matching M^* of $G_{\alpha\beta}$, such that τ^* is composed of a Kempe-sequence τ_1^* from α to an intermediate coloring γ_{M^*} and a Kempe-sequence τ_2^* from γ_{M^*} to β. Let M be a maximum-weight matching of $G_{\alpha\beta}$ and let τ_1 (resp., τ_2) be the Kempe-sequence from α to γ_M (resp., from γ_M to β) obtained according to Lemma 6 (resp., Lemma 7). We show that τ has length at most $4 \cdot \mathrm{OPT}(I) + 1$.

By lemmas 4 and 6, the τ_1 has minimal length among all Kempe-sequences from α to a coloring γ' such that for any two leaves $u, v \in L$, we have $\gamma'(u) = \gamma'(v)$ if and only if $\beta(u) = \beta(v)$. So the length of τ_1 is at most $\mathrm{OPT}(I)$. It will be convenient in the remainder of this proof to consider Kempe-sequences also to be sequences of Kempe changes.

It remains to bound the length of τ_2. Let $\mathcal{C}$ (resp., $\mathcal{P}$) be the set of cycles (resp., paths) on $\{1, 2, \ldots, k\}$ given by the successor map M. Let $j \in \beta(L)$ be a target color of some leaf of S and suppose that there is at least one Kempe change with target color j in τ_2. Then j is a vertex of an item in $\mathcal{C} \cup \mathcal{P}$ (relative to the matching M). We call a color $j \in \beta(L)$ *deficient* if $j \in V(Z)$ for some $Z \in \mathcal{C} \cup \mathcal{P}$ and τ^* contains no Kempe change with target color j.

▷ Claim 11. The only edge with positive weight incident to a deficient vertex j in $G_{\alpha\beta}$ is (j, j). Furthermore, $(j, j) \in M^*$.

Proof of Claim. Each vertex $j \in \beta(L)$ of $G_{\alpha\beta}$ has at least one incident edge with positive weight; otherwise no vertex has target color j implying that $j \notin \beta(L)$. Since, by assumption, there is no Kempe change with target color j in τ_2^*, we have that $(j, j) \in M^*$. Furthermore, we have that each edge (i, j) with $i \neq j$ has weight zero in $G_{\alpha\beta}$; otherwise there is a Kempe change in τ_1^* with target color j. ◁

We may ignore the colors $j \in \beta(L)$ such that $(j, j) \in M$, since no Kempe change in τ_2 has target color j. So let us assume that $\beta(L)$ contains no such colors. Furthermore, let d be the number of deficient colors in $\beta(L)$. We distinguish two cases.

Case 1: $d \leq |\beta(L)|/2$.

For each non-deficient color in $\beta(L)$, the Kempe-sequence τ^* contains at least one Kempe change with target color j. Therefore, $\mathrm{OPT}(I) \geq |\beta(L)|/2$. For each deficient color $j \in \beta(L)$, the corresponding vertex j on other side of $G_{\alpha\beta}$ is covered by M, since otherwise M does not have maximum weight. Therefore, each path or cycle in $\mathcal{C} \cup \mathcal{P}$ contains at least two vertices. By Lemma 7, the Kempe-sequence τ_2 has length at most $(3|\beta(L)|/2) + 1 \leq 3\,\mathrm{OPT}(I) + 1$.

Case 2: $d > |\beta(L)|/2$.

Consider the graph with edges $H := (V(G_{\alpha\beta}), M \triangle M^*)$. Since M and M^* are $\beta(L)$-perfect, each component of H has an even number of edges. Therefore, the graph H is a disjoint union of even paths $\mathcal{P}'$ and cycles $\mathcal{C}'$. Let $Z \in \mathcal{P}' \cup \mathcal{C}'$ and consider any deficient vertex j of Z and the two edges $(j, j) \in M^*$ and $(i, j) \in M$ incident to it. By Claim 11, we have that $w_{jj} > 0$ and $w_{ij} = 0$. On the other hand, since M has maximum weight, we have $w(M \cap E(Z)) \geq w(M^* \cap E(Z))$. Therefore, the weight of the edges in $M \cap E(Z)$ exceeds that of the edges in $M^* \cap E(Z)$ by at least the number of deficient vertices of Z. Now observe that for each edge $(i, j) \in M \setminus M^*$, there are at least w_{ij} Kempe changes in τ_1^* due to Lemma 4. We conclude that τ_1^* has length at least $d \geq |\beta(L)|/2 + 1$. By again considering the worst-case length of τ_2 according to Lemma 7, we bound the length of the Kempe-sequence τ by

$$3\,\mathrm{OPT}(I) \geq 3|\tau_1^*| \geq 3|\beta(L)|/2 + 1 \geq |\tau_2|$$

By combining the bounds for τ_1 and τ_2, we obtain that the Kempe-sequence τ has length at most $4\,\mathrm{OPT}(I) + 1$. ◀

We conclude this subsection by showing that our analysis in the proof of Theorem 10 is almost tight. For this purpose, consider the instance of Kempe Distance on stars and the corresponding graph $G_{\alpha\beta}$ shown in Figure 2. Note that just one Kempe change is needed to transform the source coloring into the target coloring. Both perfect matchings of the shown graph $G_{\alpha\beta}$ have weight 1, so the approximation algorithm may select the one consisting of the two crossing edges. This yields a transformation that first recolors the vertex with source and target color 1 to color 2. It remains to permute the color classes to obtain the target coloring; the permutation of the colors is a cycle of length two. By Lemma 7, takes precisely three Kempe changes to reach the target coloring. Hence, the algorithm outputs a 4-approximate solution in the worst case.

(a) (b)

Figure 2 Instance of Kempe Distance on stars (a) and corresponding edge-weighted graph $G_{\alpha\beta}$ (b) for which the approximation algorithm gives a 4-approximate solution. A label $a \to b$ indicates source color a and target color b for the corresponding vertex.

3.3 NP-completeness of Kempe Distance

We complement our polynomial-time algorithms from Section 3.2 by the following result, which we will prove in the remainder of this section.

▶ **Theorem 12.** *Kempe Distance on stars is* NP*-complete.*

Recall that any two k-colorings of a star on n vertices admit a Kempe-sequence of length $O(n)$. Hence, Kempe Distance on stars is in NP. To show NP-hardness, we give a polynomial-time reduction from the problem Hamiltonian Cycle, which asks, whether a given graph contains a simple cycle visiting each vertex. It was shown by Garey, Johnson, and Tarjan that Hamiltonian Cycle remains NP-complete even on planar cubic graphs [11]. In order to show that Kempe Distance on stars is NP-hard, we first reduce Hamiltonian Cycle on cubic graphs to the following minimum-cost permutation problem.

Minimum-Cost Permutation
Input: non-negative weights $w \in \mathbb{Z}_{\geq 0}^{V \times V}$ and number $z \in \mathbb{Z}$
For a permutation π we denote by $c(\pi)$ the number of cycles of a cycle decomposition of π. The *cost* of a permutation π of V is given by $c(\pi) - \sum_{v \in V} w(v, \pi_v)$.
Question: Is there a permutation π of V, such that π has cost at most z?

We then establish that the minimum cost of a permutation corresponds to the length of a shortest Kempe-sequence of a suitable instance of Kempe Distance. Let $G = (V, E)$ be a cubic graph. Consider the weights $w \in \mathbb{Z}^{V \times V}$ given by

$$w_{uv} = \begin{cases} K & \text{if } u \text{ and } v \text{ are adjacent in } G \\ 0 & \text{otherwise,} \end{cases}$$

where K is a suitably large number, say $K = |V|^2$. The following lemma implies that Minimum-Cost Permutation is NP-hard.

▶ **Lemma 13.** *G has a Hamiltonian cycle if and only if there is a permutation π of V, such that π has cost at most $1 - K \cdot |V|$.*

Proof. Let $C = v_1, v_2, \ldots, v_t$ be a Hamiltonian cycle of G, where $t = |V|$. Furthermore, let π be given by $\pi(v_i) := v_{i+1}$ for $1 \leq i < t$ and $\pi(v_t) := v_1$ Since C is a Hamiltonian cycle, the cost π is $1 - K \cdot |V|$.

Now suppose that π is a permutation of V of cost at most $1 - K \cdot |V|$. Then for each $v \in V$ contributes at least $-K$ to the cost and π contains not fixpoints (due to the choice of K). Therefore, each vertex $v \in V$ is on some cycle of π. Since the cost π is at most $1 - K \cdot |V|$ we have that $c(\pi) \leq 1$. It follows from the construction of w that π corresponds Hamiltonian cycle of G. ◀

Note that since w is symmetric, it corresponds to a complete edge-weighted bipartite graph G' on the vertex set (V, V). We construct an instance (S, α, β) of Kempe Distance on stars, such that the corresponding edge-weighted auxiliary graph $G_{\alpha\beta}$ (see Section 3.1) is isomorphic to the weighted graph G'. The star graph S has $3K \cdot |V|$ leaves $\bigcup_{v \in V} \{\ell_1^v, \ell_2^v, \ldots, \ell_{3K}^v\}$. We denote its center vertex by c. The source coloring α is given by $\alpha(\ell_i^v) := v$ for $1 \leq i \leq 3K$ and $v \in V$. For $1 \leq i \leq 3$, let $n_i(v)$ be the three neighbors of v in G in arbitrary order. For $1 \leq i \leq 3K$ and $v \in V$, let

$$\beta(\ell_i^v) := \begin{cases} n_1(v) & \text{if } 1 \leq i \leq K \\ n_2(v) & \text{if } K+1 \leq i \leq 2K \\ n_3(v) & \text{it } 2k+1 \leq i \leq 3K \end{cases}$$

Finally, let $\alpha(c) = \beta(c) = |V| + 1$. It is readily verified that the weighted graph G' is isomorphic to the graph $G_{\alpha\beta}$ obtained from S and the colorings α and β as in Section 3.1. Hence, we may invoke lemmas 6 and 7 and obtain the following relation between minimum-cost permutations and shortest Kempe-sequences from α to β.

▶ **Lemma 14.** *There is a permutation of V of cost at most $1 - K \cdot |V|$ if and only if α and β admit a Kempe-sequence of length at most $|V|(2K+1)+1$.*

Proof. Let π be a permutation of V of cost at most $1 - K \cdot |V|$. Since π corresponds to a perfect matching M of the weighted graph G' and G' is isomorphic to $G_{\alpha\beta}$, we may invoke lemmas 6 and 7 to obtain a sorted Kempe-sequence τ of length $|V|(2K+1)+1$ from α to β via γ_M.

Now let τ be a Kempe-sequence from α to β of length at most $|V|(2K+1)+1$. We show that there is a permutation π of V of cost at most $1 - K \cdot |V|$. We may assume that τ is sorted, so let γ be the intermediate coloring and let M_γ be the corresponding matching of $G_{\alpha\beta}$ (see Section 3.1). The successor map given by the matching M_γ gives rise to a permutation π on V, which in turn induces a set $\mathcal{C}$ of cycles on V; and potentially fixpoints. By lemmas 6 and 7 we have

$$|V|(2K+1)+1 = |\tau| \geq 3K|V| - w(M_\gamma) + c(\pi)$$

Rearranging gives $1 - K|V| + |V| \geq -w(M_\gamma) + c(\pi) = \sum_{v \in V} w(v, \pi_v)$. Since $K > 2|V|$, we have $w(v, \pi_v) = K$ for each $v \in V$. Therefore, by the construction of w, the permutation π has no fixpoints. Hence, by lemmas 6 and 7, we get the following sharper bound on the length of τ.

$$|V|(2K+1)+1 = |\tau| \geq 3K|V| - w(M_\gamma) + |V| + c(\pi)$$

Since $w(v, \pi_v) = K$ for each $v \in V$, we obtain from this inequality that $c(\pi) = 1$. Therefore, the permutation π has cost at most $1 - K \cdot |V|$. ◀

From Lemmas 13 and 14, the NP-hardness of Kempe Distance on stars is immediate. Observe that if we restrict ourselves to elementary recolorings then the problem is tractable.

▶ **Proposition 15** (∗). *There is a polynomial-time algorithm that, given two k-colorings α and β of a star graph, finds, if it exists, a Kempe-sequence of minimal length that certifies the equivalence of α and β under elementary recolorings.*

4 Conclusion

We showed that Kempe Distance on paths admits a polynomial-time algorithm and that the same problem on stars is NP-complete. Furthermore, we show that Kempe Distance on stars is FPT in the number k of colors and it admits a constant-factor approximation algorithm. There are some interesting open questions related to our results.

- Is it possible to generalize the dynamic programming algorithm for Kempe Distance on paths to trees with a bounded number of leaves?
- Does Kempe Distance on stars admit a polynomial-time approximation scheme?
- Does Kempe Distance on stars admit a polynomial kernel?

We conjecture that the answer to the last question is negative.

References

1. Oswin Aichholzer, Wolfgang Mulzer, and Alexander Pilz. Flip distance between triangulations of a simple polygon is NP-complete. *Discrete & computational geometry*, 54(2):368–389, 2015.
2. Marthe Bonamy, Nicolas Bousquet, Carl Feghali, and Matthew Johnson. On a conjecture of mohar concerning Kempe equivalence of regular graphs. *Journal of Combinatorial Theory, Series B*, 135:179–199, 2019. doi:10.1016/j.jctb.2018.08.002.
3. Marthe Bonamy, Marc Heinrich, Takehiro Ito, Yusuke Kobayashi, Haruka Mizuta, Moritz Mühlenthaler, Akira Suzuki, and Kunihiro Wasa. Diameter of colorings under kempe changes. In *Computing and Combinatorics - 25th International Conference, COCOON 2019, Proceedings*, pages 52–64, 2019. doi:10.1007/978-3-030-26176-4_5.
4. Édouard Bonnet, Tillmann Miltzow, and Paweł Rzążewski. Complexity of token swapping and its variants. *Algorithmica*, 80(9):2656–2682, 2018.
5. Paul Bonsma and Luis Cereceda. Finding Paths Between Graph Colourings: PSPACE-Completeness and Superpolynomial Distances. In *MFCS*, volume 4708 of *Lecture Notes in Computer Science*, pages 738–749, 2007.
6. Nicolas Bousquet and Marc Heinrich. A polynomial version of cereceda's conjecture, 2019. arXiv:1903.05619.
7. Luis Cereceda, Jan van den Heuvel, and Matthew Johnson. Finding paths between 3-colorings. *Journal of Graph Theory*, 67(1):69–82, 2011.
8. Sitan Chen, Michelle Delcourt, Ankur Moitra, Guillem Perarnau, and Luke Postle. Improved bounds for randomly sampling colorings via linear programming. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2216–2234. SIAM, 2019.
9. Erik D Demaine, Sarah Eisenstat, and Mikhail Rudoy. Solving the rubik's cube optimally is NP-complete. In *35th Symposium on Theoretical Aspects of Computer Science*, 2018.
10. Carl Feghali, Matthew Johnson, and Daniël Paulusma. Kempe equivalence of colourings of cubic graphs. *European Journal of Combinatorics*, 59:1–10, 2017.
11. M. Garey, D. Johnson, and R. Tarjan. The planar Hamiltonian circuit problem is NP-complete. *SIAM Journal on Computing*, 5(4):704–714, 1976. doi:10.1137/0205049.
12. Oded Goldreich. Finding the shortest move-sequence in the graph-generalized 15-puzzle is NP-hard. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 1–5. Springer, 2011.
13. Jan van den Heuvel. The complexity of change. In Simon R. Blackburn, Stefanie Gerke, and Mark Wildon, editors, *Surveys in Combinatorics 2013*, pages 127–160. Cambridge University Press, 2013.
14. Matthew Johnson, Dieter Kratsch, Stefan Kratsch, Viresh Patel, and Daniël Paulusma. Finding shortest paths between graph colourings. *Algorithmica*, 75(2):295–321, 2016.
15. Michel Las Vergnas and Henri Meyniel. Kempe classes and the Hadwiger Conjecture. *Journal of Combinatorial Theory, Series B*, 31(1):95–104, 1981. doi:10.1016/S0095-8956(81)80014-7.

16 Anna Lubiw and Vinayak Pathak. Flip distance between two triangulations of a point set is NP-complete. *Computational Geometry*, 49:17–23, 2015.

17 Henry Meyniel. Les 5-colorations d'un graphe planaire forment une classe de commutation unique. *Journal of Combinatorial Theory, Series B*, 24(3):251–257, 1978. doi:10.1016/0095-8956(78)90042-4.

18 Tillmann Miltzow, Lothar Narins, Yoshio Okamoto, Günter Rote, Antonis Thomas, and Takeaki Uno. Approximation and Hardness of Token Swapping. In Piotr Sankowski and Christos Zaroliagis, editors, *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 66:1–66:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.66.

19 Bojan Mohar. Kempe equivalence of colorings. In Adrian Bondy, Jean Fonlupt, Jean-Luc Fouquet, Jean-Claude Fournier, and Jorge L. Ramírez Alfonsín, editors, *Graph Theory in Paris*, Trends in Mathematics, pages 287–297. Birkhäuser Basel, 2007. doi:10.1007/978-3-7643-7400-6_22.

20 Bojan Mohar and Jesús Salas. A new Kempe invariant and the (non)-ergodicity of the Wang–Swendsen–Koteckỳ algorithm. *Journal of Physics A: Mathematical and Theoretical*, 42(22):225204, 2009.

21 Bojan Mohar and Jesús Salas. On the non-ergodicity of the swendsen–wang–koteckỳ algorithm on the kagomé lattice. *Journal of Statistical Mechanics: Theory and Experiment*, 2010(05):P05016, 2010.

22 Amer E Mouawad, Naomi Nishimura, Vinayak Pathak, and Venkatesh Raman. Shortest reconfiguration paths in the solution space of boolean formulas. *SIAM Journal on Discrete Mathematics*, 31(3):2185–2200, 2017.

23 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4:52), 2018.

24 Daniel Ratner and Manfred K Warmuth. Finding a shortest solution for the $n \times n$ extension of the 15-puzzle is intractable. In *AAAI*, pages 168–172, 1986.

25 Eric Vigoda. Improved bounds for sampling colorings. *Journal of Mathematical Physics*, 41(3):1555–1569, 2000.

26 Katsuhisa Yamanaka, Erik D. Demaine, Takashi Horiyama, Akitoshi Kawamura, Shin-ichi Nakano, Yoshio Okamoto, Toshiki Saitoh, Akira Suzuki, Ryuhei Uehara, and Takeaki Uno. Sequentially swapping colored tokens on graphs. *Journal of Graph Algorithms and Applications*, 23(1):3–27, 2019. doi:10.7155/jgaa.00482.

27 Katsuhisa Yamanaka, Takashi Horiyama, David Kirkpatrick, Yota Otachi, Toshiki Saitoh, Ryuhei Uehara, and Yushi Uno. Swapping colored tokens on graphs. In *Workshop on Algorithms and Data Structures*, pages 619–628. Springer, 2015.

Elimination Distances, Blocking Sets, and Kernels for Vertex Cover

Eva-Maria C. Hols
Department of Computer Science, Humboldt-Universität zu Berlin, Germany
eva-maria.hols@fkie.fraunhofer.de

Stefan Kratsch
Department of Computer Science, Humboldt-Universität zu Berlin, Germany
kratsch@informatik.hu-berlin.de

Astrid Pieterse
Department of Computer Science, Humboldt-Universität zu Berlin, Germany
astrid.pieterse@informatik.hu-berlin.de

Abstract

The Vertex Cover problem plays an essential role in the study of polynomial kernelization in parameterized complexity, i.e., the study of provable and efficient preprocessing for NP-hard problems. Motivated by the great variety of positive and negative results for kernelization for Vertex Cover subject to different parameters and graph classes, we seek to unify and generalize them using so-called blocking sets. A blocking set is a set of vertices such that no optimal vertex cover contains all vertices in the blocking set, and the study of minimal blocking sets played implicit and explicit roles in many existing results.

We show that in the most-studied setting, parameterized by the size of a deletion set to a specified graph class $\mathcal{C}$, bounded minimal blocking set size is necessary but not sufficient to get a polynomial kernelization. Under mild technical assumptions, bounded minimal blocking set size is showed to allow an essentially tight efficient reduction in the number of connected components.

We then determine the exact maximum size of minimal blocking sets for graphs of bounded elimination distance to any hereditary class $\mathcal{C}$, including the case of graphs of bounded treedepth. We get similar but not tight bounds for certain non-hereditary classes $\mathcal{C}$, including the class $\mathcal{C}_{\text{LP}}$ of graphs where integral and fractional vertex cover size coincide. These bounds allow us to derive polynomial kernels for Vertex Cover parameterized by the size of a deletion set to graphs of bounded elimination distance to, e.g., forest, bipartite, or $\mathcal{C}_{\text{LP}}$ graphs.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Vertex Cover, kernelization, blocking sets, elimination distance, structural parameters

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.36

Related Version A full version of the paper is available at `https://arxiv.org/abs/1905.03631`.

Funding *Eva-Maria C. Hols*: Supported by DFG Emmy Noether-grant (KR 4286/1).
Astrid Pieterse: Supported by NWO Gravitation grant "Networks".

1 Introduction

In the Vertex Cover problem we are given an undirected graph $G = (V, E)$ and an integer k; the question is whether there exists a set $S \subseteq V$ of at most k vertices such that each edge of G is incident with a vertex of S, or, in other words, such that $G - S$ is an independent set. Despite Vertex Cover being NP-complete, it is known that there are efficient preprocessing algorithms that reduce any instance (G, k) to an equivalent instance with $\mathcal{O}(k^2)$ or even at most $2k$ vertices (and size polynomial in k). The existence or non-existence of such preprocessing routines for NP-hard problems has been studied intensively in

the field of parameterized complexity under the term *polynomial kernelization*,[1] and Vertex Cover has turned out to be one of the most fruitful research subjects with a variety of upper and (conditional) lower bounds subject to different parameters (see, e.g., [2]).

In the present work, we seek to unify and generalize existing results by using so-called *blocking sets*. A blocking set in a graph $G = (V, E)$ is any set $Y \subseteq V$ that is not a subset of any minimum cardinality vertex cover of G, e.g., the set V itself is always a blocking set. Of particular interest are *minimal* blocking sets, i.e., those that are minimal under set inclusion. Several graph classes have constant upper bounds on the size of minimal blocking sets, e.g., in any forest (or even in any bipartite graph) every minimal blocking set has size at most two. On the other hand, even restrictive classes like outerplanar graphs have unbounded minimal blocking set size, i.e., for each d there is a graph in the class with a minimal blocking set of size greater than d. As a final example, cliques are the unique graphs for which V is the only (minimal) blocking set because all optimal vertex covers have form $V \setminus \{v\}$ for any $v \in V$; in particular, any graph class containing all cliques has unbounded minimal blocking set size.

For ease of reading, the introduction focuses mostly on hereditary graph classes, i.e., those closed under vertex deletion; the later sections give more general results modulo suitable technical conditions, where appropriate. Apart from that, throughout the paper, we restrict study to graph classes $\mathcal{C}$ that are *robust*, that is, they are closed under disjoint union and under deletion of connected components. In other words, a graph G is in $\mathcal{C}$ if and only if all of its connected components belong to $\mathcal{C}$. Most graph classes studied in the context of kernels for Vertex Cover are robust, and a large number of them are also hereditary. A particular non-hereditary graph class of interest for us is the class $\mathcal{C}_{\text{LP}}$ of graphs whose minimum vertex cover size equals the minimum size of a fractional vertex cover (denoted LP as it is also the optimum solution value for the vertex cover LP relaxation).

1.1 Blocking sets and kernels for Vertex Cover

Most known polynomial kernelizations for Vertex Cover are for parameterization by the vertex deletion distance to some fixed hereditary graph class $\mathcal{C}$ that is also robust, e.g., for $\mathcal{C}$ being the class of forests [13], graphs of maximum degree one or two [18], pseudoforests [8] (each component has at most one cycle), bipartite graphs [17], d-quasi forests/bipartite graphs [10] (at most d vertex deletions per component away from being a forest/bipartite), cluster graphs of bounded clique size [18], or graphs of bounded treedepth [2]. Concretely, the input is of form (G, k, X), asking whether G has a vertex cover of size at most k, where $X \subseteq V$ such that $G - X \in \mathcal{C}$; the size $\ell = |X|$ of the *modulator* X is the parameter. Blocking sets have been implicitly or explicitly used for most of these results and we point out that all the mentioned classes have bounded minimal blocking set size.

As our first result, we show that this is not a coincidence: If $\mathcal{C}$ is closed under disjoint union (or, more strongly, if $\mathcal{C}$ is robust) then bounded size of minimal blocking sets in graphs of $\mathcal{C}$ is necessary for a polynomial kernel to exist (Section 3.1). Moreover, the maximum size of minimal blocking sets in $\mathcal{C}$ yields a lower bound for the possible kernel size.

▶ **Theorem 1.** *Let $\mathcal{C}$ be a graph class that is closed under disjoint union. If $\mathcal{C}$ contains any graph with a minimal blocking set of size d then Vertex Cover parameterized by the size of a modulator X to $\mathcal{C}$ does not have a kernelization of size $\mathcal{O}(|X|^{d-\varepsilon})$ for any $\varepsilon > 0$ unless* $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ *and the polynomial hierarchy collapses.*

[1] A polynomial kernelization is an efficient algorithm that given any instance with parameter value ℓ returns an equivalent instance of size polynomial in ℓ. In the initial Vertex Cover example we have parameter $\ell = k$. For more information on the topic, see [4, 7]

To the best of our knowledge, this theorem captures all known kernel lower bounds for Vertex Cover parameterized by deletion distance to any union-closed graph class $\mathcal{C}$, e.g., ruling out polynomial kernels for $\mathcal{C}$ being the class of mock forests (each vertex is in at most one cycle) [8], outerplanar graphs [12], or any class containing all cliques [1]; and getting kernel size lower bounds for graphs of bounded treedepth [2] or cluster graphs of bounded clique size [18]. To get lower bounds of this type, it now suffices to prove (or observe) that $\mathcal{C}$ has large or even unbounded minimal blocking set size.

It is natural to ask whether the converse holds, i.e., whether a bound on the minimal blocking set size directly implies the existence of a polynomial kernelization. Unfortunately, we show that this does not hold in a strong sense: There is a class $\mathcal{C}$ such that all graphs in $\mathcal{C}$ have minimal blocking sets of size one, but there is no polynomial kernelization (Section 3.2). More strongly, solving Vertex Cover on $\mathcal{C}$ is not in $\mathsf{RP} \supseteq \mathsf{P}$ unless $\mathsf{NP} = \mathsf{RP}$.

▶ **Theorem 2.** *There exists a graph class $\mathcal{C}$ such that all graphs in $\mathcal{C}$ have minimal blocking set size one and such that Vertex Cover on $\mathcal{C}$ is not solvable in polynomial time, unless $\mathsf{NP} = \mathsf{RP}$.*

In light of this result, one could ask what further assumptions on $\mathcal{C}$, apart from the necessity of bounded minimal blocking set size, are required to allow for polynomial kernels. Clearly, polynomial-time solvability of Vertex Cover on the class $\mathcal{C}$ is necessary and (as we implicitly showed) not implied by $\mathcal{C}$ having bounded blocking set size. If, slightly stronger, we require that blocking sets in graphs of $\mathcal{C}$ can be efficiently recognized[2] then we show that there is an efficient algorithm that reduces the number of components of $G - X$ for any instance (G, k, X) of Vertex Cover parameterized by deletion distance to $\mathcal{C}$ to $\mathcal{O}(|X|^d)$ (Section 3.3). This is a standard opening step for kernelization and can be followed up by shrinking and bounding the size of those components. Note that this requires that deletion of any component yields a graph in $\mathcal{C}$ (e.g., implied by $\mathcal{C}$ being robust), which here is covered already by $\mathcal{C}$ being hereditary.

▶ **Theorem 3.** *Let $\mathcal{C}$ be any hereditary graph class with minimal blocking set size d on which Vertex Cover can be solved in polynomial time. There is an efficient algorithm that given (G, k, X) such that $G - X \in \mathcal{C}$ returns an equivalent instance (G', k', X) such that $G' - X \in \mathcal{C}$ has at most $\mathcal{O}(|X|^d)$ connected components.*

We point out that the number $\mathcal{O}(|X|^d)$ of components is essentially tight (assuming that $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$) because the lower bound underlying Theorem 1 creates instances where components have a constant $c = c(d)$ many vertices. Reducing to $\mathcal{O}(|X|^{d-\varepsilon})$ components, for any $\varepsilon > 0$, would violate the kernel size lower bound.

1.2 Minimal blocking set size relative to elimination distances

Recently, Bougeret and Sau [2] presented a polynomial kernelization for Vertex Cover parameterized by the size of a modulator X such that $G - X$ has treedepth at most d; here d is a fixed constant and the degree of the polynomial in the kernel size depends exponentially on d. To get the kernelization, they prove (in different but equivalent terms) that the size of any minimal blocking set in a graph of treedepth d is at most 2^d, and they give a lower bound of 2^{d-3}. As our first result here, we determine the exact maximum size of minimal blocking sets in graphs of treedepth d (see below, and see Section 4 for all these results).

[2] This condition holds for all hereditary $\mathcal{C}$ on which Vertex Cover can be solved in polynomial time: Given $G = (V, E)$ and $Y \subseteq V$ it suffices to compute solutions for G and $G - Y$. Clearly, the set Y is a blocking set if and only if $\mathrm{OPT}(G) < \mathrm{OPT}(G - Y) + |Y|$.

Bulian and Dawar [3] introduced the notion of elimination distance to a class $\mathcal{C}$, generalizing treedepth, which corresponds to elimination to the empty graph (see Section 2 for a formal definition). This is defined in the same way as treedepth except that all graphs from $\mathcal{C}$ get value 0 rather than just the empty graph. (Note that here it is convenient that $\mathcal{C}$ is robust because the definition assigns value 0 to disjoint unions of graphs from $\mathcal{C}$.) Intuitively, elimination distance to $\mathcal{C}$ can be pictured as having a tree-like deletion of vertices (as for treedepth) but being allowed to stop when the remaining connected components belong to $\mathcal{C}$ (rather than continuing to the empty graph). For hereditary $\mathcal{C}$, we determine the exact maximum size of minimal blocking sets in graphs of elimination distance at most d to $\mathcal{C}$, denoted $\beta_{\mathcal{C}}(d)$, depending on the maximum minimal blocking set size $b_{\mathcal{C}}$ in the class $\mathcal{C}$.

▶ **Theorem 4.** *Let $\mathcal{C}$ be a robust hereditary graph class where $b_{\mathcal{C}}$ is bounded. For every integer $d \geq 1$ it holds that*

$$\beta_{\mathcal{C}}(d) = \begin{cases} 2^{d-1}+1 & \text{, if } b_{\mathcal{C}} = 1, \\ (b_{\mathcal{C}}-1)2^d+1 & \text{, if } b_{\mathcal{C}} \geq 2. \end{cases}$$

The lower bound holds as well for any non-hereditary class $\mathcal{C}$ but we only get a slightly weaker upper bound for such classes (and also require a further technical condition called f-robustness). In particular, we get such an upper bound for the class $\mathcal{C}_{\text{LP}}$ mentioned above. Note that if $\mathcal{C}$ has unbounded minimal blocking set size then the same is true for graphs of any bounded elimination distance to $\mathcal{C}$ (irrespective of $\mathcal{C}$ being hereditary or not).

The bound for graphs of treedepth at most d is included in the theorem by using that having treedepth at most d is equivalent to having elimination distance at most $d-1$ to the class of independent sets (i.e., graphs of treedepth one), for which all minimal blocking sets have size 1. Concretely, for treedepth d we get $\beta(d) = 1$ for $d = 1$ and $\beta(d) = 2^{d-2}+1$ for $d \geq 2$.

1.3 Some consequences for kernels for Vertex Cover

Our bounds for the minimal blocking set size relative to elimination distances allow us to generalize and combine previous polynomial kernelization results for Vertex Cover. We state this explicitly for elimination distances to hereditary graph classes.

▶ **Theorem 5.** *Let $\mathcal{C}$ be a hereditary and robust graph class for which $b_{\mathcal{C}}$ is bounded, such that Vertex Cover has a (randomized) polynomial kernelization parameterized by the size of a modulator to $\mathcal{C}$. Then Vertex Cover also has a (randomized) polynomial kernelization parameterized by the size of a modulator to graphs of bounded elimination distance to $\mathcal{C}$.*

As an example, this combines known polynomial kernels relative (to the size of) modulators to a forest [13] resp. to graphs of bounded treedepth [2] to polynomial kernels relative to a modulator to graphs of bounded forest elimination distance. Similarly, the randomized polynomial kernel for Vertex Cover parameterized by a modulator to bipartite graphs is generalized to a modulator to graphs of bounded bipartite elimination distance. The approach to this result (also in the non-hereditary case) uses our bounds for minimal blocking set size relative to elimination distances and, apart from that, is inspired by the result of Bougeret and Sau [2]. Intuitively, these kernels are obtained by suitable reductions to the known kernelizable cases, and thus carry over their properties (e.g., being deterministic or randomized).

As an explicit example for the non-hereditary case, we state a new kernelization result relative to the size of a modulator to the class of graphs of bounded elimination distance to $\mathcal{C}_{\text{LP}}$, i.e., bounded elimination distance to graphs where optimum vertex cover size equals optimum fractional vertex cover size.

▶ **Theorem 6.** *Vertex Cover admits a randomized polynomial kernel parameterized by the size of a modulator to graphs that have bounded elimination distance to* $\mathcal{C}_{\text{LP}}$.

This subsumes several polynomial kernels for Vertex Cover (except for their size bounds).

2 Preliminaries

For $n \in \mathbb{N}$ we use $[n]$ to denote $\{1, \ldots, n\}$. We use standard graph notation [6]. We say that a graph class $\mathcal{C}$ is *robust* if $\mathcal{C}$ is closed under disjoint union and deletion of connected components. A set $S \subseteq V(G)$ is a *vertex cover* of a graph G, if for each edge $e \in E(G)$, at least one of its endpoints is contained in S. We will use $\text{OPT}(G)$ to denote the size of a minimum vertex cover of G. The linear program relaxation for Vertex Cover for G is

$$\text{LP}(G) = \min \left\{ \sum_{v \in V(G)} x_v \mid \forall \{u,v\} \in E(G) : x_u + x_v \geq 1 \wedge \forall v \in V(G) : 0 \leq x_v \leq 1 \right\}.$$

It is well known that there is always an optimal feasible solution such that $x_v \in \{0, \frac{1}{2}, 1\}$ for all $v \in V(G)$. We call a solution for which this holds a *half-integral* solution.

▶ **Definition 7.** *Let $\mathcal{C}$ be a graph class and let G be a graph. We define the* elimination distance to $\mathcal{C}$ *as*

$$\text{ed}_{\mathcal{C}}(G) := \begin{cases} 0 & \text{if } G \in \mathcal{C}, \\ \min_{v \in V(G)} \text{ed}_{\mathcal{C}}(G - \{v\}) + 1 & \text{if } G \notin \mathcal{C} \text{ and } G \text{ is connected}, \\ \max_{i \in [t]} \text{ed}_{\mathcal{C}}(G_i) & \text{if } G \text{ consists of connected components } G_1, \ldots, G_t. \end{cases}$$

The treedepth *of a graph G is simply its elimination distance to the empty graph.*

Let $\mathcal{C}$ be a graph class, let G be a graph and let $X \subseteq V(G)$. We say X is a *$\mathcal{C}$-modulator* if $G - X \in \mathcal{C}$. We say that X is a *$(\mathcal{C}, d)$-modulator* if $\text{ed}_{\mathcal{C}}(G - X) \leq d$. When considering Vertex Cover parameterized by the size of a $\mathcal{C}$-modulator or a $(\mathcal{C}, d)$-modulator, we will assume that this modulator is given on input. As such, inputs to the problem are triplets (G, k, X) such that X is a modulator and the problem is to decide whether G has a vertex cover of size at most k.

For a graph class $\mathcal{C}$, let $\mathcal{C} + c$ be the graph class consisting of all graphs that have a $\mathcal{C}$-modulator of size at most c, i.e., $\mathcal{C} + c := \{G \mid \exists X \subseteq V(G), |X| \leq c \colon G - X \in \mathcal{C}\}$.

▶ **Definition 8.** *Let G be a graph and let $Y \subseteq V(G)$ be a subset of its vertices. We say that Y is a* blocking set *in G if there exists no vertex cover S of G such that $Y \subseteq S$ and $|S| = \text{OPT}(G)$. In other words, there is no optimal vertex cover of G that contains Y. A blocking set Y is* minimal *if no strict subset of Y is also a blocking set.*

Let G be a graph, we use $\beta(G)$ to denote the size of the largest minimal blocking set in G. For a graph class $\mathcal{C}$, let $b_{\mathcal{C}} := \max_{G \in \mathcal{C}} \beta(G)$, let $b_{\mathcal{C}} := \infty$ if the minimal blocking set size of graphs in this graph class can be arbitrarily large. Define $\beta_{\mathcal{C}}(d) := \max\{\beta(G) \mid \text{ed}_{\mathcal{C}}(G) \leq d\}$.

3 Relation between minimal blocking sets and polynomial kernels

In this section, we show relations between the size of minimal blocking sets in $\mathcal{C}$ and kernelization bounds for Vertex Cover parameterized by a $\mathcal{C}$-modulator.

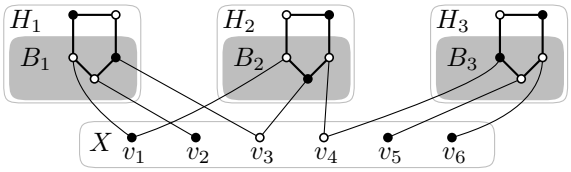


Figure 1 The Vertex Cover instance G' obtained in the proof of Theorem 1, corresponding to instance $(\mathcal{F}, 2)$ with $\mathcal{F} = \{\{v_1, v_2, v_3\}, \{v_1, v_3, v_4\}, \{v_4, v_5, v_6\}\}$. Since $\mathcal{F}$ has a hitting set of size 2, G' has a vertex cover of size $2 + 3 \cdot 3$, indicated in white.

3.1 Polynomial kernel implies a bound on the minimal blocking set size

In this section we prove Theorem 1, showing that if $\mathcal{C}$ is a graph class where minimal blocking sets can have size d, then this gives a kernelization lower bound for Vertex Cover when parameterized by the size of a $\mathcal{C}$-modulator. This generalizes most existing lower bounds for Vertex Cover when parameterized by the size of a modulator to $\mathcal{C}$ for some graph class $\mathcal{C}$ [8, 12, 14, 18]. Under the assumption that NP $\not\subseteq$ coNP/poly, the theorem shows that having bounded blocking set size is necessary to obtain a polynomial kernel in the following sense. For a graph class $\mathcal{C}$ closed under disjoint union, for which Vertex Cover parameterized by a modulator to $\mathcal{C}$ admits a polynomial kernel of size $\mathcal{O}(k^d)$, it must hold that $b_{\mathcal{C}} \leq d$.

▶ **Theorem 1** (★[3]). *Let $\mathcal{C}$ be a graph class that is closed under disjoint union. If $\mathcal{C}$ contains any graph with a minimal blocking set of size d then Vertex Cover parameterized by the size of a modulator X to $\mathcal{C}$ does not have a kernelization of size $\mathcal{O}(|X|^{d-\varepsilon})$ for any $\varepsilon > 0$ unless* NP $\subseteq$ coNP/poly *and the polynomial hierarchy collapses.*

Proof sketch. The lower bound is obtained by a linear-parameter transformation from d-Hitting Set. An input to this problem consists of a set family $\mathcal{F}$ over a universe U and integer k where every set in $\mathcal{F}$ has size exactly d. The problem is to decide whether $\mathcal{F}$ has a hitting set of size at most k. A hitting set is a set $X \subseteq U$ such that for every set $S \in \mathcal{F}$ we have $S \cap X \neq \emptyset$. The lower bound will then follow from the fact that for $d \geq 2$, d-Hitting Set parameterized by the universe size n does not have a kernel of size $\mathcal{O}(n^{d-\varepsilon})$ for any $\varepsilon > 0$ unless NP $\subseteq$ coNP/poly [5, Theorem 2].

Suppose we are given an instance $(\mathcal{F}, k)$ for d-Hitting Set. Choose $H \in \mathcal{C}$ with a minimal blocking set of size d. We construct G' for Vertex Cover starting with vertex set $X := U$ (note $G' - X \in \mathcal{C}$), and adding a copy H_j of graph H for each $S_j \in \mathcal{F}$. The vertices in X that correspond to vertices in S_j are now each connected to a distinct vertex in the size-d minimal blocking set of H_j. This ensures that for any vertex cover S' of G', if none of the vertices from S_j is contained in S', then $|S' \cap H_j| > \mathrm{OPT}(H_j)$. We can use this to show that there exists a minimum vertex cover S' of G' such that $S' \cap X$ corresponds to a hitting set of $\mathcal{F}$. A sketch of the constructed instance $(G', k' := k + m \cdot \mathrm{OPT}(H))$ for Vertex Cover is shown in Figure 1. ◀

3.2 Bounded minimal blocking set size is not sufficient

Now that it is clear that, proving that a graph class has bounded blocking set size is essential towards obtaining a polynomial kernel for Vertex Cover parameterized by the size of a modulator to this graph class, one may wonder whether this condition is also sufficient. It

[3] For statements marked with a (★), the (full) proof can be found in the full version of the paper, which is available at `https://arxiv.org/abs/1905.03631`.

turns out that it is *not*. We show that there exists a graph class $\mathcal{C}$ for which all minimal blocking sets have size 1, for which Vertex Cover is not solvable in polynomial time unless NP = RP. This implies that Vertex Cover is unlikely to be FPT when parameterized by the size of a $\mathcal{C}$-modulator, immediately implying that it does not have a polynomial kernel when parameterized by a $\mathcal{C}$-modulator.

▶ **Theorem 2.** *There exists a graph class $\mathcal{C}$ such that all graphs in $\mathcal{C}$ have minimal blocking set size one and such that Vertex Cover on $\mathcal{C}$ is not solvable in polynomial time, unless* NP = RP.

Proof. It is known that the Unique-SAT problem cannot be solved in polynomial-time unless NP = RP [19, Corollary 1.2]. An input to Unique-SAT is a CNF-formula $\mathcal{F}$ that has either exactly one satisfying solution or is unsatisfiable. The problem is to decide whether $\mathcal{F}$ is satisfiable. It can be shown that the same result holds for Unique-3-SAT [15, Example 26.7], where the input formula is further restricted to be in 3-CNF. We show that the following polynomial-time reduction from Unique-3-SAT to Vertex Cover exists.

▷ Claim 9 (★). There is a polynomial-time reduction from Unique-3-SAT to Vertex Cover, that given a formula $\mathcal{F}$, outputs an instance (G, k) for Vertex Cover such that:
- If $\mathcal{F}$ has exactly one satisfying assignment, then G has a unique minimum vertex cover of size k.
- If $\mathcal{F}$ is unsatisfiable, then G has a unique minimum vertex cover of size $k + 1$.

To conclude the proof, let $\mathcal{C}$ be the graph class consisting of all graphs that are obtained via the reduction given in the claim above, when starting from a formula $\mathcal{F}$ that has zero or one satisfying assignments. As such, solving Vertex Cover on $\mathcal{C}$ in polynomial time corresponds to solving Unique-3-SAT in polynomial time, implying NP = RP. Since any graph in $\mathcal{C}$ has exactly one minimum vertex cover, we obtain that indeed $b_{\mathcal{C}} = 1$, as any vertex that is not part of the minimum vertex cover forms a (minimal) blocking set. ◀

Graphs in the graph class $\mathcal{C}$ constructed in the proof of Theorem 2 are always connected, since they are the complement of a disconnected graph. As such, $\mathcal{C}$ is closed under removing connected components. However, $\mathcal{C}$ is not robust because it is not closed under disjoint union. We can however define $\mathcal{C}'$ to contain all graphs for which all connected components lie in $\mathcal{C}$. Observe that $\mathcal{C}'$ is robust, but that $b_{\mathcal{C}'} = 1$ and Vertex Cover is not solvable in polynomial time on $\mathcal{C}' \supseteq \mathcal{C}$ unless RP = NP.

3.3 Reducing the number of components outside the modulator

As mentioned in the previous subsections, bounded blocking set size is necessary to obtain polynomial kernels for Vertex Cover. Many papers that give polynomial kernels for Vertex Cover parameterized by the size of a $\mathcal{C}$-modulator showed that their graph class $\mathcal{C}$ has bounded blocking set size, see for example [2, 8, 10, 13, 18]. Some of them used the blocking set size of class $\mathcal{C}$ to bound the number of connected components. More precisely, given an instance (G, k, X) of Vertex Cover with $G - X \in \mathcal{C}$ they showed that one can reduce the number of connected components of $G - X$ to $\mathcal{O}(|X|^{b_{\mathcal{C}}+1})$. We will show that one can reduce the number of connected components of $G - X$ to $|X|^{b_{\mathcal{C}}}$, as a first step towards proving Theorem 3. Here we assume that the class $\mathcal{C}$ is robust in order to guarantee that deletion of connected components of $G - X$ again results in a graph of $\mathcal{C}$. At the end of this section, we discuss suitable conditions so that this component reduction can be done efficiently.

Let (G, k, X) be an instance of Vertex Cover parameterized by the size of a $\mathcal{C}$-modulator. First, we define the set $\mathcal{X} = \{Z \subseteq X \mid Z \text{ is an independent set in } G \text{ and } 1 \leq |Z| \leq b_{\mathcal{C}}\}$ as the collection of *chunks* of X. The intuition of defining the set $\mathcal{X}$ of chunks is to find sets in the modulator X for which at least one vertex must be contained in any optimum vertex cover of G. The concept of chunks was first introduced by Jansen and Bodlaender [13].

To reduce the number of connected components of $G - X$, we use the following result due to Hopcroft and Karp [11] which computes a certain crown-like structure in a bipartite graph. The second part of the theorem is not standard (but well known).

▶ **Theorem 10** ([11]). *Let G be an undirected bipartite graph with bipartition V_1 and V_2, on n vertices and m edges. Then we can find a maximum matching of G in time $\mathcal{O}(m\sqrt{n})$. Furthermore, in time $\mathcal{O}(m\sqrt{n})$ we can find either a maximum matching that saturates V_1 or a set $Z \subseteq V_1$ such that $|N_G(Z)| < |Z|$ and such that there exists a maximum matching M of $G - N_G[Z]$ that saturates $V_1 \setminus Z$.*

We construct a bipartite graph G_B to which we will apply Theorem 10 to find a set of connected components in $G - X$ that can be safely removed from G. We denote the set of connected components in $G - X$ by $\mathcal{F}$. The two parts of the bipartite graph G_B are the set $\mathcal{X}$ of chunks and the set $\mathcal{F}$ of connected components in $G - X$. More precisely, for every chunk $Z \in \mathcal{X}$ and for every connected component $H \in \mathcal{F}$ we add a vertex to the bipartite graph. To simplify notation we denote the vertex of G_B that corresponds to a connected component $H \in \mathcal{F}$ resp. a chunk $Z \in \mathcal{X}$ by H resp. Z. We add an edge between a vertex $H \in \mathcal{F}$ and a vertex $Z \in \mathcal{X}$ when $N_G(Z) \cap V(H)$ is a blocking set in H, i.e., when $\mathrm{OPT}(H - N_G(Z)) + |N_G(Z) \cap V(H)| > \mathrm{OPT}(H)$. Observe that G_B can only be constructed in polynomial time, if this condition can be tested in polynomial time, which is possible under some additional assumptions on $\mathcal{C}$, as we will see later.

It follows from Theorem 10 that there exists either a maximum matching M of G_B that saturates $\mathcal{X}$ or a set $\mathcal{X}' \subseteq \mathcal{X}$ such that $|N_{G_B}(\mathcal{X}')| < |\mathcal{X}'|$ and such that there exists a maximum matching M of $G_B - N_{G_B}[\mathcal{X}']$ that saturates $\hat{\mathcal{X}} = \mathcal{X} \setminus \mathcal{X}'$. If there exists a maximum matching M of G_B that saturates $\mathcal{X}$ then let $\mathcal{X}' = \emptyset$ and let $\hat{\mathcal{X}} = \mathcal{X}$. Let $\mathcal{F}_D = \mathcal{F} \setminus (N_{G_B}(\mathcal{X}') \cup V(M))$ be the set of connected components in $\mathcal{F}$ that are neither in the neighborhood of $\mathcal{X}'$ nor endpoint of a matching edge of M.

▶ **Reduction Rule 1.** Delete all connected components in $\mathcal{F}_D$ from G and decrease the size of k by $\mathrm{OPT}(\mathcal{F}_D)$ the size of an optimum vertex cover in $\mathcal{F}_D$.

Observe that Reduction Rule 1 deletes also all connected components $H \in \mathcal{F}$ which have the property that for all sets $Z \in \mathcal{X}$ it holds that $N(Z) \cap V(H)$ is not a blocking set of H because these connected components correspond to isolated vertices in the bipartite graph G_B. To show the correctness of Reduction Rule 1 we will use the following lemma which guarantees us the existence of certain optimum vertex covers of G.

▶ **Lemma 11 (★).** *There exists an optimum vertex cover S of G with $S \cap Z \neq \emptyset$ for all $Z \in \hat{\mathcal{X}}$.*

Now, we show the correctness of Reduction Rule 1 using Lemma 11. Let $(\widetilde{G}, \widetilde{k}, X)$ be the reduced instance, i.e., $\widetilde{G} = G - \mathcal{F}_D$ and $\widetilde{k} = k - \mathrm{OPT}(\mathcal{F}_D)$. Obviously, if (G, k, X) is a yes-instance then $(\widetilde{G}, \widetilde{k}, X)$ is a yes-instance. For the other direction, assume that $(\widetilde{G}, \widetilde{k}, X)$ is a yes-instance. Observe that M is also a matching in $\widetilde{G}_B$ that saturates $\hat{\mathcal{X}}$ because we delete no connected component that is an endpoint of a matching edge. Furthermore, it holds that either $\hat{\mathcal{X}} = \mathcal{X}$ or $|N_{\widetilde{G}_B}(\mathcal{X}')| < |\mathcal{X}'|$ because we delete no connected component that

corresponds to a vertex in $|N_{\widetilde{G}_B}(\mathcal{X}')|$. Thus, it follows from Lemma 11 that there exists an optimum vertex cover $\widetilde{S}$ of $\widetilde{G}$ with $\widetilde{S} \cap Z \neq \emptyset$ for all sets $Z \in \hat{\mathcal{X}}$. Note that every connected component $H \in \mathcal{F}_D$ is only adjacent to vertices in $\hat{\mathcal{X}}$ in G_B. Since every set $Z \in \hat{\mathcal{X}}$ has a non-empty intersection with the set $\widetilde{S}$, it holds that there exists an optimum vertex cover S_H of H which contains the set $N_G(X \setminus \widetilde{S}) \cap V(H)$. Let S be the set that results from adding for each connected component $H \in \mathcal{F}_D$ the optimum vertex cover S_H to the set $\widetilde{S}$. By construction, it holds that S is a vertex cover of G of size $|\hat{S}|+\text{OPT}(\mathcal{F}_D) \leq \hat{k}+\text{OPT}(\mathcal{F}_D) = k$. This proves that (G, k, X) is also a yes-instance. Overall, we showed that Reduction Rule 1 is safe.

▶ **Theorem 12 (★).** *Let (G, k, X) be an instance of Vertex Cover parameterized by the size of a $\mathcal{C}$-modulator that is reduced under Reduction Rule 1. The graph $G - X$ has at most $|X|^{b_{\mathcal{C}}}$ connected components.*

To use Theorem 12 to prove that we can efficiently reduce the number of connected components in $G - X$ when X is a $\mathcal{C}$-modulator, we need to show that, under certain assumptions, Reduction Rule 1 can be applied in polynomial time. We start by providing two sufficient conditions in the next lemma.

▶ **Lemma 13 (★).** *If $b_{\mathcal{C}}$ is bounded, if Vertex Cover is solvable in polynomial time on graphs of class $\mathcal{C}$ and if we can verify in polynomial time whether a given set Y is a blocking set in a graph of class $\mathcal{C}$ then we can apply Reduction Rule 1 in polynomial time.*

We continue by providing two cases that satisfy the preconditions for the lemma above, such that Reduction Rule 1 can be applied in polynomial time on these graph classes.

First of all, we consider the case that graph class $\mathcal{C}$ is hereditary. In this case, being solvable in polynomial time on the class $\mathcal{C}$ is sufficient to also be able to verify whether a given subset of the vertices is a blocking set, thus allowing us to apply Reduction Rule 1 in polynomial time. As mentioned in Subsection 3.3 we also need that $b_{\mathcal{C}}$ is bounded. Overall, we assume that $\mathcal{C}$ is a hereditary graph class on which Vertex Cover is polynomial-time solvable and where $b_{\mathcal{C}}$ is bounded.

▶ **Lemma 14 (★).** *Let $\mathcal{C}$ be any hereditary graph class on which Vertex Cover can be solved in polynomial time and where $b_{\mathcal{C}}$ is bounded. Then Reduction Rule 1 can be applied in polynomial time.*

Theorem 3 (restated below) now follows directly from Theorem 12 and Lemmas 13 and 14.

▶ **Theorem 3.** *Let $\mathcal{C}$ be any hereditary graph class with minimal blocking set size d on which Vertex Cover can be solved in polynomial time. There is an efficient algorithm that given (G, k, X) such that $G - X \in \mathcal{C}$ returns an equivalent instance (G', k', X) such that $G' - X \in \mathcal{C}$ has at most $\mathcal{O}(|X|^d)$ connected components.*

We can actually further generalize Theorem 3 to some non-hereditary graph classes. However, we have more problems to show that Lemma 13 holds for non-hereditary graph classes, because after deleting vertices from a graph G that is contained in a non-hereditary graph class $\mathcal{C}$ we do not know whether the resulting graph still belongs to the graph class $\mathcal{C}$. As such we need the additional assumption that Vertex Cover is also polynomial-time solvable on graph class $\mathcal{C} + 1$. This additional assumption is not unreasonable, when our goal is to obtain a kernelization algorithm for Vertex Cover. In fact, in order to obtain any kernel for Vertex Cover parameterized by the size of a modulator to $\mathcal{C}$ it is necessary to assume that the problem is FPT. From this, it immediately follows that we can solve Vertex Cover in polynomial time on $\mathcal{C} + 1$.

▶ **Theorem 15 (★).** *Let $\mathcal{C}$ be any robust graph class with minimal blocking set size d on which* Vertex Cover *can be solved in polynomial time. Furthermore, assume that* Vertex Cover *can be solved in polynomial time on graphs of graph class $\mathcal{C}+1$. There is an efficient algorithm that given (G,k,X) such that $G-X\in\mathcal{C}$ returns an equivalent instance (G',k',X) such that $G'-X\in\mathcal{C}$ has at most $\mathcal{O}(|X|^d)$ connected components.*

4 Minimal blocking sets in graphs of bounded elimination distance

As seen in the previous section, minimal blocking sets play an important role for Vertex Cover kernelization. In this section we try to combine different structural parameters by considering the minimal blocking set size of graphs that have elimination distance d to some graph class $\mathcal{C}$ that has bounded minimal blocking set size. We prove the following theorem.

▶ **Theorem 4 (★).** *Let $\mathcal{C}$ be a robust hereditary graph class where $b_{\mathcal{C}}$ is bounded. For every integer $d\geq 1$ it holds that*

$$\beta_{\mathcal{C}}(d)=\begin{cases}2^{d-1}+1 & \text{, if } b_{\mathcal{C}}=1,\\ (b_{\mathcal{C}}-1)2^d+1 & \text{, if } b_{\mathcal{C}}\geq 2.\end{cases}$$

Proving the theorem consists of proving both the upper and the lower bound. The upper bound for $\beta_{\mathcal{C}}(d)$ given above only holds when $\mathcal{C}$ is hereditary. When $\mathcal{C}$ is not hereditary, we obtain an upper bound when $\mathcal{C}$ satisfies the following additional property:

▶ **Definition 16.** *We say that a graph class $\mathcal{C}$ is* f-robust*, if $b_{\mathcal{C}+c}\leq f(c)=f(b_{\mathcal{C}},c)$ for a computable function f.*

▶ **Theorem 17 (★).** *Let $\mathcal{C}$ be an f-robust and robust graph class where $b_{\mathcal{C}}$ is bounded, and let $d\geq 0$. It holds that*

$$\beta_{\mathcal{C}+c}(d)\leq\left(\sum_{i=0}^{d}\binom{d}{i}f(c+i)\right)-2^d+1.$$

The lower bound given by Theorem 4 however extends to any robust graph class $\mathcal{C}$. The lower bound is proven by constructing for each graph class $\mathcal{C}$ where $b_{\mathcal{C}}$ is bounded and each integer $d\geq 1$ a graph G with $\mathrm{ed}_{\mathcal{C}}(G)=d$ that contains a minimal blocking set of size at least $2^{d-1}+1$ when $b_{\mathcal{C}}=1$, and of size at least $(b_{\mathcal{C}}-1)2^d+1$ when $b_{\mathcal{C}}\geq 2$. Since $\beta_{\mathcal{C}}(d)=\max\{\beta(G)\mid \mathrm{ed}_{\mathcal{C}}(G)\leq d\}$ this gives the desired result. This result is obtained by showing that, given a graph H, we can obtain a graph H' such that $\mathrm{ed}_{\mathcal{C}}(H')\leq\mathrm{ed}_{\mathcal{C}}(H)+1$ and $\beta(H')\geq 2\beta(H)-1$. For $b_{\mathcal{C}}=1$ we need an additional construction that, given a graph H, allows us to obtain a graph H' with $\mathrm{ed}_{\mathcal{C}}(H')\leq\mathrm{ed}_{\mathcal{C}}(H)+1$ and $\beta(H')\geq\beta(H)+1$. A depiction of both constructions is shown in Figure 2.[4]

5 Kernelization results

In this section, we will combine the results from Sections 3.3 and 4 to obtain polynomial kernelizations for Vertex Cover parameterized by a $\mathcal{C}$-modulator or a $(\mathcal{C},d)$-modulator. In Section 3.3 we have seen necessary assumptions on a graph class $\mathcal{C}$ such that Reduction Rule 1 can be applied efficiently. We can show that for hereditary graph classes, for which Vertex Cover is solvable in polynomial time, Vertex Cover can also be solved efficiently on graphs for which $\mathrm{ed}_{\mathcal{C}}(G)$ is bounded. From that, we then obtain the following.

[4] For a proof of the correctness of this construction, refer to Section 4.1 in the full version of the paper.

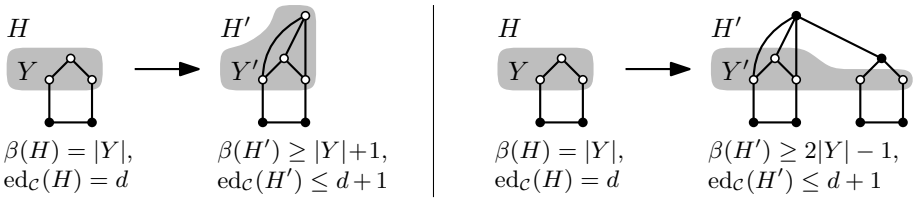


Figure 2 This figure depicts the construction used to prove the lower bound on $\beta_{\mathcal{C}}(d)$ in Theorem 4. White vertices form minimal blocking sets.

▶ **Corollary 18 (★).** *Reduction Rule 1 is applicable in polynomial time on graphs G with a given $(\mathcal{C}, d)$-modulator X, where $\mathcal{C}$ is a hereditary graph class on which* Vertex Cover *is solvable in polynomial time, and where $b_{\mathcal{C}}$ is bounded.*

For non-hereditary graph classes $\mathcal{C}$ we also need that Vertex Cover is solvable in polynomial time on graph class $\mathcal{C} + c$ with c constant. For the next corollary, observe that in particular $\beta_{\mathcal{C}}(d)$ is bounded if $\mathcal{C}$ is known to be either hereditary or f-robust, by Theorems 4 and 17.

▶ **Corollary 19 (★).** *Reduction Rule 1 is applicable in polynomial time on graphs G with a given $(\mathcal{C}, d)$-modulator X, where $\mathcal{C}$ is a robust graph class with the properties that $\beta_{\mathcal{C}}(d)$ is bounded for any constant d and that* Vertex Cover *is polynomial-time solvable on graph class $\mathcal{C} + c$ for constant c.*

5.1 General results

In this section, we show that Vertex Cover parameterized by the size of a $(\mathcal{C}, d)$-modulator has a polynomial kernel when the graph class $\mathcal{C}$ fulfills some additional properties. The assumptions that $\beta_{\mathcal{C}}(d)$ is bounded and that Vertex Cover is polynomial-time solvable on the considered graph class are necessary, if these assumptions fail a polynomial kernel is unlikely to exist. The same holds for the assumption that Vertex Cover parameterized by a $\mathcal{C}$-modulator has a polynomial kernel. We additionally require that $\mathcal{C}$ is a robust graph class that is either hereditary, or has the property that Vertex Cover is polynomial-time solvable on $\mathcal{C} + c$. These assumptions will ensure that our reduction rule can be applied in polynomial time.

▶ **Lemma 20 (★).** *Let $\mathcal{C}$ be a robust graph class for which $\beta_{\mathcal{C}}(d)$ is bounded and on which* Vertex Cover *is polynomial-time solvable, such that $\mathcal{C}$ is hereditary or* Vertex Cover *is polynomial-time solvable on $\mathcal{C} + c$ for all constants c.*

Suppose Vertex Cover *parameterized by the size of a $\mathcal{C}$-modulator $\hat{X}$ has a (randomized) polynomial kernel with $g(|\hat{X}|)$ vertices. Then* Vertex Cover *parameterized by the size of a $(\mathcal{C}, d)$-modulator X has a (randomized) polynomial kernel with $\mathcal{O}(g(|X|^b))$ vertices, where $b = \prod_{i=1}^{d} \beta_{\mathcal{C}}(i)$.*

The kernel is obtained by induction, by transforming an instance (G, k, X) of Vertex Cover parameterized by $(\mathcal{C}, d)$-modulator to an instance parameterized by $(\mathcal{C}, d-1)$-modulator. This is done by first reducing the number of connected components in $G - X$ using Reduction Rule 1, and then adding the root of the treedepth decomposition of each connected component of $G - X$ to the modulator. This method for kernelization is similar to the kernelization for Vertex Cover parameterized by the size of a d-treedepth modulator (see [2]). One difference is that we do not introduce hyper-edges.

Observe that in the above lemma statement, when Vertex Cover parameterized by a $\mathcal{C}$ modulator allows a polynomial kernel, the fact that Vertex Cover is solvable in polynomial time on graphs from $\mathcal{C} + c$ is immediate: since the problem has a polynomial kernel, it must be FPT in the parameter. Since in this case the size of a $\mathcal{C}$-modulator is c, which is constant, the result follows.

In the above lemma statement, we assume that $\beta_{\mathcal{C}}(d)$ is bounded to obtain the kernelization. We observe that for hereditary graph classes, this assumption is not needed, it follows from our results in Theorem 4 that it suffices to bound $b_{\mathcal{C}}$. Furthermore, a bound on $b_{\mathcal{C}}$ often comes naturally: if Vertex Cover parameterized by a $\mathcal{C}$-modulator has a polynomial kernel, it follows from Theorem 1 that, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$, there must exist a constant d such that $b_{\mathcal{C}} \leq d$.

▶ **Theorem 5 (★).** *Let $\mathcal{C}$ be a hereditary and robust graph class for which $b_{\mathcal{C}}$ is bounded, such that Vertex Cover has a (randomized) polynomial kernelization parameterized by the size of a modulator to $\mathcal{C}$. Then Vertex Cover also has a (randomized) polynomial kernelization parameterized by the size of a modulator to graphs of bounded elimination distance to $\mathcal{C}$.*

Similarly, for non-hereditary graph classes, it suffices if $\mathcal{C}$ is f-robust to obtain a polynomial kernel. The size of the kernel depends on f.

▶ **Corollary 21 (★).** *Let $\mathcal{C}$ be a robust and f-robust graph class for which $b_{\mathcal{C}}$ is bounded and for which Vertex Cover parameterized by the size of a $\mathcal{C}$ modulator $\hat{X}$ has a (randomized) polynomial kernel. Then Vertex Cover parameterized by the size of a $(\mathcal{C}, d)$-modulator has a (randomized) polynomial kernel.*

5.2 Kernel for modulator to bounded $\mathcal{C}_{\mathrm{LP}}$ elimination distance

In this section, we show how Theorem 6 follows from the general results in the previous section, to have an explicit example for a non-hereditary base class $\mathcal{C}$. That is, we show how to get a randomized polynomial kernel for Vertex Cover parameterized by the size of a modulator X such that $G - X$ has bounded elimination distance to the non-hereditary class $\mathcal{C}_{\mathrm{LP}}$ of graphs where integral and fractional vertex cover size coincide. Towards proving this result, we show the following relation between the value of $\ell = \mathrm{OPT}(G) - \mathrm{LP}(G)$ and the size of a $\mathcal{C}_{\mathrm{LP}}$-modulator in G.

▶ **Lemma 22 (★).** *Let G be a graph, and let $\ell = \mathrm{OPT}(G) - \mathrm{LP}(G)$. There exists a vertex set $X \subseteq V(G)$ of size at most 2ℓ such that $\mathrm{OPT}(G - X) = \mathrm{LP}(G - X)$.*

We can show that $\mathcal{C}_{\mathrm{LP}}$ is f-robust[5] with $f(c) = f(b_{\mathcal{C}}, c) = 2c + b_{\mathcal{C}_{\mathrm{LP}}} = 2c + 2$. Using Theorem 17 and Lemma 20 we can now generalize the kernelization for Vertex Cover parameterized by the size of a d-treedepth modulator and parameterized by the difference between an optimum vertex cover and an optimum LP solution using the size of a $(\mathcal{C}_{\mathrm{LP}}, d)$-modulator as the parameter. The following theorem subsumes Theorem 6.

▶ **Theorem 23 (★).** *An optimum $(\mathcal{C}_{\mathrm{LP}}, d)$-modulator of a graph G has at most the size of a d-treedepth modulator of G and at most twice the size of $\mathrm{OPT}(G) - \mathrm{LP}(G)$. Furthermore, Vertex Cover parameterized by the size of a $(\mathcal{C}_{\mathrm{LP}}, d)$-modulator admits a randomized polynomial kernel.*

[5] This is proven in the full version at the end of Section 4.2.

6 Conclusion

In the first part (Section 3) we have showed that bounded minimal blocking set size in $\mathcal{C}$ is necessary but not sufficient to get a polynomial kernel for Vertex Cover when parameterized by the size of a modulator X to a robust (or at least union-closed) class $\mathcal{C}$. We then showed that bounded minimal blocking set size suffices to efficiently reduce the number of components of $G - X$ assuming that $\mathcal{C}$ is robust (so deletion of components lets $G - X$ stay in $\mathcal{C}$) and that we can efficiently compute optimum vertex covers and test blocking sets in graphs of $\mathcal{C}$. The obtained bound of $\mathcal{O}(|X|^{bc})$ components is likely optimal because it matches the size lower bound proved earlier, which requires only components of constant size.

In the second part we first proved bounds for the minimal blocking set size relative to elimination distances to classes $\mathcal{C}$, motivated by the bounds that Bougeret and Sau [2] obtained relative to treedepth (Section 4). We obtain the exact value for all hereditary classes $\mathcal{C}$ and slightly weaker upper bounds for certain non-hereditary classes $\mathcal{C}$. This enabled new polynomial kernelization results for Vertex Cover that effectively replace (the size of) a modulator to a class $\mathcal{C}$ to modulators to graphs of bounded elimination distance to $\mathcal{C}$, e.g., when $\mathcal{C}$ is the class of forests, bipartite graphs, or C_{LP} (where integral and fraction vertex cover size coincide).

As future work it would be great to get a similar kernelization result when parameterized by the size of a modulator to bounded elimination distance to the graph class $\mathcal{C}_{2\text{LP}-\text{MM}}$ where $\text{OPT} = 2\text{LP} - \text{MM}$ (i.e., minimum vertex cover size equals two times fractional cost minus size of a maximum matching, cf. [9]), which relates to the randomized kernelization for the corresponding above guarantee parameterization [16]. This would essentially subsume and generalize all currently known polynomial kernelizations for Vertex Cover (to which we came close with the result for bounded elimination distance to $\mathcal{C}_{\text{LP}}$). It would also be nice to have tight bounds for the maximum size of minimal blocking sets in the non-hereditary case, and to get such bounds with fewest possible technical assumptions.

References

1 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discrete Math.*, 28(1):277–305, 2014. doi:10.1137/120880240.

2 Marin Bougeret and Ignasi Sau. How much does a treedepth modulator help to obtain polynomial kernels beyond sparse graphs? In Daniel Lokshtanov and Naomi Nishimura, editors, *12th International Symposium on Parameterized and Exact Computation, IPEC 2017, September 6-8, 2017, Vienna, Austria*, volume 89 of *LIPIcs*, pages 10:1–10:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.IPEC.2017.10.

3 Jannis Bulian and Anuj Dawar. Graph isomorphism parameterized by elimination distance to bounded degree. *Algorithmica*, 75(2):363–382, 2016. doi:10.1007/s00453-015-0045-3.

4 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

5 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4):23:1–23:27, 2014. doi:10.1145/2629620.

6 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

7 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, December 2018. doi:10.1017/9781107415157.

8 Fedor V. Fomin and Torstein J. F. Strømme. Vertex cover structural parameterization revisited. In Pinar Heggernes, editor, *Graph-Theoretic Concepts in Computer Science - 42nd International Workshop, WG 2016, Istanbul, Turkey, June 22-24, 2016, Revised Selected Papers*, volume 9941 of *Lecture Notes in Computer Science*, pages 171–182, 2016. doi:10.1007/978-3-662-53536-3_15.

9 Shivam Garg and Geevarghese Philip. Raising the bar for vertex cover: Fixed-parameter tractability above A higher guarantee. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1152–1166. SIAM, 2016. doi:10.1137/1.9781611974331.ch80.

10 Eva-Maria C. Hols and Stefan Kratsch. Smaller parameters for vertex cover kernelization. In Daniel Lokshtanov and Naomi Nishimura, editors, *12th International Symposium on Parameterized and Exact Computation, IPEC 2017, September 6-8, 2017, Vienna, Austria*, volume 89 of *LIPIcs*, pages 20:1–20:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.IPEC.2017.20.

11 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973. doi:10.1137/0202019.

12 Bart M. P. Jansen. *The power of data reduction: kernels for fundamental graph problems*. PhD thesis, Utrecht University, 2013.

13 Bart M. P. Jansen and Hans L. Bodlaender. Vertex cover kernelization revisited - upper and lower bounds for a refined parameter. *Theory Comput. Syst.*, 53(2):263–299, 2013. doi:10.1007/s00224-012-9393-4.

14 Bart M. P. Jansen and Astrid Pieterse. Polynomial kernels for hitting forbidden minors under structural parameterizations. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPIcs*, pages 48:1–48:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.48.

15 Dexter Campbell Kozen. *Design and Analysis of Algorithms*. Texts and Monographs in Computer Science. Springer, 1992. doi:10.1007/978-1-4612-4400-4.

16 Stefan Kratsch. A randomized polynomial kernelization for vertex cover with a smaller parameter. *SIAM J. Discrete Math.*, 32(3):1806–1839, 2018. doi:10.1137/16M1104585.

17 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 450–459. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.46.

18 Diptapriyo Majumdar, Venkatesh Raman, and Saket Saurabh. Polynomial kernels for vertex cover parameterized by small degree modulators. *Theory Comput. Syst.*, 62(8):1910–1951, 2018. doi:10.1007/s00224-018-9858-1.

19 Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(3):85–93, 1986. doi:10.1016/0304-3975(86)90135-0.

Near-Optimal Complexity Bounds for Fragments of the Skolem Problem

S. Akshay
IIT Bombay, India
akshayss@cse.iitb.ac.in

Nikhil Balaji
University of Oxford, UK
nikhil.balaji@cs.ox.ac.uk

Aniket Murhekar
University of Illinois, Urbana Champaign, Urbana, IL, USA
aniket1602@gmail.com

Rohith Varma
Indian Institute of Technology Palakkad, India
rvarma.kvm@gmail.com

Nikhil Vyas
MIT, Cambridge, MA, USA
nikhilv@mit.edu

Abstract

Given a linear recurrence sequence (LRS), specified using the initial conditions and the recurrence relation, the Skolem problem asks if zero ever occurs in the infinite sequence generated by the LRS. Despite active research over last few decades, its decidability is known only for a few restricted subclasses, by either restricting the *order* of the LRS (upto 4) or by restricting the *structure* of the LRS (e.g., roots of its characteristic polynomial).

In this paper, we identify a subclass of LRS of arbitrary order for which the Skolem problem is *easy*, namely LRS all of whose characteristic roots are (possibly complex) roots of real algebraic numbers, i.e., roots satisfying $x^d = r$ for r real algebraic. We show that for this subclass, the Skolem problem can be solved in $\mathsf{NP}^{\mathsf{RP}}$. As a byproduct, we implicitly obtain *effective* bounds on the zero set of the LRS for this subclass. While prior works in this area often exploit deep results from algebraic and transcendental number theory to get such effective results, our techniques are primarily algorithmic and use linear algebra and Galois theory. We also complement our upper bounds with a NP lower bound for the Skolem problem via a new direct reduction from 3-CNF-SAT, matching the best known lower bounds.

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

Keywords and phrases Linear Recurrences, Skolem problem, NP-completeness, Weighted automata

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.37

Funding This work was partly supported by DST/CEFIPRA/INRIA Associated team EQuaVE.
S. Akshay: Partly supported by DST-INSPIRE Faculty Award [IFA12-MA-17] and SERB Matrices grant MTR/2018/000744.
Nikhil Vyas: Supported by NSF CCF-1909429.

Acknowledgements This research was supported in part by the International Centre for Theoretical Sciences (ICTS) during a visit for participating in the program - Workshop on Algebraic Complexity Theory (Code: ICTS/wact2019/03).

37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020).
Editors: Christophe Paul and Markus Bläser; Article No. 37; pp. 37:1–37:18

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

1 Introduction

A (rational) linear recurrence sequence (LRS) is an infinite sequence of rationals $u_1 \ldots$ such that the n-th term can be written as a linear combination of the previous terms, $u_n = a_1 u_{n-1} + \ldots + a_k u_{n-k}$, where each coefficient a_i is a rational. The number k is called the *order* of the LRS. Once we fix the initial k values, the equation above uniquely determines the infinite sequence. LRS are a fundamental object of study in discrete mathematics, with a rich theory and widespread applications and have been widely investigated. However, some very basic computational questions remain unsolved for the last several decades despite considerable interest. The most well-known of these is the so-called Skolem problem (or Skolem-Pisot problem): given an LRS $\mathbf{u} = \{u_n\}_{n=0}^{\infty}$ with coefficients $\{a_i\}_{i=1}^{k}$ and initial conditions $\{u_i\}_{i=1}^{k}$, does there exist an n such that $u_n = 0$. This problem is known to be NP-hard [9] (see also [3]), but even decidability is open. A fundamental result in this area is the Skolem-Mahler-Lech theorem, which states that the *zero set* of an LRS is a *semi-linear set*[18], i.e, the zero set is the union of a finite set and a finite union of arithmetic progressions. Unfortunately, this nice characterization does not result in an algorithm due its use of *non-effective* techniques [28], and it does not help in deciding if the zero set is non-empty. To obtain decidability of the Skolem problem, researchers have considered restricted classes of LRS, along two broad avenues.

The first is by *restricting the order* of the LRS. Vereshchagin [30] gave an algorithm to decide Skolem problem up to order 4; the computational complexity of this algorithm was analyzed by Chonev, Ouaknine and Worrell (Appendix of [12]) to show that it is in the complexity)[1] class $\mathsf{NP}^{\mathsf{RP}}$, which is contained in the second level of Polynomial Hierarchy (PH). But, no lower bound is known and hence we do not know if these results are tight, even upto the RP-oracle. Indeed, the NP-hardness reductions in [9, 3] do not work when the order is restricted. The second approach to obtaining decidability has been to *restrict the spectral structure* of the LRS. Given an LRS $\mathbf{u}$ of order k, the roots of its characteristic polynomial $x^k - a_1 x^{k-1} \ldots - a_k = 0$, also called characteristic roots, can be used to give a closed form expression for the LRS (see Proposition 1). Restricting the spectral structure of the LRS refers to imposing conditions on these roots, i.e., considering classes of LRS where the characteristic roots have special properties. In [3], it was shown that for LRS whose characteristic roots are complex roots of unity, the Skolem problem is NP-complete. To the best of our knowledge, no efficient bounds (e.g., within the Polynomial Hierarchy) or optimality results are known for the Skolem problem for any other natural non-trivial subclasses (e.g., for simple LRS, where the roots are distinct), even if decidability is known or considered folklore [31, 4, 7, 17].

In this paper, we take a step in this direction, and provide optimal complexity bounds on the Skolem problem for a highly expressive subclass of LRS, obtaining by restricting its spectral structure. More precisely, we consider the class of LRS where all the roots of the characteristic polynomial are roots of real numbers, i.e., λ such[2] that $\lambda^n = r$ for some $n \in \mathbb{N}, r$ real algebraic. We denote this class by LRS(r$\mathbb{R}$) (and by LRS($\mathbb{R}$) the class of LRS with real characteristic roots). Notice that this class considerably extends the subclass in [3], which corresponds to roots of unity, i.e., λ such that $\lambda^n = 1$ for some $n \in \mathbb{N}$. Restricting the spectrum of a polynomial to be reals or roots of reals has been used to recover decidability

[1] RP is the class of problems that admits a randomized polynomial time algorithm with one-sided error
[2] Notice that every complex algebraic number is a root of a quadratic polynomial with real coefficients. However not all complex algebraic numbers are n-th roots of a real number.

across various areas, ranging from hybrid systems [17] to probabilistic verification [4, 1], and weighted automata [7]. Our results allow us to infer strong complexity results on these models, and solves an open problem stated in [7], as discussed later in the paper.

Our contributions: Our main result is that the Skolem problem for LRS(r$\mathbb{R}$) can be solved in $\mathsf{NP}^{\mathsf{RP}}$. Since this class contains LRS over roots of unity, it inherits the NP hardness for Skolem. Now standard [20] derandomization assumptions in computational complexity imply that $\mathsf{P} = \mathsf{RP}$, under which condition, we obtain that our result on Skolem problem for LRS(r$\mathbb{R}$) is tight. To the best of our knowledge, this is the first tight, upto derandomization, complexity bound on the Skolem problem for a class of LRS whose roots may contain arbitrary reals. We also remark that our results combined with the order 4 results (where also an $\mathsf{NP}^{\mathsf{RP}}$ upper bound is obtained in [12]), seem to suggest that when Skolem problem is decidable, it is easy, i.e., in $\mathsf{NP}^{\mathsf{RP}}$.

To understand the difficulty in showing our result, note that while it is a folklore result that the Skolem problem for LRS($\mathbb{R}$) is decidable, standard ideas don't seem to yield even a PSPACE upper bound. The usual way to analyze an LRS is via the *exponential polynomial solution* (Proposition 1), where the coefficients of the exponential polynomials can be shown to be algebraic numbers from a (exponentially) large field extension of $\mathbb{Q}$. A linear combination of these algebraic numbers could be double exponentially small (See for example the work of Tiwari on the sign problem [29] and Allender et al. [5]) and we currently don't have techniques to handle numerical computation in this regime efficiently. Thus, while one does get decidability, none of these approaches seem to immediately provide precise complexity bounds beyond NEXP to the best of our knowledge. Another example comes from a recent work of Fijalkow et al. [17], where to decide the reachability problem for Linear Time Invariant systems (which they also prove to be Skolem-hard) specified by a matrix A, a key step in the algorithm relies on the Jordan block decomposition $P^{-1}DP$ of A to analyse the product $P^{-1}D^nP$ for various values of n. Though the authors do not analyse the complexity of their algorithm, since the matrices P and P^{-1} have algebraic numbers of exponential degree, a similar complexity bottleneck seems unavoidable.

Thus, in order to prove our results, we introduce new techniques to circumvent numerical difficulties that are usually encountered in computations involving irrational numbers, which we believe could be of independent interest. Numerical difficulties arise regularly in problems in computational geometry (for example, the Square roots sum problem [29]), numerical analysis [10] and algorithmic game theory (computing Nash equilibria of 3-player games [15]), to mention a few examples. A key step in our proof is to revisit and strengthen the folklore result (Proposition 1) that LRS correspond precisely to the class of exponential polynomials. We give a refined version (see Lemma 6) of the closed form of the LRS. Using this and appealing to the classical root separation theorems [23], we obtain an $\mathsf{NP}^{\mathsf{RP}}$ algorithm for the Skolem problem over reals, i.e., LRS($\mathbb{R}$) (Theorem 7). We then reduce Skolem problem for LRS(r$\mathbb{R}$) to that of LRS($\mathbb{R}$) in two steps. First we reduce the Skolem problem for LRS(r$\mathbb{R}$) to the Skolem problem for *simple* LRS(r$\mathbb{R}$), i.e., where the roots are assumed to be all distinct. Then we show that we can reduce the Skolem problem for *simple* LRS(r$\mathbb{R}$) to exponentially many instances of the Skolem problem for *simple* LRS($\mathbb{R}$). In doing so, the most technical part is to prove that numerical issues do not surface again after the reduction (Lemma 12).

We may also contrast our techniques with those of earlier results on the Skolem problem. While the authors of [12] also obtain a $\mathsf{NP}^{\mathsf{RP}}$ upper bound for the Skolem problem (only up to order 4), they use Baker's theorem on linear forms in logarithms [6]. Though Baker's theorem seems unavoidable to show decidability for special cases where the characteristic

roots have irrational phases (even here, currently the method can help prove decidability of Skolem problem only for LRS up to order 4), it is potentially an obstacle to prove good complexity bounds since the "effective" version of Baker's theorem has constants that could be double exponentially small as a function of the order of the LRS. Since [12] deal only with constant order LRS, the constants from Baker's theorem do not pose a problem. Since LRS(r$\mathbb{R}$) does not have roots with irrational phases, we can avoid using Baker's theorem and instead rely on elementary linear algebra and Galois theory to obtain a strong lower bound on the zeros of the LRS.

As a final contribution, we also analyze the hardness proof for Skolem to see how it behaves with respect to various parameters. Towards this, we first provide a direct reduction from 3-CNF-SAT for showing the NP-hardness proof. Using this we observe that Skolem is strongly NP-hard with respect to the initial conditions while it is only weakly NP-hard with respect to the coefficients as they seem to blow up in the reduction. Note that this strengthens the lower bound from [3], which only shows weak NP-hardness in both initial conditions and coefficients. We must however point out that a careful analysis of [9] suggests that their indirect reduction, through universality of automata, is also weakly NP-hard wrt coefficients and strongly wrt initial conditions. However, all the three reductions are weakly NP-hard wrt the coefficients of the LRS since in all the reductions the numerical value of the coefficients may require polynomially (in the size of the LRS) many bits to represent.

Other related work. A recent work of Min Sha [25] shows effective bounds for *simple* LRS, when there are one or two dominant roots (and other roots are arbitrary), using Baker's theorem on linear forms of logarithms. The class of LRS considered here are orthogonal to the one in the current paper since they cannot handle the case of repeated complex dominant roots, even if the roots are complex roots of rationals (for which we obtain $\mathsf{NP}^{\mathsf{RP}}$) or roots of unity (for which we have containment in NP by [3]). On the other hand, we require all roots to be roots of reals, even if there is a single dominant root.

2 Preliminaries and notations

We first set up some notation that we use throughout the paper. We denote by $\mathsf{poly}(m)$, any quantity that is bounded from above by $m^{O(1)}$ and by $\exp(m)$ any quantity that is bounded from above by $2^{m^{O(1)}}$. By $\mathbb{Q}_{\exp(m)}$ we denote rational numbers where both the numerator and denominator are bounded by integers of magnitude at most $\exp(m)$. Note that such a number can be represented in binary by a string of length at most $\mathsf{poly}(m)$. Throughout the paper, we say that the magnitude of a rational number being $\exp(m)$-bounded and the rational number being representable by $\mathsf{poly}(m)$-bits interchangeably. For an algebraic number λ, we denote by $\mathbb{Q}_{\exp(m)}(\lambda)$ all the elements of the field extension which are obtained by rational linear combination of powers of λ, where the rationals used in the linear combination are $\exp(m)$-bounded. Also for a field $\mathbb{F}$, we use $\overline{\mathbb{F}}$ to denote its algebraic closure.

We introduce some standard definitions and properties of LRS. For a detailed treatment of LRS, see the book of Evereste et al. [16]. An LRS of order k is a sequence whose n^{th} term is given by $u_n = \sum_{i=1}^{k} a_i u_{n-i}$, where $u_1, \ldots, u_k$ and $a_1, \ldots, a_k$ are respectively called the *initial conditions* and *coefficients* of the LRS. We assume all initial conditions and coefficients to be rational and hence all terms are rational. Such LRS are sometimes called rational LRS, but we will call them just LRS for simplicity, and denote by $\mathbf{u}$ a rational LRS and by u_n the n^{th} term of $\mathbf{u}$.

Given an LRS, its *characteristic polynomial* is $\chi_{\mathbf{u}}(x) = x^k - \sum_{i=0}^{k-1} a_{k-i}x^i = \prod_{j=1}^{g} f_j(x)^{h_j}$, where the latter equality is obtained by factorizing into irreducible square-free factors f_j over $\mathbb{Q}$. We can also break as $\chi_{\mathbf{u}}(x) = \prod_{j=1}^{r}(x-\lambda_j)^{e_j}$, where λ_j is a root, called a characteristic root with multiplicity e_j and $e_j = h_{j'}$ if λ_j is a root of $f_{j'}$. A *perfect field* is one where every irreducible polynomial over the field has distinct roots. It is known that all characteristic zero fields are perfect. As a result, when $\mathbf{u}$ is a rational LRS, all characteristic roots coming from an irreducible factor occur with same multiplicty. An LRS is called *simple* if $e_j = 1$ for all j, i.e., the characteristic roots are distinct.

▶ **Definition 1** (Exponential polynomial). *An exponential polynomial over a field $\mathbb{F}$ is a special bivariate polynomial $P(x,y) \in \mathbb{F}(x,y)$ of the form $P(x,e^x)$. Such a polynomial is a finite polynomial combination of exponentials $E(x) = \sum_{i=1}^{k} p_i(x)e^{\delta_i x}$, where $\delta_i \in \mathbb{F}$ and $p_i(x) \in \mathbb{F}(x)$.*

▶ **Proposition 1** (Exponential polynomial solution [16]). *Given an LRS $\mathbf{u}$, a closed form solution of the n-th term of the LRS is given as a solution to an exponential polynomial, i.e., $u_n = \sum_{j=1}^{r} p_j(n)\lambda_j^n$, where for $j \in [r]$, $\lambda_j \in \mathbb{C}$, $p_j(n) = \sum_{\ell=0}^{e_j-1} c_{j\ell}n^\ell$ are polynomials of degree at most (e_j-1) and $c_{j\ell} \in \bar{\mathbb{Q}}(\lambda_1,\ldots,\lambda_n)$. Note that for a simple LRS we get $u_n = \sum_{j=1}^{r} c_{j0}\lambda_j^n$.*

Let $e = \max_j(e_j)$ be the highest multiplicity. With this, we can rewrite this equation as

$$u_n = \sum_{j=1}^{r}\left(\sum_{\ell=0}^{e_j-1} c_{j\ell}n^\ell\right)\lambda_j^n \tag{1}$$

We call the coefficients $c_{j\ell}$ the *defining coefficients* of the LRS u_n. We denote by m the bit-size needed to describe the LRS, namely the order, coefficients and initial conditions of the LRS, i.e., $m = ||u|| = k + \sum_{i=1}^{k}(\log a_i + \log u_i)$. We refer to m as the *size* of the LRS.

▶ **Proposition 2.** *Given LRS $\mathbf{u}$ of size m, $n \leq \mathsf{poly}(m)$, u_n is $\mathsf{poly}(m)$-bit representable.*

Algebraic numbers. We introduce some basic notions about algebraic numbers, found in any standard text (e.g., Cohen [13]).

▶ **Definition 2** (Algebraic number, Height, Degree). *A complex number α is called* algebraic *if there is a univariate polynomial $p_\alpha(x)$ with rational coefficients of minimum degree that vanishes at α. p_α is said to be the* defining polynomial *or the* minimal polynomial *of the algebraic number α. The* degree *and* height *of α are then the degree and the maximum value of the coefficients of p_α. The roots of p_α are called the* Galois conjugates *of α.*

▶ **Lemma 3** (Mignotte's Root Separation bound [23]). *If α_i and α_j are roots of an integer polynomial $p(x)$ of degree d and height H, then $|\alpha_i - \alpha_j| > \frac{\sqrt{6}}{d^{\frac{d+1}{2}}H^{d-1}}$*

▶ **Proposition 3.** *If α is an algebraic number of degree d and height H, then the degree and height of α^t are bounded by d and $exp(d)H^{dt}$ respectively for any $t \in \mathbb{N}$.*

We call an algebraic number α, t^{th} primitive root of unity if $\alpha^t = 1$ and for all $i \in \mathbb{N}$, $i, 0 \leq i < t$, $\alpha^i \neq 1$.

3 LRS to Exponential Polynomials : A finer analysis

In this section, we give a refined analysis of the exponential polynomials obtained from LRS, thus strengthening Proposition 1. To do this, we show two lemmas which are possibly of independent interest. The first is a structural lemma, which shows how one can decompose an LRS into a polynomial combination of *simple* LRS, with only a polynomial blow-up in the resulting size.

▶ **Lemma 4.** *[Splitting lemma] Given an (rational) LRS $\boldsymbol{u}$ of size m and order k, we can write $\boldsymbol{u} = \sum_{\ell=0}^{e-1} n^\ell \mathbf{u}^\ell$, with the following properties:*
1. *$\mathbf{u}^\ell$ is a **simple** LRS of order $k_\ell \geq 1$ such that $\sum_{\ell=0}^{e-1} k_\ell = k$.*
2. *The initial conditions and coefficients of $\mathbf{u}^\ell$ are also $\mathsf{poly}(m)$-bit rationals.*

Proof. From equation 1 we have

$$u_n = \sum_{j=1}^{r} \left(\sum_{\ell=0}^{e_j-1} c_{j\ell} n^\ell \right) \lambda_j^n = \sum_{\ell=0}^{e-1} n^\ell \left(\sum_{j:e_j>\ell} c_{j\ell} \lambda_j^n \right) = \sum_{\ell=0}^{e-1} n^\ell u_n^\ell$$

for $e = \max_j(e_j)$ and $u_n^\ell = \sum_{j:e_j>\ell} c_{j\ell}\lambda_j^n$. Note that $\sum_{j:e_j>\ell} c_{j\ell}\lambda_j^n$ is a simple rational LRS because it is an exponential sum. The only λ_j that occur in the expression for u_n^ℓ are the ones with $e_j > \ell$ or the roots of all $f_{j'}$ such that $h_{j'} > \ell$. Hence the characteristic polynomials of u_n^ℓ is $\chi_{\mathbf{u}^\ell}(x) = \prod_{j:h_j>\ell} f_j(x)$. Since $\chi_{\mathbf{u}}(x) = \prod_{j=1}^{g} f_j(x)^{h_j}$ is a rational univariate polynomial, we have that $\chi_{\mathbf{u}^\ell}$ is a product of all those irreducible factors of $\chi_{\mathbf{u}}(x)$ where $h_j > \ell$. Hence the coefficients of the polynomial $\chi_{\mathbf{u}^\ell}(x)$ are all $\mathsf{poly}(m)$-bit bounded since the coefficients of $\chi_{\mathbf{u}}(x)$ can all be written in $\leq m$ bits. Thus in fact the coefficients of each LRS $\mathbf{u}^\ell$ have size $\mathsf{poly}(m)$ bits.

Further the order of $\mathbf{u}^\ell$ is exactly $k_\ell = \ell(\sum_{j:h_j>\ell} deg(f_j))$. Since $deg(\chi_{\mathbf{u}}(x)) = \sum_{j=1}^{g} h_j deg(f_j)$ we have $\sum_{\ell=0}^{e-1} k_\ell = k$ by double counting. We now set up a system of k linear equations with the initial conditions of the LRSs $\mathbf{u}^\ell$ as variables. Each of the k linear equations expresses one initial condition of $\mathbf{u}$ as a linear combination of the initial conditions of $\mathbf{u}^\ell$. This requires us to in turn express the first k terms of $\mathbf{u}^\ell$ as a linear combination of the k_ℓ initial terms of $\mathbf{u}^\ell$ (variables in our system).

Suppose $u_n^\ell = \sum_{i=1}^{k_\ell} a_i u_{n-i}^\ell$ is the recurrence equation for $\mathbf{u}^\ell$, with $u_1^\ell, \ldots, u_{k_\ell}^\ell$ being the initial conditions, and $a_1, \ldots, a_{k_\ell}$ the coefficients of the recurrence. We first express u_n^ℓ for $k_\ell < n \leq k$ in terms of the initial values as $u_n^\ell = c_1 u_1^\ell + \cdots + c_k u_{k_\ell}^\ell$ by applying the recurrence repeatedly. The constants c_i are $\mathsf{poly}(m)$-bit bounded, since $c_i < (a_1 + a_2 + \cdots + a_{k_\ell})^n$. Thus $c_i < (k_\ell \cdot 2^{\mathsf{poly}(m)})^n \leq (m \cdot 2^m)^k = 2^{\mathsf{poly}(m)}$, since we have seen that each a_i is $\mathsf{poly}(m)$-bit bounded. Hence we have a linear system with k equations and k variables, the initial conditions of each $\mathbf{u}^\ell$. Further all constants in the linear systems are $\mathsf{poly}(m)$-bit bounded, and thus we get that all initial conditions of each $\mathbf{u}^\ell$ are also $\mathsf{poly}(m)$-bit bounded. ◀

Our second step is to show that for a *simple* LRS, the coefficients of the exponential polynomial solution are exponentially bounded (and poly in the bit representation).

▶ **Lemma 5.** *Let $\boldsymbol{u}$ be a simple LRS of order k and size m, whose exponential polynomial solution is given by $u_n = \sum_{i=1}^{k} c_i \lambda_i^n$ and initial conditions are $u_1, \ldots, u_k$. Then the coefficients c_i in the exponential polynomial solution are uniquely determined and have the property $c_i \in \mathbb{Q}_{\exp(m)}(\lambda_i)$. That is, c_i are bounded in magnitude by $\frac{1}{2^{\mathsf{poly}(m)}} \leq c_i \leq 2^{\mathsf{poly}(m)}$ for all $i \in [k]$.*

Proof. We proceed by interpolation. We set up a system of linear equations by substituting $n = 1, 2, \ldots, k$. We get, $Vc = U$, where V is the classical Vandermonde matrix given by $V_{ij} = \lambda_j^i$, for $1 \leq i, j \leq k$ and $c = [c_1 c_2 \ldots c_k]^T$ and $U = [u_1 u_2 \ldots u_k]^T$. Now we have $c = V^{-1}U$, and in fact c_i is given by the inner product $\sum_{j=1}^{k} V_{ij}^{-1} u_j$. Since $U \in \mathbb{Q}^{k\times 1}$, to prove the claim, it suffices to show that $V_{ij}^{-1} \in \mathbb{Q}(\lambda_i)$ for $j \in [k]$. To this end, we start with the following well-known (See for example [21], Exercise 40 in Section 1.2.3, wherein the expression is attributed to De Moivre [14]) formula for the inverse of the Vandermonde matrix given by $V = (b_{i,j})_{1\leq i,j\leq k}$ where

$$b_{ij} = \begin{cases} (-1)^{j-1} \left(\dfrac{\displaystyle\sum_{\substack{1 \leq \ell_1 < \ldots < \ell_{k-j} \leq k \\ \ell_1, \ldots, \ell_{k-j} \neq i}} \lambda_{\ell_1} \cdots \lambda_{\ell_{k-j}}}{\lambda_i \displaystyle\prod_{\substack{1 \leq \ell \leq k \\ \ell \neq i}} (\lambda_\ell - \lambda_i)} \right) & : 1 \leq j < k \\ \dfrac{1}{\lambda_i \displaystyle\prod_{\substack{1 \leq \ell \leq k \\ \ell \neq i}} (\lambda_i - \lambda_\ell)} & : j = k \end{cases}$$

First let's consider the denominator. As $\chi_u(x) = x^k - a_{k-1}x^{k-1} + \cdots + a_0 = \prod_{i\in[k]} (x - \lambda_i)$ its derivative $\chi'_u(x) = kx^{k-1} - (k-1)a_{k-1}x^{k-2} + \cdots + a_1 = \sum_{i\in[k]} \prod_{j\in[k], j\neq i} (x - \lambda_j)$. Now we observe that the denominator is just $\lambda_i \chi'_u(\lambda_i) \in \mathbb{Q}_{exp(m)}(\lambda_i)$ as a_i's are m bit rationals.

Now notice that all elements of the k-th column of the Vandermonde matrix is just 1 scaled by a $\mathsf{poly}(m)$-bit number. The $(k-1)$-th column is given by,

$$b_{i,k-1} = (-1)^{k-2} \sum_{1\leq \ell_1 \leq k; \ell_1 \neq i} \lambda_{\ell_1} = (-1)^{k-2}(a_{k-1} - \lambda_i)$$

where a_{k-1} is the coefficient of x^{k-1} in the characteristic polynomial of the recurrence via Vieta's identities [32].

Similarly we use the fact that the coefficient of x^{k-2} in the characteristic polynomial is the elementary symmetric polynomial of its roots (namely the $\{\lambda_i\}_{i=1}^k$) and rewrite the expression of the Vandermonde inverse above to get

$$\begin{aligned} b_{i,k-2} &= (-1)^{k-3}\Big(\sum_{1\leq \ell_1, \ell_2 \leq k; \ell_1, \ell_2 \neq i} \lambda_{\ell_1}\lambda_{\ell_2} \Big) = (-1)^{k-3}\Big(a_{k-2} - \lambda_i \sum_{1\leq \ell \leq k; \ell \neq i} \lambda_\ell\Big) \\ &= (-1)^{k-3}(a_{k-2} - (-1)^{k-1}\lambda_i(a_{k-1} - \lambda_i)) \end{aligned}$$

Proceeding inductively, let us assume that $b_{i,j+1} \in \mathbb{Q}(\lambda_i)$. Let $e_k^j(x_1, \ldots, x_k)$ denote the j-th elementary symmetric polynomial in the variables $\{x_1, \ldots, x_k\}$. Now we have,

$$\begin{aligned} b_{i,j} &= \left((-1)^{j-1} \sum_{\substack{1 \leq \ell_1 < \ldots < \ell_{k-j} \leq k \\ \ell_1, \ldots, \ell_{k-j} \neq i}} \lambda_{\ell_1} \cdots \lambda_{\ell_{k-j}} \right) \\ &= (-1)^{j-1}(e_k^j(\lambda_1, \ldots, \lambda_k) - \lambda_i b_{i,j+1}) \\ &= (-1)^{j-1}(a_{k-j} - \lambda_i b_{i,j+1}) \end{aligned}$$

which indeed shows that $b_{i,j} \in \mathbb{Q}(\lambda_i)$.

To see that $\frac{1}{2^{\mathcal{O}(m)}} \leq c_i \leq 2^{\mathcal{O}(m)}$, just notice that c_i is obtained as a rational linear combination of powers of λ_i. Since λ_i are the roots of the characteristic polynomial of the LRS we started out with, their magnitude is upper bounded by $2^{\mathcal{O}(m)}$ (which is also the

height of the characteristic polynomial) and lower bounded by $1/2^{\mathcal{O}(m)}$ (to see this just notice that if α is a root of $\chi_u(x)$, then $1/\alpha$ is a root of $x^k\chi_u(1/x)$, where k is the degree of $\chi_u(x)$). This concludes the proof. ◀

Armed with these lemmas, we are now able to refine Proposition 1 considerably. Most standard references on the subject [31, 16] observe that the coefficients $c_{j\ell}$ reside in a finite extension of the rationals, namely $\bar{\mathbb{Q}}(\lambda_1, \ldots, \lambda_n)$. We have the following

▶ **Lemma 6.** *Given a LRS* $\boldsymbol{u} = \langle u_n \rangle$*, the exponential polynomial solution is:* $u_n = \sum_{j=1}^{r} p_j(n)\lambda_j^n$ *where* $p_j(n) = \sum_{\ell=0}^{e_j-1} c_{j\ell}n^\ell$*. Then, the defining coefficients* $c_{j\ell} \in \mathbb{Q}_{\exp(m)}(\lambda_j)$*, where* m *is the size of the given LRS instance.*

Proof. We want to show here that all the coefficients appearing in the polynomials $p_j(n)$ in the exponential polynomial solution belong to $\mathbb{Q}(\lambda_j)$ and are expressible in $\mathsf{poly}(m)$-bits. To show this we rely on Lemmas 4 and 5. When the LRS is simple, we have from Lemma 5 that each coefficient c_i in the exponential polynomial solution $u_n = \sum_{j=1}^{r} c_j\lambda_j^n$ belongs to $\mathbb{Q}_{\exp(m)}(\lambda_j)$. From Lemma 4, we can write $\mathbf{u} = \sum_{i=0}^{e-1} n^i\mathbf{u^i}$, where each $\mathbf{u^i}$ is a simple LRS of order d_i whose initial conditions and recurrence coefficients belong to $\mathbb{Q}_{\exp(m)}$. Let $u_n^i = \sum_{\ell'=1}^{d_i} c_{i\ell'}\lambda_{i\ell'}^n$. From Lemma 5, we have that each $c_{i\ell'} \in \mathbb{Q}_{\exp(m)}(\lambda_{i\ell'})$, since size of $\mathbf{u^i}$ is also $\mathsf{poly}(m)$. Together with Equation 1, we have: $u_n = \sum_{j=1}^{r}\left(\sum_{\ell=0}^{e_j-1} c_{j\ell}n^\ell\right)\lambda_j^n = \sum_{i=0}^{e-1} n^i\left(\sum_{\ell'=1}^{d_i} c_{i\ell'}\lambda_{i\ell'}^n\right)$

Note from the expression on the left that $c_{j\ell}$ is exactly the coefficient of $n^\ell\lambda_j^n$. By comparing the two expressions, we see that the coefficient of $n^\ell\lambda_j^n$ in the expression on the right is exactly $c_{i\ell'}$ where $i = \ell$ and $\lambda_{i\ell'} = \lambda_j$. Thus we have that each $c_{j\ell} \in \mathbb{Q}_{\exp(m)}(\lambda_j)$, since we noted that each $c_{i\ell'} \in \mathbb{Q}_{\exp(m)}(\lambda_{i\ell'})$. ◀

The lemma above is implicit in the work of Cai [11] on computing Jordan forms of matrices. However, we give a direct proof using elementary techniques. The main purpose of Lemma 6 is to provide a good upper and lower bound on the magnitude of the coefficients of $p_j(n)$ in the exponential polynomial solution to *any* LRS. While, this is of no consequence with respect to decidability, as we elaborate in the forthcoming sections, it affects the computational complexity of the problem considerably.

4 Real characteristic roots

In this section we analyze the exact complexity of the Skolem problem for LRS($\mathbb{R}$) and obtain an upper bound of $\mathsf{NP}^{\mathsf{RP}}$.

▶ **Theorem 7.** *Given an LRS* u *with real characteristic roots, Skolem problem is decidable in* $\mathsf{NP}^{\mathsf{RP}}$

Proof. Let $\mathbf{u}$ be a LRS of order k with distinct (not considering repeated) roots $\lambda_1, \ldots, \lambda_r \in \mathbb{R}$. Let m denote the number of bits required to specify the LRS $\mathbf{u}$. We first assume that all roots are positive. We then show a decision procedure for zero testing in this case by showing that there is an exponential bound after which all terms of the LRS are non-zero, and thus it is enough to only consider terms before this bound. In fact, we show this for a class of real exponential polynomials:

▶ **Lemma 8.** *Consider a real exponential polynomial* $\mathbf{u}$ *given by* $u_n = \sum_{j=1}^{r} p_j(n)\lambda_j^n$ *s.t.,*

1. $\lambda_j \in \mathbb{R}^+$ *are the (distinct) absolute values of the roots of a polynomial* $\chi_u(x)$ *whose coefficients are expressible in* m *bits,*
2. *the coefficients of all the polynomials* $p_j(n)$ *are expressible in* $\mathsf{poly}(m)$ *bits,*

3. *r and the degrees of each $p_j(n) \leq m$.*
where m is a size parameter. Then there exists $N = 2^{m^{\mathcal{O}(1)}}$ such that either (i) for all $n > N$, $u_n > 0$, or (ii) for all $n > N$, $u_n < 0$.

Proof. Note first that the number of bits required to specify the real exponential polynomial as the coefficients of $p_j(n)$, λ_j's and r is $\mathsf{poly}(m)$. Let us assume $\lambda_1 > \lambda_2 > \cdots > \lambda_r > 0$. Now we can write:

$$\frac{u_n}{\lambda_1^n} = p_1(n)\left(\frac{\lambda_1}{\lambda_1}\right)^n + \sum_{j=2}^{r} p_j(n)\left(\frac{\lambda_j}{\lambda_1}\right)^n = p_1(n)1^n + \sum_{j=2}^{r} p_j(n)\rho_j^n$$

where $\rho_j = \lambda_j/\lambda_1$ and $\rho_j \in (0,1)$ for $2 \leq j \leq r$. Let $r(n) = \sum_{j=2}^{r} p_j(n)\rho_j^n$. We first place bounds on $r(n)$:

▷ Claim 9. There exist $\epsilon \in (0,1)$ and N, where $1/\epsilon = 2^{m^{\mathcal{O}(1)}}$ and $N = 2^{m^{\mathcal{O}(1)}}$ such that for every $n > N$, $|r(n)| < (1-\epsilon)^n$.

Proof. Note that $|r(n)| \leq \sum_{j=2}^{r} |p_j(n)|{\rho_j}^n$. We are considering a class of real exponential polynomials where the degree of any polynomial p_j is at most m, and that for every $j \in [r]$, the defining coefficients of $p_j(n)$ are upper and lower bounded by numbers expressible in $\mathsf{poly}(m)$-bits. Hence the height of $p_j(n)$ is $2^{\mathcal{O}(m)}$, and putting the height and degree bounds together, we can upper and lower[3] bound $|p_j(n)|$ by $2^{\mathcal{O}(m)}n^m$.

Now we are left with obtaining bounds on ρ_j to have a bound on $r(n)$. Consider a polynomial χ'_u which has roots $1, \rho_2, \ldots, \rho_r$. The roots of χ'_u are the roots of the polynomial χ_u scaled by a constant factor $1/\lambda_1$. Since χ_u has size $\mathcal{O}(m)$, so does χ'_u. Thus bounds on its degree $d = \mathcal{O}(m)$ and height $H = 2^{\mathcal{O}(m)}$ follow. We now use Mignotte's root separation bound (Lemma 3). When applied to χ'_u this gives: $|1-\rho_j| > \frac{\sqrt{6}}{d^{(d+1)/2}H^{d-1}} = \frac{1}{2^{m^{\mathcal{O}(1)}}}$. Since $\rho_j \in (0,1)$, we have ${\rho_j}^n < (1-2^{-m^{\mathcal{O}(1)}})^n$. Now observe that,

$$|r(n)| = \sum_{j=2}^{r} |p_j(n)||\rho_j|^n \leq \sum_{j=2}^{r} 2^{\mathcal{O}(m)}n^{\mathcal{O}(m)}(1-2^{-m^{\mathcal{O}(1)}})^n$$

We also have that the polynomials $p_j(n)$ for $j \in [n]$ have coefficients upper and lower bounded by values that are expressible in $\mathsf{poly}(m)$-bits. Since an exponential function eventually (after a certain N) dominates a polynomial function, we can find an ϵ such that $|r(n)| < (1-\epsilon)^n$ for all $n > N$. Since the degree of the polynomial is $\mathsf{poly}(m)$, the height of the polynomial is $\mathsf{poly}(m)$-bit bounded, and the base of the (decaying) exponential function is $(1-2^{-m^{\mathcal{O}(1)}})$, $1/\epsilon$ and N are exponentially bounded in m. ◁

We now proceed to prove Lemma 8. The n-th term of the sequence is given by: $\frac{u_n}{\lambda_1^n} = p_1(n) + r(n)$. From Claim 9, there exist $\epsilon \in (0,1)$ and N_1, where $1/\epsilon = 2^{m^{\mathcal{O}(1)}}$ and $N_1 = 2^{m^{\mathcal{O}(1)}}$ such that for every $n > N_1$, $|r(n)| < (1-\epsilon)^n$. We also know from Lemma 6 that the coefficients of $p_1(n)$ are $\mathsf{poly}(m)$-bit bounded. Thus, similar to the proof of Claim 9, we can find an exponentially bounded $N_2 = 2^{m^{\mathcal{O}(1)}}$ such that for all $n > N_2$, $|p_1(n)| > (1-\epsilon)^n$. Note here that Lemma 6 was crucial in obtaining the exponential bound on N_2, since without

[3] Notice that just an upper bound is not sufficient, as since the coefficient are algebraic numbers, it might be the case that the coefficients are too small, which might hurt the complexity of our algorithm. However this turns out not to be the case, thanks to Lemma 6

it only a double exponentially smaller lower bound on the coefficients of $p_1(n)$ is known, which does not translate to an exponential bound on N_2.

Putting these two together, we observe that if $p_1(n)$ has a positive leading coefficient, we have that for all $n \geq \mathsf{max}\{N_1, N_2\}$, $u_n > p_1(n) - |r(n)| > (1-\epsilon)^n - (1-\epsilon)^n > 0$. If $p_1(n)$ has a negative leading coefficient, then consider $-u_n = q(n) - r(n)$, where $q(n) = -p(n)$ has a positive leading coefficient, and again, we can have that for all $n \geq \mathsf{max}\{N_1, N_2\}$, $-u_n > 0$. This means that all terms of the sequence beyond $N = \mathsf{max}\{N_1, N_2\}$ are either all strictly positive or all strictly negative. ◀

We now show how to decide Skolem problem for LRSs with positive real roots using Lemma 8. Recall that m is the size of the LRS. Now observe that

1. u_n is an exponential polynomial solution (see Equation 1).
2. λ_j's are positive reals, and roots of $\chi_u(x)$, the characteristic polynomial of the LRS. Since the coefficients of $\chi_u(x)$ are part of the input, they are expressible in m bits.
3. r and the degrees of the polynomials are all $\leq k$, which in turn is $\leq m$.

Thus we can apply Lemma 8. Thus we see that deciding if there is an n such that $u_n = 0$ amounts to checking the only terms u_n where $n < N$, where N is the exponentially bounded constant in Lemma 8. Since N can be represented in polynomial number of bits in the size of the input LRS, we can guess $0 \leq n < N$ in NP and check if $u_n = 0$ by iterated squaring of the companion matrix of the LRS, which can be represented as a small circuit. We can now invoke the randomized algorithm for circuit zero testing (commonly called EqSLP, see for example [5]). This places Skolem problem for LRSs with positive real characteristic roots in $\mathsf{NP}^{\mathsf{EqSLP}}$ which is in $\mathsf{NP}^{\mathsf{RP}}$, since $\mathsf{EqSLP} \subseteq \mathsf{coRP}$. To reduce the general case of real roots (LRS with both positive and negative real roots) to the case of positive real roots discussed above, notice that since reals form an ordered field, any LRS with real characteristic roots can have at most two dominant roots: say λ_1 and $-\lambda_1$. We have: $u_n = p_1(n)\lambda_1^n + p_2(n)(-\lambda_1)^n + \sum_{j=3}^{r} p_j(n)\lambda_j^n$. Consider the sequences $\mathbf{v}$ and $\mathbf{w}$ defined by $v_n = u_{2n}$ and $w_n = u_{2n+1}$

$$v_n = u_{2n} = (p'_1(n) + p'_2(n))(\lambda_1^2)^n + \sum_{j=3}^{r} p'_j(n)(\lambda_j^2)^n$$

$$w_n = u_{2n+1} = (p''_1(n) + p''_2(n))(\lambda_1^2)^n + \sum_{j=3}^{r} p''_j(n)(\lambda_j^2)^n$$

where $p'_1(n) = p_1(2n)$, $p''_1(n) = \lambda_1 p_1(2n)$, and so on. Since the expressions for v_n and w_n are in the exponential polynomial form, the sequences $\mathbf{v}$ and $\mathbf{w}$ are linear recurrences. Observing the exponential polynomial solution further reveals their characteristic roots are squares of the characteristic roots of $\mathbf{u}$, and thus are positive reals. Notice that deciding Skolem problem for $\mathbf{u}$ is equivalent to deciding Skolem problem for both $\mathbf{v}$ and $\mathbf{w}$. Since the LRSs $\mathbf{v}$ and $\mathbf{w}$ can be computed in polynomial time from $\mathbf{u}$, and using the fact that Skolem problem can be decided in $\mathsf{NP}^{\mathsf{RP}}$ for LRSs with positive real roots, we have that Skolem problem for LRSs with real algebraic characteristic roots is also in $\mathsf{NP}^{\mathsf{RP}}$. ◀

5 Roots of reals

In this section, we use the results proved in the previous sections, to finally show the following main result of this paper.

▶ **Theorem 10.** *Skolem problem for* $\mathrm{LRS}(\mathrm{r}\mathbb{R})$ *can be decided in* $\mathsf{NP}^{\mathsf{RP}}$

At a high level, the main idea is to show that the Skolem problem for an LRS $\mathbf{u} \in \text{LRS}(\text{r}\mathbb{R})$ can be reduced to testing for zeros of a set of *real* exponential polynomials, for which we can then appeal to Lemma 8. We divide the proof into two parts: first we show the result for *simple* LRS and then, we use splitting lemma (Lemma 4) to solve the general case. We start with a technical lemma about the phases of roots in a $\text{LRS}(\text{r}\mathbb{R})$ instance:

▶ **Lemma 11.** *For an irreducible polynomial which factors as* $p(x) = \prod_{j=1}^{d}(x - \alpha_j\omega_j)$*, where* $\alpha_j \in \overline{\mathbb{Q}} \cap \mathbb{R}$ *and* $\omega_j = e^{\frac{2\pi\iota s_j}{t_j}}$ *is a* t_j*-th primitive root of unity, we have for all* $j, t_j = O(d^2 \log d)$ *and* $lcm\{t_j\}_{j=1}^{d} = 2^{O(d \log d)}$*.*

Proof. Let ω be some ω_j, which is the $t_j = t^{th}$ primitive root of unity. Since $\alpha\omega$ is a root of an irreducible polynomial of degree d, we have $deg_{\mathbb{Q}}(\alpha\omega) = d$. Notice that $\alpha\omega^{-1}$ is a conjugate of $\alpha\omega$ and hence also satisfies $deg_{\mathbb{Q}}(\alpha\omega^{-1}) = d$. Hence we have $[\mathbb{Q}(\alpha\omega, \alpha\omega^{-1}) : \mathbb{Q}] \leq d^2$, by the multiplicative property of field extensions. This gives $(\alpha\omega)/(\alpha\omega^{-1}) = \omega^2 \in \mathbb{Q}(\alpha\omega, \alpha\omega^{-1})$. Hence $deg_{\mathbb{Q}}(\omega) \leq 2d^2$. But $\phi(t) = deg_{\mathbb{Q}}(\omega)$. Now, a well-known lower bound for the Euler totient function is $\phi(t) \geq \Omega\left(\frac{t}{\log t}\right)$ (see for example Theorem 328 in [19]) and together with $\phi(t) \leq 2d^2$, this yields $t = O(d^2 \log d)$ and the LCM bound follows. ◀

The lemma above can be considered as a weak generalization of the well-known fact that any polynomial with degree-d^2 cannot have as one of its roots a d'-th primitive root of unity, whenever $d' > d$.

5.1 The case of Simple LRS($\text{r}\mathbb{R}$)

We first set up some notation. Let $\mathbf{v} = \{v_n\}_{n\geq 0} \in \text{LRS}(\text{r}\mathbb{R})$ be a simple LRS of order k having size m. Let the characteristic roots of the LRS $\mathbf{v}$ be $\alpha_j\omega_j$, for $j \in [k]$, where $\alpha_j \in \mathbb{R}^+$, and $\omega_j = e^{\iota 2\pi s_j/t_j}$ (where $s_j \in \mathbb{Z}$ and $t_j \in \mathbb{Z}^+$ and $\mathsf{lcm}(|s_j|, t_j) = 1$) is the t_j^{th} primitive root of unity. For a multiset S, define $\mathsf{supp}(S)$ to be the set obtained from S. Let A be the set of absolute values of the characteristic roots, $A = \mathsf{supp}(\{\alpha_j\}_{j=1}^{k}) = \{\beta_j\}_{j=1}^{k'}$. Define the set $T = \mathsf{supp}(\{t_j\}_{j=1}^{k})$. Let $K = \mathsf{lcm}(\{t \mid t \in T\})$. We call a number from the set $\{0, \ldots, K-1\}$ as the *global phase* of the LRS and numbers from T as the *local phases*.

The main idea behind our algorithm is that once a global phase ℓ is fixed, the terms of the LRS v_n where $n \equiv \ell \mod K$ can be captured by a real exponential polynomial $q_\ell(n)$. More formally we have the following,

▶ **Lemma 12.** *Given a simple LRS* $\boldsymbol{v} \in \text{LRS}(\text{r}\mathbb{R})$*, for every* $\ell \in \{0, 1, \ldots, K-1\}$*, there exists a* real *exponential polynomial* $q_\ell(n) = \sum_{j=1}^{k'} c_{\ell j}\beta_j^n$ *such that* $v_n = q_\ell(n)$ *whenever* $n \equiv \ell \mod K$*. Further* $c_{\ell j} \in \mathbb{Q}_{\exp(m)}(\beta_j)$ *and can be computed in* $\mathsf{poly}(m)$ *time.*

The crucial point to establish is that the coefficients of this real exponential polynomial are also bounded by polynomially many bits. Once we have this, checking the existence of a Skolem zero reduces to zero testing of a real exponential polynomial, as was done in Lemma 8. Armed with Lemma 12, we can now prove

▶ **Theorem 13.** *The Skolem problem for simple LRS from the class* $\text{LRS}(\text{r}\mathbb{R})$ *is decidable in* $\mathsf{NP}^{\mathsf{RP}}$*.*

Proof. The following algorithm takes as input the LRS $\mathbf{v}$ and outputs "yes" if and only if $v_n = 0$ for some n:

1. Guess the global phase, i.e., a value ℓ from $\{0, 1, \ldots, K-1\}$.

2. Compute a real exponential polynomial $q_\ell(n) = \sum_{j=1}^{k'} c_{\ell j}\beta_j^n$ such that $v_n = q_\ell(n)$ for $n \equiv \ell \mod K$. We will show in Lemma 12 below that each $c_{\ell j} \in \mathbb{Q}_{exp(m)}(\beta_j)$ and can be computed in $\mathsf{poly}(m)$ time.
3. Use the algorithm of Lemma 8 to guess a zero n' of the real exponential polynomial $q_\ell(n)$. If $q_\ell(n') = 0$ and $n' \equiv \ell \mod K$, then output "yes", else output "no".

Correctness. Suppose there is a Skolem zero n' such that $v_{n'} = 0$, and that $n' \equiv \ell \mod K$. Then the algorithm correctly guesses ℓ and the zero n' of $q_\ell(n)$. On the other hand if the algorithm finds an n' in step 3 such that $n' \equiv \ell \mod K$, and $q_\ell(n') = 0$, then by Lemma 12 it holds that $v_{n'} = q_\ell(n') = 0$. Thus n' is a Skolem zero.

Complexity. Guessing ℓ, n' and computing $q_\ell(n)$ can be done in polynomial time. This is due to Lemma 12 and because $\ell \leq 2^{\mathsf{poly}(k)}$ due to Lemma 11. Due to the bounds established in Lemma 8, checking if $n' \equiv \ell \mod K$ and if $q_\ell(n') = 0$ can be done in RP. Thus, this algorithm runs in $\mathsf{NP}^{\mathsf{RP}}$. ◀

It remains to prove Lemma 12.

Proof of Lemma 12. We fix some $\ell \in \{0, 1, \ldots, K-1\}$. From the exponential polynomial solution (Definition 1), we have $v_n = \sum_{j=1}^{k} c_j(\alpha_j\omega_j)^n$. For $n \equiv \ell \mod K$, we have that $n \equiv \ell \mod t$, for every $t \in T$. Thus,

$$v_n = \sum_{j=1}^{k} c_j\omega_j^n\alpha_j^n = \sum_{j=1}^{k} c_j\omega_j^{\ell \bmod t_j}\alpha_j^n$$

Grouping terms by same α_j's, we have that

$$v_n = \sum_{j=1}^{k'} \Big(\sum_{j':\alpha_{j'}=\beta_j} c_{j'}\omega_{j'}^{\ell \bmod t_{j'}} \Big)\beta_j^n \tag{2}$$

Now we show that the coefficient of β_j^n in the above expression is in $\mathbb{Q}_{exp(m)}(\beta_j)$ for each j. To do this, we first divide up the characteristic roots of the LRS into sets defined as follows, for each $\alpha \in A$ and $t \in T$:

$$S_t^\alpha = \{j \mid \alpha_j = \alpha, \omega_j \text{ is a } t_j^{th} \text{ primitive root of unity s.t. } t_j = t\}$$
$$S_{\preceq t}^\alpha = \{j \mid \alpha_j = \alpha, \omega_j \text{ is a } t_j^{th} \text{ primitive root of unity s.t. } t_j \preceq t\}$$

where $\preceq$ is the partial order given by $a \preceq b \iff a \mid b$. The utility of these sets is that we have $j \in S_{\preceq t}^\alpha$, if and only if $(\alpha_j\omega_j)^t = \alpha^t$. We now show:

▷ Claim 14. For every $\alpha \in A$ and $t \in T$, the constants $c_{\alpha t} = \sum_{j \in S_t^\alpha} c_j\omega_j^\ell$ and $d_{\alpha t} = \sum_{j \in S_{\preceq t}^\alpha} c_j\omega_j^\ell$ are in $\mathbb{Q}_{exp(m)}(\alpha)$, and can be computed in $\mathsf{poly}(m)$ time.

Proof. We first show that $d_{\alpha t} \in \mathbb{Q}_{\exp(m)}(\alpha)$ can be computed in $\mathsf{poly}(m)$-time. Let $\mathsf{supp}(\{(\alpha_j\omega_j)^t \mid j \in [k]\}) = \{\tau_j \mid j \in [h]\}$. We proceed by grouping terms in the exponential polynomial solution for v_n for the first h values of n where $n \equiv \ell \mod t$. Specifically, for each $0 \leq g < h$, we write $\sum_{j=1}^{k} c_j(\alpha_j\omega_j)^{gt+\ell} = v_{gt+\ell}$. We rewrite this equation as follows:

$$\sum_{j=1}^{k} c_j(\alpha_j\omega_j)^{\ell-t}[(\alpha_j\omega_j)^t)^{g+1}] = v_{gt+\ell}$$

Grouping terms according with same $(\alpha_j\omega_j)^t$, we get:

$$\sum_{j=1}^{h}\left(\sum_{j':(\alpha_{j'}\omega_{j'})^t=\tau_j} c_{j'}(\alpha_{j'}\omega_{j'})^{\ell-t}\right)\tau_j^{g+1} = v_{gt+\ell}$$

Denoting by c'_j the expression in the bracket, we see that we have h equations in h variables, namely the c'_j. We get the following linear system, $Tc' = V$, where $T_{ij} = \tau_j^i$ for $1 \le i, j \le h$ and $c' = [c'_1 c'_2 \dots c'_h]^T$ and $V = [v_\ell v_{t+\ell} \dots v_{(h-1)t+\ell}]^T$. Notice that T is a Vandermonde matrix. Also, since we know t is polynomially bounded in k, by Proposition 2 the values $v_\ell, \dots, v_{(h-1)t+\ell}$ are in $\mathbb{Q}_{\exp(m)}$. We can now invoke the following lemma with $\lambda_j = \tau_j$ and $\gamma_j = \alpha_j\omega_j$ to conclude that each constant $c'_j \in \mathbb{Q}_{exp(m)}(\tau_j)$.

▶ **Lemma 15.** *Let u be a sequence of order k, whose exponential polynomial solution is given by $u_n = \sum_{i=1}^{k} c_i\lambda_i^n$ and initial conditions are given by $u_1, \dots, u_k$. Let $\chi(x) = x^{k'} - a_{k'-1}x^{k'-1} + \dots + a_0$ be a polynomial with $a_i \in \mathbb{Q}_{exp(m)}$, roots $(\gamma_j)_{j=1}^{k'}$, $k \le k' \le poly(k)$. Define the set $\{\lambda_i\}_{i=1}^{k}$ from the multiset $\{\gamma_j^t\}_{j=1}^{k'}, t \le \mathsf{poly}(k)$. Then the coefficients c_i in the exponential polynomial solution have the property that $c_i \in \mathbb{Q}_{exp(m)}(\lambda_i)$.*

Proof. For all λ_i there exists a j such that $\lambda_i = \gamma_j^t$. As $t \le \mathsf{poly}(m)$ by Proposition 3 all λ_i's have a monic polynomial over $\mathbb{Q}$ with coefficients from $\mathbb{Q}_{\exp(m)}$.

Let $conj(\gamma)$ be the set of all galois conjugates of γ over $\mathbb{Q}$. Let $\{\delta_i\}_{i=1}^{k''} = \cup_{i=1}^{k'} conj(\lambda_i)$. Each λ_i has degree $\mathsf{poly}(k)$ as it is a $\mathsf{poly}(k)$ power of some γ_j which itself has degree $\mathsf{poly}(k)$. Hence $k'' \le k' \max_i(deg(\lambda_i)) \le \mathsf{poly}(k)$. The product of all the monic polynomials of λ_i's has exactly the list of roots $\{\delta_i\}_{i=1}^{k''}$ with no repeated roots. Also this product has coefficients from $\mathbb{Q}_{\exp(m)}$ as we are multiplying at most $\mathsf{poly}(k) \le \mathsf{poly}(m)$ polynomials of $\mathsf{poly}(k) \le \mathsf{poly}(m)$ degree each having coefficients from $\mathbb{Q}_{\exp(m)}$.

As the set $\{\delta_i\}_{i=1}^{k''}$ is a superset of $\{\lambda_i\}_{i=1}^{k}$, by Lemma 5 we have a unique solution $u_n = \sum_{j=1}^{k''} w_j\delta_j^n$ with $w_j \in \mathbb{Q}_{exp(m)}(\delta_j)$. But as we also have $u_n = \sum_{i=1}^{k} c_i\lambda_i^n$ it must be the case that $w_j = 0$ if $\delta_j \notin \{\lambda_i\}_{i=1}^{k}$ and $w_j = c_i$ when $\delta_j = \lambda_i$ some i. So we have $u_n = \sum_{i=1}^{k} c_i\lambda_i^n$ with $c_i = w_j \in \mathbb{Q}_{exp(m)}(\delta_i) = \mathbb{Q}_{exp(m)}(\lambda_i)$. ◀

We observed that $j \in S^{\alpha}_{\preceq t} \iff (\alpha_j\omega_j)^t = \alpha^t$. Let $\alpha^t = \tau_j$ for some $1 \le j \le h$. Thus all j' for which $(\alpha_{j'}\omega_{j'})^t = \tau_j$ are in fact in $S^{\alpha}_{\preceq t}$. Hence the constant

$$d_{\alpha t} = \sum_{j\in S^{\alpha}_{\preceq t}} c_j\omega_j^\ell = \sum_{j\in S^{\alpha}_{\preceq t}} \frac{c_j(\alpha_j\omega_j)^{\ell-t}(\alpha_j\omega_j)^t}{(\alpha_j)^\ell} = \sum_{j\in S^{\alpha}_{\preceq t}} \frac{c_j(\alpha_j\omega_j)^{\ell-t}\alpha^t}{\alpha^\ell} = c'_j\alpha^{t-\ell}$$

Since $c'_j \in \mathbb{Q}_{exp(m)}(\tau_j) = \mathbb{Q}_{exp(m)}(\alpha^t)$, and $t \le \mathsf{poly}(k)$, by Proposition 3 we have in fact $c'_j \in \mathbb{Q}_{exp(m)}(\alpha)$. Hence $d_{\alpha t} = c'_j(\alpha)^{t-\ell} \in \mathbb{Q}_{exp(m)}(\alpha)$. It is clear that since the expression for c'_j involves $\mathsf{poly}(m)$ operations on $\mathsf{poly}(m)$-sized algebraic numbers, $d_{\alpha t}$ can be computed in $\mathsf{poly}(m)$-time.

We now show that $c_{\alpha t} \in \mathbb{Q}_{\exp(m)}(\alpha)$ and can be computed in $\mathsf{poly}(m)$ time. For the minimal nodes of the partial order $\preceq$, i.e. for prime t, it is the case that $c_{\alpha t} = d_{\alpha t}$. Thus for such t, we directly have that $c_{\alpha t}$ is in $\mathbb{Q}_{exp(m)}(\alpha)$, and can be computed in $\mathsf{poly}(m)$ time.

For any other t we can compute $c_{\alpha t}$ recursively by using the equation:

$$c_{\alpha t} = \sum_{j\in S^{\alpha}_{t}} c_j\omega_j^\ell = \sum_{j\in S^{\alpha}_{\preceq t}} c_j\omega_j^\ell - \sum_{t'\prec t}\sum_{j'\in S^{\alpha}_{t'}} c_{j'}\omega_{j'}^\ell = d_{\alpha t} - \sum_{t'\prec t} c_{\alpha t'}$$

Since for all $t \in T$, $t \leq \mathsf{poly}(k)$, we have that the longest chain in this partial order is of length $O(\log k) = O(\log m)$. Using an inductive argument together with the proof that $d_{\alpha t} \in \mathbb{Q}_{\exp(m)}(\alpha)$, we see that all the relevant quantities appearing in the above equation are in $\mathbb{Q}_{exp(m)}(\alpha)$. This implies that $c_{\alpha t} \in \mathbb{Q}_{\exp(m)}(\alpha)$ and can be computed in $\mathsf{poly}(m)$ time. ◁

We now revisit Equation 2:

$$v_n = \sum_{j=1}^{k'} \Big(\sum_{j' : \alpha_{j'} = \beta_j} c_{j'} \omega_{j'}^{\ell \bmod t_{j'}} \Big) \beta_j^n = \sum_{j=1}^{k'} c_{\ell j} \beta_j^n$$

Here we define $c_{\ell j}$ as the coefficient of β_j^n in Equation 2. Roots with real part equal to β_j with different local phases contribute to $c_{\ell j}$. Separating roots according to different local phases, we have

$$c_{\ell j} = \sum_{j' : \alpha_{j'} = \beta_j} c_{j'} \omega_{j'}^{\ell \bmod t_{j'}} = \sum_{t \in T} \sum_{j' \in S_t^{\beta_j}} c_{j'} \omega_{j'}^{\ell \mod t} = \sum_{t \in T} c_{\beta_j t}$$

Now it follows directly from Claim 14 that for every j, $c_{\ell j} \in \mathbb{Q}_{exp(m)}(\beta_j)$ and can be computed in $\mathsf{poly}(m)$ time since $t \leq \mathsf{poly}(k)$. This establishes Lemma 12. ◀

5.2 The general case

We now consider the general case of LRS in LRS(r$\mathbb{R}$) and show:

▶ **Theorem 16.** *Skolem problem for the class* LRS(r$\mathbb{R}$) *is decidable in* $\mathsf{NP}^{\mathsf{RP}}$.

Proof. In the general case, the exponential polynomial solution (Definition 1) for a LRS $\mathbf{u} \in$ LRS(r$\mathbb{R}$) takes the following form: $u_n = \sum_{j=1}^{r} p_j(n)(\alpha_i \omega_j)^n$ where $p_j(n) = \sum_{i=0}^{e_j - 1} p_{ji} n^i$. From Lemma 5 and 6, we know that every defining coefficient $p_{ji} \in \mathbb{Q}_{\exp(m)}(\alpha_j \omega_j)$. We can now use Lemma 4 to decompose LRS $\mathbf{u}$ as: $u_n = \sum_{i=0}^{e-1} n^i \mathbf{u^i}$ where each $\mathbf{u}^i$ is a simple LRS in LRS(r$\mathbb{R}$). As in proof of Theorem 13, once we fix $\ell \in \{0, 1, \ldots, K-1\}$, whenever $n \equiv \ell \bmod K$, we have from Lemma 12 that we can write $u_n^i = \sum_{j=1}^{k'} c_{\ell ij} \beta_j^n$ for every LRS $\mathbf{u^i}$, where each $c_{\ell ij} \in \mathbb{Q}_{exp(m)}(\beta_j)$. Thus, whenever $n \equiv \ell \bmod K$, we have: $u_n = \sum_{i=0}^{e-1} n^i \left(\sum_{j=1}^{k'} c_{\ell ij} \beta_j^n \right)$. ◀

Observe that the right hand side of the equation above is a real exponential polynomial $q_\ell(n)$ and $c_{\ell ij}$ is the coefficient of $n^i \beta_j^n$, where $c_{\ell ij} \in \mathbb{Q}_{exp(m)}(\beta_j)$. This allows us to invoke Lemma 8 to accomplish zero testing of the exponential polynomial $q_\ell(n)$ in $\mathsf{NP}^{\mathsf{RP}}$. We note that if there is an n such that $q_\ell(n) = 0$ and $n \equiv \ell \mod K$ then $u_n = 0$. On the other hand if there is an n such that $u_n = 0$, then $q_\ell(n) = 0$ where $\ell \in \{0, 1, \ldots, K-1\}$ is such that $n \equiv \ell \mod K$. Thus the above is an algorithm to decide Skolem for LRS(r$\mathbb{R}$) in $\mathsf{NP}^{\mathsf{RP}}$.

6 Revisiting NP-hardness

Blondel and Portier [9] proved that Skolem problem is NP-hard by a reduction from the *non-universality problem* for unary NFAs [27]. More recently, in [3], an alternate proof was obtained by a reduction from the subset sum problem. In this section, we provide yet another proof of NP-hardness, by directly reducing from 3-SAT. Given a 3-SAT formula ϕ over s

variables $x_1, \ldots, x_s$, we will construct an LRS y such that ϕ is satisfiable if and only if the LRS y has a Skolem zero, i.e., $\exists n \in \mathbb{N}\ y_n = 0$. Let $p_1, \ldots, p_s$ be the first s primes. By the Prime number theorem, the number of primes less than s is roughly $s/\log s$, and thus $p_s \sim s \log s$.

For each prime p_i, define an LRS u^i with order p_i given by:

$$u^i_n = \begin{cases} 0, \text{ for } 1 \leq n < p_i \\ 1, \text{ for } n = p_i \\ u^i_{n-p_i}, \text{ for } n > p_i \end{cases}$$

With 1 and 0 representing the boolean values true and false, we define a surjection f from $\mathbb{N}$ to the set of assignments to variables of ϕ as $f : \mathbb{N} \to \{0,1\}^s$, given by $f(n) = (a_1, \ldots, a_s)$ where $a_i = 1 \iff p_j | n$. The inverse map of an assignment, $f^{-1}(a_1, \ldots, a_s)$ is the set $\{n : p_j | n \iff a_j = 1\}$. For sequences u, v and w, we will denote $u = v$ to mean $\forall n \in \mathbb{N}\ u_n = v_n$, and $w = u + v$ to mean $\forall n \in \mathbb{N}\ w_n = u_n + v_n$. Let $\phi = C_1 \wedge \cdots \wedge C_m$. For a clause $C_i = v_{i_1} \vee v_{i_2} \vee v_{i_3}$ define the LRS y^{i_l} for $l = 1, 2, 3$ as follows:

$$y^{i_l} = \begin{cases} 1 - u^k, \text{ if } v_{i_l} = x_k \text{ for some } k \in \{1, \ldots, s\}, \\ u^k, \text{ if } v_{i_l} = \neg x_k \text{ for some } k \in \{1, \ldots, s\} \end{cases}$$

Define the sequences $y^i = y^{i_1} y^{i_2} y^{i_3}$ for $1 \leq i \leq m$, and $y = y^1 + \cdots + y^m$. Since the sum and product of LRS is a LRS (see Theorem 4.1 in [16]), y is also an LRS. Then we have:

▷ Claim 17. $y^i_n = 0$ if and only if $f(n)$ satisfies C_i.

Now one can argue that

▶ **Proposition 18.** *ϕ is satisfiable if and only if $\exists n$ s.t. $y_n = 0$.*

The order of y is at most $m(p_s)^3$, and thus is polynomial in the number of variables and the clauses and y can be constructed from an instance of 3-SAT in polynomial time. Thus, we have shown that the Skolem problem for integral LRS is at least as hard as 3-SAT, and hence NP-hard.

Weak vs Strong NP-hardness. A simple consequence of the above reduction is that we can now show that the Skolem problem remains NP-hard even when the initial values are given in unary, i.e., it is *strongly* NP*-hard wrt the initial values*. This follows since the initial values of the LRS y used in the construction above, are at most m in value, and thus can be represented in $\log m$ bits, as opposed to poly(m) bits.

7 Applications and Discussion

We have shown that for a natural and large subclass of recurrences namely LRS(r$\mathbb{R}$), the Skolem problem can be solved in $\mathsf{NP}^{\mathsf{RP}}$. This immediately implies effective bounds for two well-known questions on LRS, namely, *Positivity and Ultimate Positivity for* LRS(r$\mathbb{R}$). Given an LRS $\mathbf{u}$, the *positivity problem* asks to decide if $u_n > 0$ for all $n \in \mathbb{N}$. Similarly, the *ultimate positivity problem* asks if there exists $n_0 \in \mathbb{N}$ s.t., $u_n > 0$ for all $n > n_0$. Using the machinery that we developed in this paper, we obtain:

▶ **Corollary 19** (to Theorem 10). *Positivity and Ultimate positivity for* LRS(r$\mathbb{R}$) *can be decided in*[4] $\mathsf{coNP}^{\mathsf{PosSLP}}$.

Proof. We can reduce an LRS(r$\mathbb{R}$) instance to a exponentially bounded set of real exponential polynomials, and in such polynomials the sign of the dominant root dominates the sign after an n that is exponentially bounded in m. Hence if the LRS is negative or zero before an exponential point, we can guess such an n in NP and verify the sign using PosSLP. This solves both Positivity and Ultimate Positivity in the fourth level of the Counting Hierarchy. ◀

Ultimate Positivity and Positivity are known to be hard for $\forall\mathbb{R}$ (the *universal theory of reals*) for the class of simple LRS [24] and hence considered unlikely to be in the Counting Hierarchy [5]. An inspection of the proof in [24] reveals that the LRS constructed to show hardness have characteristic roots with phases that are irrational multiple of π and hence do not fall in the class LRS(r$\mathbb{R}$). Finally, as mentioned in the introduction we show three applications of our results to obtain complexity bounds for problems from three completely different areas.

- **Weighted automata:** In a recent result, Barloy et al. [7] define a subclass of LRS called *poly-rational sequences*, denoted by rational expressions closed under sum and product. They show that polynomially ambiguous weighted automata, copyless cost-register automata, rational formal series, and LRS whose eigenvalues are roots of rational numbers (called **PolyRat**, these are exactly those LRS where the characteristic roots are of the form λ, where $\lambda^n = r$ for some $r \in \mathbb{Q}$) are equivalent. They leave open the precise complexity of the Skolem problem for **PolyRat** sequences. We solve this problem, since Theorem 10 immediately implies that Skolem for PolyRat is in $\mathsf{NP}^{\mathsf{RP}}$.
- **Probabilistic finite automata (PFA):** A second application is in the language theoretic properties of unary probabilistic finite automata. Given the link between the Skolem problem and the Markov chain reachability problem [2], the work in [4] considers regularity of unary PFA, whose dynamics are described using Markov chains. Proposition 2 give the decidability of reachability and positivity problems in the special case where roots are distinct (i.e., the "simple" case) roots of real numbers. Again, from Theorem 7 we obtain that these problems are in $\mathsf{NP}^{\mathsf{RP}}$ and $\mathsf{NP}^{\mathsf{PosSLP}}$ respectively. One interesting line of future work would be to see if the techniques introduced in this paper would also help in showing complexity bounds for regularity problems, which is the focus of [4].
- **Hybrid systems:** Most reachability problems on hybrid systems are known to be undecidable. Two well-behaved decidable fragments here are o-minimal hybrid systems [22] (Theorem 6.2), [26] (Theorem 4.6) and linear-time invariant (LTI) systems [17] (Theorem 3.10). In both these cases, decidability is obtained by assuming that the eigenvalues of the matrix associated with the linear system are reals or roots of reals (for example, called *simple* LTI systems in [17]). Given that [17] proves that reachability for LTI systems is hard for both the Skolem and Positivity problems for LRS, this raises the question of the precise computational complexity of reachability in LTI systems. Whether the techniques introduced in this paper will yield more precise complexity bounds to the computability results in these papers is part of ongoing research.

Implicit in our $\mathsf{NP}^{\mathsf{RP}}$ algorithm for Skolem problem for LRS(r$\mathbb{R}$) is an effective bound, i.e a number $N \in \mathbb{N}$ which is $\exp(m)$ such that for all $n > N$, $u_n \neq 0$. Since if such an n can be effectively bounded by $\exp(m)$ for all LRS would imply the decidability (in fact

[4] Given an arithmetic circuit representing a number, the PosSLP problem introduced by Allender et al. [5] is to decide if the number is positive. It is known to be P-hard and lies in the *Counting Hierarchy*.

in $\mathsf{NP}^{\mathsf{RP}}$) for the Skolem problem, one interesting way to improve the hardness result in Section 6 would be to construct an explicit LRS for which the first zero provably occurs[5] after $n > \exp(\exp(m))$. We leave this as a challenging open question for future work. In an orthogonal recent work, Bell et al. [8] introduce a class multidimensional version of LRS (n-LRS). In their language, Skolem problem is the question of finding zeroes in a 1-LRS. The zeroness problem for n-LRS of depth 2 is NP-hard, but in general the problem of n-LRS of depth k is undecidable. It would be interesting to see if spectral restrictions such as ours could yield decidability for special cases of n-LRS.

References

1 Manindra Agrawal, S. Akshay, Blaise Genest, and P. S. Thiagarajan. Approximate verification of the symbolic dynamics of Markov chains. *J. ACM*, 62(1):2:1–2:34, 2015.

2 S. Akshay, Timos Antonopoulos, Joël Ouaknine, and James Worrell. Reachability problems for Markov chains. *Inf. Process. Lett.*, 115(2):155–158, 2015.

3 S. Akshay, Nikhil Balaji, and Nikhil Vyas. Complexity of restricted variants of Skolem and related problems. In *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, pages 78:1–78:14, 2017.

4 S. Akshay, Blaise Genest, Bruno Karelovic, and Nikhil Vyas. On regularity of unary probabilistic automata. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 8:1–8:14, 2016.

5 Eric Allender, Peter Bürgisser, Johan Kjeldgaard-Pedersen, and Peter Bro Miltersen. On the complexity of numerical analysis. *SIAM J. Comput.*, 38(5):1987–2006, 2009. doi:10.1137/070697926.

6 Alan Baker. *Transcendental number theory*. Cambridge university press, 1990.

7 Corentin Barloy, Nathanaël Fijalkow, Nathan Lhote, and Filip Mazowiecki. A robust class of linear recurrence sequences. In *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain*, pages 9:1–9:16, 2020.

8 Paul C Bell, Igor Potapov, and Pavel Semukhin. On the mortality problem: From multiplicative matrix equations to linear recurrence sequences and beyond. In *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

9 V. D. Blondel and N. Portier. The presence of a zero in an integer linear recurrent sequence is NP-hard to decide. In *Linear Algebra and its Applications*, pages 351–352. Elsevier, 2002.

10 J-Y Cai, Richard J Lipton, Robert Sedgewick, and AC-C Yao. Towards uncheatable benchmarks. In *[1993] Proceedings of the Eigth Annual Structure in Complexity Theory Conference*, pages 2–11. IEEE, 1993.

11 Jin-yi Cai. Computing Jordan normal forms exactly for commuting matrices in polynomial time. *International Journal of Foundations of Computer Science*, 5(03n04):293–302, 1994.

12 Ventsislav Chonev, Joël Ouaknine, and James Worrell. On the complexity of the orbit problem. *J. ACM*, 63(3):23:1–23:18, 2016. doi:10.1145/2857050.

13 Henri Cohen. *A course in computational algebraic number theory*, volume 138. Springer Science & Business Media, 2013.

14 Abraham De Moivre. The doctrine of chances. In *Annotated Readings in the History of Statistics*, pages 32–36. Springer, 2001.

15 Kousha Etessami and Mihalis Yannakakis. On the complexity of Nash equilibria and other fixed points. *SIAM Journal on Computing*, 39(6):2531–2597, 2010.

[5] This question was also raised by Kousha Etessami in a comment on a blogpost in [28]

16 Graham Everest, Alfred J. van der Poorten, Igor E. Shparlinski, and Thomas Ward. *Recurrence Sequences*, volume 104 of *Mathematical surveys and monographs*. American Mathematical Society, 2003. URL: http://www.ams.org/bookstore?fn=20&arg1=survseries&item=SURV-104.

17 Nathanaël Fijalkow, Joël Ouaknine, Amaury Pouly, João Sousa-Pinto, and James Worrell. On the decidability of reachability in linear time-invariant systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 77–86. ACM, 2019.

18 Georges Hansel. A simple proof of the Skolem-Mahler-Lech theorem. In *International Colloquium on Automata, Languages, and Programming*, pages 244–249. Springer, 1985.

19 Godfrey Harold Hardy, Edward Maitland Wright, et al. *An introduction to the theory of numbers*. Oxford university press, 1979.

20 Russell Impagliazzo and Avi Wigderson. P= bpp unless e has subexponential circuits: derandomizing the xor lemma. In *Proceedings of the 29th STOC*, pages 220–229, 1997.

21 Donald E Knuth. The art of computer programming. volume 1: Fundamental algorithms. *Pearson education*, 1997.

22 Gerardo Lafferriere, George J Pappas, and Shankar Sastry. O-minimal hybrid systems. *Mathematics of control, signals and systems*, 13(1):1–21, 2000.

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

24 Joël Ouaknine and James Worrell. Ultimate positivity is decidable for simple linear recurrence sequences. In *International Colloquium on Automata, Languages, and Programming*, pages 330–341. Springer, 2014.

25 Min Sha. Effective results on the Skolem problem for linear recurrence sequences. *Journal of Number Theory*, 197:228–249, 2019.

26 Omid Shakernia, George J Pappas, and Shankar Sastry. Decidable controller synthesis for classes of linear systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 407–420. Springer, 2000.

27 Larry J Stockmeyer and Albert R Meyer. Word problems requiring exponential time (preliminary report). In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 1–9. ACM, 1973.

28 Terence Tao. *Structure and randomness: pages from year one of a mathematical blog*. American Mathematical Society Providence, RI, 2008.

29 Prasoon Tiwari. A problem that is easier to solve on the unit-cost algebraic RAM. *J. Complexity*, 8(4):393–397, 1992. doi:10.1016/0885-064X(92)90003-T.

30 Nikolai Konstantinovich Vereshchagin. Occurrence of zero in a linear recursive sequence. *Mathematical Notes*, 38(2):609–615, 1985.

31 M.Hirvensalo V.Halava, T.Harju and J.Karhumäki. Skolem's problem on the border between decidability and undecidability. In *TUCS Technical Report Number 683*, 2005.

32 F Viete. Opera mathematica. 1579. *Reprinted Leiden, Netherlands*, 1646, 1970.

Efficient Parameterized Algorithms for Computing All-Pairs Shortest Paths

Stefan Kratsch
Humboldt-Universität zu Berlin, Germany
kratsch@informatik.hu-berlin.de

Florian Nelles
Humboldt-Universität zu Berlin, Germany
nelles@informatik.hu-berlin.de

Abstract

Computing all-pairs shortest paths is a fundamental and much-studied problem with many applications. Unfortunately, despite intense study, there are still no significantly faster algorithms for it than the $\mathcal{O}(n^3)$ time algorithm due to Floyd and Warshall (1962). Somewhat faster algorithms exist for the vertex-weighted version if fast matrix multiplication may be used. Yuster (SODA 2009) gave an algorithm running in time $\mathcal{O}(n^{2.842})$, but no combinatorial, truly subcubic algorithm is known.

Motivated by the recent framework of efficient parameterized algorithms (or "FPT in P"), we investigate the influence of the graph parameters clique-width (cw) and modular-width (mw) on the running times of algorithms for solving ALL-PAIRS SHORTEST PATHS. We obtain efficient (and combinatorial) parameterized algorithms on non-negative vertex-weighted graphs of times $\mathcal{O}(\mathsf{cw}^2\, n^2)$, resp. $\mathcal{O}(\mathsf{mw}^2\, n + n^2)$. If fast matrix multiplication is allowed then the latter can be improved to $\mathcal{O}(\mathsf{mw}^{1.842}\, n + n^2)$ using the algorithm of Yuster as a black box. The algorithm relative to modular-width is adaptive, meaning that the running time matches the best unparameterized algorithm for parameter value mw equal to n, and they outperform them already for $\mathsf{mw} \in \mathcal{O}(n^{1-\varepsilon})$ for any $\varepsilon > 0$.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Shortest paths

Keywords and phrases All-pairs shortest Paths, efficient parameterized Algorithms, parameterized Complexity, Clique-width, Modular-width

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.38

Related Version A full version of the paper is available at [17], `http://arxiv.org/abs/2001.04908`.

1 Introduction

ALL-PAIRS SHORTEST PATHS (APSP) is a fundamental and much-studied problem in the field of algorithmic graph theory. Next to the theoretical interest in the problem, ALL-PAIRS SHORTEST PATHS is important for many practical applications, e.g., it is closely related to several vertex centrality measures in networks (for example, the betweenness centrality of a vertex v is defined as the sum of the fraction of all-pairs shortest paths that pass through v). The ALL-PAIRS SHORTEST PATHS problem is also considered as the core of many routing problems and has applications for example in areas such as routing protocols, driving direction on web mappings, transportation, and traffic assignment problems, and many more. See also the survey of Susmita [23] for more applications.

Despite the large interest in ALL-PAIRS SHORTEST PATHS, there are only small improvements known since the well-known $\mathcal{O}(n^3)$-time algorithm by Floyd and Warshall [8, 26] from 1962: Chan [3] as well as Han and Takaoka [12] gave an algorithm running in $\mathcal{O}(n^3/\log^2 n)$ (omitting $poly(\log\log n)$ factors) and Williams [27] gave an randomized algorithm running in time $\mathcal{O}(n^3/2^{\Omega(\log n)^{1/2}})$. While there are no unconditional lower bounds known, it has been

conjectured that there is no truly subcubic algorithm for ALL-PAIRS SHORTEST PATHS, i.e., that no algorithm achieves time $\mathcal{O}(n^{3-\varepsilon})$ for any $\varepsilon > 0$. Using suitable subcubic reductions, this is tightly connected to the existence of subcubic algorithms for several network centrality measures, finding a directed triangle of negative total edge length, finding the second shortest simple path between two nodes in an edge-weighted graph, or checking if a given matrix defines a metric. This means that if one of those problems can be solved in truly subcubic time (i.e., can be solved in time $\mathcal{O}(n^{3-\varepsilon}) \cdot poly(\log M)$ for an $\varepsilon > 0$ and weights in $[-M, M]$ for weighted problems), then all of the problems admit algorithms with truly subcubic running time [28]. The situation is different for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS: While it is conjectured that there is no truly subcubic *combinatorial* algorithm, faster algorithms are known if fast matrix multiplication may be used. The currently fastest algorithm is due to Yuster [29] and runs in time $\mathcal{O}(n^{2.842})$. For sparse graphs there is an algorithm running in time $\mathcal{O}(nm + n^2 \log\log n)$ for directed graphs [20] and an algorithm for undirected graphs [21] with a running time of $\mathcal{O}(mn \log \alpha(m, n))$, where α is the inverse of the Ackermann's function.

Independently of whether one believes in conditional lower bounds and hypotheses, the fact remains that we do not know any truly subcubic algorithms for ALL-PAIRS SHORTEST PATHS nor truly subcubic, combinatorial algorithms for the vertex-weighted case. Besides heuristics or approximation algorithms, one possible solution for faster algorithms for at least *some* input graphs is to exploit *structure* in the input graph. In addition to measuring the complexity of a problem relative to the input size of a graph (number n of vertices and number m of edges), one may additionally consider some *parameter*, say k, that quantifies structure that may be exploited by an algorithm; i.e., we may study the *parameterized complexity* of the problem. This framework typically aims at NP-hard problems and a key goal is to obtain *fixed-parameter tractable* (FPT) algorithms that run in time $f(k)n^c$ for some constant c and some (usually exponential) function $f(k)$ of the parameter. Initiated by the work of Giannopoulou et al. [11], also efficient parameterized algorithms for tractable problems are considered (apart from many older results that predate even parameterized complexity). In this framework, also called "FPT in P", one is interested in running times $\mathcal{O}(k^\alpha n^\beta)$ when the best dependence on the input size alone is $\mathcal{O}(n^\gamma)$ with $\gamma > \beta$, which then results in a better running time for sufficiently small parameter k. Typically, the parameter k is at most n, thus, in the case of $\alpha + \beta = \gamma$, one already achieves truly better running times for $k \in o(n)$. We call such algorithms, which even for $k = n$ are not worse than the best unparameterized algorithm, *adaptive* algorithms.

Several recent publications dealt with efficient parameterized algorithms for different problems and parameters [9, 14, 4, 1, 13, 19], however, ALL-PAIRS SHORTEST PATHS got very little attention. Coudert et al. [4] considered the *clique-width* of a graph as a parameter for tractable problems related to cycle problems. Intuitively, clique-width captures the closeness of a graph to a cograph, with cographs being exactly the graphs of clique-width at most two. Alongside some positive results for TRIANGLE COUNTING or GIRTH, they proved a conditional lower bound for DIAMETER namely that there is no $\mathcal{O}(2^{o(\mathsf{cw})} \cdot n^{2-\varepsilon})$ time algorithm for any $\varepsilon > 0$. That is, even computing just the greatest length of any shortest path in an unweighted graph admits no such algorithm. A weaker parameter and an upper bound for clique-width is the modular-width of a graph, which is another parameter that has been previously studied regarding its use for efficient parameterized algorithms [4, 16].

Note that small clique-width or small modular-width does not imply the sparsity of the graph, e.g. cliques have clique-width and modular-width two. For parameters that do imply the sparsity of the graph (meaning that for parameter value k, the number of edges is

bounded by $\mathcal{O}(kn)$, where n denotes the number of vertices in the graph), the algorithm of Pettie and Ramachandran directly yield a running time of $\mathcal{O}(kn^2 + n^2 \log\log n)$, which is nearly optimal.

Our work. We study efficient parameterized algorithms for ALL-PAIRS SHORTEST PATHS for its vertex-weighted variant. We consider the structural parameters *clique-width* (cw) and *modular-width* (mw). As our main result, we present an $\mathcal{O}(\mathsf{cw}^2\, n^2)$-time algorithm for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS, yielding a truly subcubic algorithm for $\mathsf{cw} \in \mathcal{O}(n^{0.5-\varepsilon})$. This immediately allows to solve the DIAMETER problem in the same asymptotic time $\mathcal{O}(\mathsf{cw}^2\, n^2)$, even with vertex weights, and thereby nicely complements the lower bound ruling out $\mathcal{O}(2^{o(\mathsf{cw})} \cdot n^{2-\varepsilon})$ for any $\varepsilon > 0$ [4].

Further, we present a general framework to determine the running time for many algorithms that use modular-width and the related modular decomposition tree. We use this framework to prove an algorithm of time $\mathcal{O}(\mathsf{mw}^2\, n + n^2)$ for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS on graphs of modular-width at most mw. This algorithm is combinatorial, however, it can benefit from subcubic algorithms for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS that use fast matrix multiplication. For example, we achieve a running time of $\mathcal{O}(\mathsf{mw}^{1.842}\, n + n^2)$ by using an $\mathcal{O}(n^{2.842})$-time algorithm for the vertex-weighted case by Yuster [29] in each prime node; this algorithm uses fast matrix multiplication whereas all other algorithms (previous and new) are combinatorial.

Related Work. Following the work of Floyd and Warshall [8, 26], Fredman [10] achieved the first subcubic algorithm, running in time $\mathcal{O}(n^3 \log^{1/3} \log n / \log^{1/3} n)$. Chan [3] and Han and Takaoka [12] both achieved a running time of $\mathcal{O}(n^3/\log^2 n)$ (omitting $poly(\log\log n)$ factors). Recently, Williams [27] solved APSP in randomized time $\mathcal{O}(n^3/2^{\Omega(\log n)^{1/2}})$. For sparse graphs, Pettie and Ramachandran [21] get a running time of $\mathcal{O}(n^2\alpha(n,m) + mn)$. All these algorithms solve the standard edge-weighted case.

In the vertex-weighted case, the currently fastest algorithm by Yuster [29] runs in $\mathcal{O}(n^{2.842})$ and relies on fast matrix multiplication. Shapira et al. [22] considered some variants of APSP, namely the ALL-PAIRS BOTTLENECK PATHS, where one seeks the maximum bottleneck weight on a graph, and provided an algorithm of time $\mathcal{O}(n^{2.575})$ for vertex-weighted graphs. Czumaj and Lingas [6] analyzed the related problem of finding the minimum-weight triangle in vertex-weighted graphs and achieved a running time of $\mathcal{O}(n^\omega + n^{2+o(1)})$. All of these algorithms for vertex-weighted graphs exploit fast matrix multiplication. There is no truly subcubic *combinatorial* algorithm known for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS.

There are some subcubic algorithms known for APSP on special graph classes, such as uniform disk graphs with non-negative vertex weights, induced by point sets of bounded density within a unit square. Lingas and Sledneu [18] showed how to solve APSP on such graphs in time $\mathcal{O}(\sqrt{r}n^{2.75})$, where r is the radius of the disk around the vertices in a unit square. Bentert and Nichterlein [2] considered the related problem of computing the diameter of a graph, parameterized by several parameters.

Organization. Section 2 contains the preliminaries, in particular, the definition of clique-width. In Section 3, we present the algorithm for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS parameterized by the clique-width. Due to space restrictions, most of the proofs, the algorithm for modular-width as well as the running time framework can only be found in the full version of this paper [17]. We conclude in Section 4.

2 Preliminaries

We follow basic graph notations [7]. For a natural number $k \in \mathbb{N}$, define $[k] = \{1, \ldots, k\}$. All graphs are simple, i.e., without loops or multiple edges. In a graph $G = (V, E)$, a path $P = (v_1, v_2, \ldots, v_n)$ is a sequence of vertices $v_i \in V$ with $\{v_i, v_{i+1}\} \in E$ for $i \in [n-1]$. We define by $P_{[v_i, v_j]}$ the subpath of P starting in v_i and ending in v_j for $i, j \in [n]$ with $i < j$. The *length* of a path is the number of edges in it. In a vertex-weighted graph $G = (V, E)$ with weights $\omega \colon V \to \mathbb{R}_{\geq 0}$, the *weight* (also called *cost*) of a path P is defined as $\omega(P) = \sum_{i=1}^{n} \omega(v_i)$. Thus, every paths between two distinct vertices u and v has minimum weight $\omega(v) + \omega(u)$ and a path of length 0 from a vertex v to itself has always weight $\omega(v)$. For a graph $G = (V, E)$ and $u, v \in V$, we denote the minimum weight of all paths between u and v as $dist_G(u, v)$. For a set of vertices $X \subseteq V$ and a vertex $u \in V$ we define $dist_G(u, X) = \min_{v \in X} dist_G(u, v)$ and for two sets of vertices $X, Y \subseteq V$, we define $dist_G(X, Y) = \min_{u \in X, v \in Y}(u, v)$.

For two sets A and B we denote the disjoint union by $A \dot\cup B$ and we say that two sets A and B *overlap* if $A \cap B \neq \emptyset$, $A \setminus B \neq \emptyset$, and $B \setminus A \neq \emptyset$.

2.1 Clique-width and NLC-width

A *k-labeled* graph is a graph in which each vertex is assigned one out of k labels. Formally, a vertex-labeled graph G is a triple (V, E, lab) with V being the vertex set, E denotes the set of edges, and $lab : V \to [k]$ is a function that defines the label for each vertex. For a k-labeled graph $G = (V, E, lab)$ we denote by $unlab(G) = (V, E)$ the underlying unlabeled graph. Intuitively, a graph G has clique-width at most k, if it is the underlying graph of some k-labeled graph that can be constructed by using four operations: (1) Introducing a single labeled vertex, (2) redefining one label to another label, (3) taking the disjoint union of two already created k-labeled graphs, and (4) adding all edges between vertices of label i to vertices of label j for a pair (i, j) of labels.

▶ **Definition 1** (Clique-width, [5]). *Let $k \geq 2$. The class CW_k consists of all k-labeled graphs that can be constructed by the following operations:*

- *The nullary operation $\bullet_a$, that corresponds to a graph consisting of a single vertex with a label $a \in [k]$.*
- *Let $G = (V, E, lab) \in CW_k$ be a k-labeled graph, and let $a, b \in [k]$. Then*

$$\rho_{a,b}(G) = (V, E, lab') \qquad \text{with } lab'(v) = \begin{cases} lab(v) & , \text{ if } lab(v) \neq a \\ b & , \text{ if } lab(v) = a \end{cases}$$

  is in CW_k.
- *Let $G = (V_G, E_G, lab_G) \in CW_k$ and $H = (V_H, E_H, lab_H) \in CW_k$ be two k-labeled graphs in CW_k with $V_G \cap V_H = \emptyset$. Then the disjoint union, defined by*

$$G \oplus H = (V_G \dot\cup V_H, E_G \dot\cup E_H, lab') \qquad \text{with } lab'(v) = \begin{cases} lab_G(v) & , \text{ if } v \in V_G \\ lab_H(v) & , \text{ if } v \in V_H \end{cases}$$

  is in CW_k.
- *Let $G = (V, E, lab) \in CW_k$ be a k-labeled graph, and let $a, b \in [k]$ with $a \neq b$. Then*

$$\eta_{a,b}(G) = (V, E', lab) \qquad \text{with } E' = E \cup \{\{u, v\} \mid lab(u) = a, lab(v) = b\}$$

  is in CW_k.

The clique-width of a graph G, denoted by $cw(G)$, is the smallest $k \geq 2$ such that there is a labeled graph $G' \in CW_k$ with $unlab(G') = G$. The expression consisting of the operations defined in Definition 1 is called a (clique-width) k-expression. For a k-expression t, we denote with $val(t)$ the resulting labeled graph and by $tree(t)$ the so called k-expression tree of t, which is the canonical tree representation of t. Clique-width is a strict generalization of modular-width, which is defined in the appendix. In fact, the clique-width of a graph G is equal to the maximum clique-width of any quotient graph of a prime node in the modular decomposition tree of G. On the other hand, modular-width cannot be bounded by a function of clique-width.

Very similar to clique-width, one can define NLC-width, which was introduced by Wanke [25]. The main differences are that the join operation η and the disjoint union operation $\oplus$ are somewhat combined and consecutive relabel operations are compressed into one operation.

▶ **Definition 2** (NLC-width). *Let $k \geq 1$. The class NLC_k consists of all k-labeled graphs that can be constructed by the following operations:*

- *The nullary operation $\bullet_a$, that corresponds to a graph consisting of a single vertex with a label $a \in [k]$.*
- *Let $G = (V, E, lab) \in NLC_k$ and let $R\colon [k] \to [k]$. Then*

$$\circ_R(G) = (V, E, lab') \qquad \text{with } lab'(v) = R(lab(v))$$

is in NLC_k.

- *Let $G = (V_G, E_G, lab_G) \in NLC_k$ and $H = (V_H, E_H, lab_H) \in NLC_k$ be two k-labeled graphs in NLC_k. Let $S \subseteq [k]^2$. Then*

$$G \times_S H = (V_G \cup V_H, E', lab') \qquad \text{with } lab'(v) = \begin{cases} lab_G(v) & , \text{ if } v \in V_G \\ lab_H(v) & , \text{ if } v \in V_H \end{cases}$$
$$\text{and } E' = E_G \cup E_H \cup \{\{u, v\} \mid u \in V_G, v \in V_H, \text{ and } (lab_G(u), lab_H(v)) \in S\}$$

is in NLC_k.

The NLC-width of a graph G, denoted by $nlc(G)$, is the smallest $k \geq 2$ such that there is a labeled graph $G' \in NLC_k$ with $unlab(G') = G$. As for clique-width, the expression consisting of the operations defined in Definition 2 is called a (NLC-width) k-expression. For a k-expression t, we again denote with $val(t)$ the resulting labeled graph and by $tree(t) = T$ canonical tree representation of t, the so called k-expression tree of t. This means each leaf node of T is marked with $\bullet_a$ for some $a \in [k]$ and each internal node is either marked with $\circ_R$ for some $R\colon [k] \mapsto [k]$ or with $\times_S$ for some $S \subseteq [k]^2$, according to the operations defined in Definition 1 resp. Definition 2. For a node $x \in V(T)$ we denote by G^x the labeled graph defined by the k-expression represented by the subtree of T rooted in x and we define by $L_i^x = \{v \in V(G^x) \mid lab(v) = i\}$ the set of vertex in G^x with label $i \in [k]$. For a node $x \in V(T)$, we will use the shortcut $dist^x(u, v) := dist_{G^x}(u, v)$ to denote the distance between two vertices u and v in G^x.

The following lemma shows that we can safely focus on NLC k-expression trees, since the NLC-width and clique-width only differs by a factor of two at most.

▶ **Lemma 3** ([15]). *For any graph G it holds that $nlc(G) \leq cw(G) \leq 2 \cdot nlc(G)$.*

3 APSP parameterized by clique-width

Assuming SETH, one cannot solve DIAMETER (and thus, unweighted ALL-PAIRS SHORTEST PATHS) in time $2^{o(\mathsf{cw})} \cdot n^{2-\varepsilon}$ [4]. In this section, we show how to solve VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS in time $\mathcal{O}(\mathsf{cw}^2\, n^2)$.

▶ **Theorem 4.** *For every graph $G = (V, E)$, given together with a clique-width k-expression and vertex weights $\omega\colon V \to \mathbb{R}_{\geq 0}$, VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS can be solved in time $\mathcal{O}(k^2 n^2)$.*

For an input graph $G = (V, E)$, given together with a clique-width k-expression for some $k \geq 2$, we transform in a first step the clique-width k-expression to an NLC-width k-expression in linear time as described for example in [15]. For the rest of this section, by writing k-expression we always refer to an NLC-width k-expression instead of a clique-width k-expression. We interpret the (NLC-width) k-expression as a k-expression tree T, in which each node $v \in V(T)$ is marked with an operation of the k-expression that is applied to the children of v. Accordingly, T has exactly n leafs, each marked with an operation $\bullet_i$ for $i \in [k]$, and exactly $n-1$ nodes marked with an operation $\times_S$ for some $S \subseteq [k]^2$. For ease of presentation, we assume that there is exactly one node marked with an operation $\circ_R$ for some $R\colon [k] \to [k]$ in between any two nodes marked with $\times_S$ (using $R(i) = i$ when no actual relabeling is necessary), hence, the length of the k-expression is $\mathcal{O}(n)$. Note, that the length of a clique-width k-expression is $\mathcal{O}(n+m)$ in general. For a node $x \in V(T)$ we denote by G^x the labeled graph that is defined by the subexpression tree of T rooted in x.

The algorithm consists of three phases. In the first phase, we traverse T in a bottom-up manner: For each node $x \in V(T)$ we partition the vertex set into sets of same-labeled vertices and compute the shortest distance for each single vertex to (the closest vertex in) each label set. Additionally, we compute the distance between each pair of label sets, i.e., the shortest distance of two vertices of the respective sets. Note, however, that in the first phase we only consider for each each node $x \in V(T)$ the distances in the graph G^x. In the second phase, we perform a top-down traversal of the k-expression tree T and consider the whole graph G. Once we have computed the necessary values in phase one and two, we traverse T one last time and finally compute the shortest path distances between all pairs of vertices.

First Phase. For a node $x \in V(T)$, which corresponds to the k-labeled graph G^x, we define $L_i^x = \{v \in V(G^x) \mid lab(v) = i\}$ as the set of all vertices in G^x with label i. Note, that $unlab(G^x)$ is an induced subgraph of G for any $x \in V(T)$. We traverse T in a bottom-up manner and compute for each node $x \in V(T)$ and for all pairs $(i,j) \in [k]^2$ of labels the shortest distance between some vertex in L_i^x and some vertex in L_j^x. Additionally, we compute for any vertex $v \in V(G^x)$ and any label $i \in [k]$ the shortest distance from v to some vertex in L_i^x. To be precise, for a node $x \in V(T)$ we compute the following values:

$$c_{i,j}^x = dist^x(L_i^x, L_j^x) \qquad \text{for } i,j \in [k]$$
$$a_{v,i}^x = dist^x(v, L_i^x) \qquad \text{for } v \in V(G^x), i \in [k]$$

For nodes $x \in V(T)$ that are marked with $\times_S$ for some $S \subseteq [k]^2$ we need to compute some auxiliary values. Let y and z be the two children of x in T. This means that G^x consists of the disjoint union of G^y and G^z together with a full join between the vertex sets L_i^y and L_j^z for each $(i,j) \in S$. Thus, one can partition the vertex set of G^x into the $2k$ sets $\{L_1^y, \ldots, L_k^y, L_1^z, \ldots, L_k^z\} = \mathcal{L}^x$. For each pair $(A, B) \in \mathcal{L}^x \times \mathcal{L}^x$ of vertex sets, we compute the shortest distance between some vertex in A to some vertex of B. In addition, we compute

the shortest distance between A and B with the constraint that either the first edge, the last edge, or the first and the last edge of the shortest path is an edge of a newly inserted full join defined by S. This achieves the effect that we additionally compute the shortest distance from (1) *all* vertices of A to *some* vertex of B, (2) from *some* vertex of A to *all* vertices of B, and (3) from *all* vertices of A to *all* vertices of B. Doing this, one can e.g. combine a path that ends at *some* vertex of A with a path that can start at *any* vertex of A. In the following, we will describe how to compute the required values for each of the three different types of nodes in the k-expression tree T.

For the base case, let x be a leaf of the k-expression tree T. Thus, the node $x \in V(T)$ is marked with $\bullet_\ell$ for some $\ell \in [k]$. This means that G^x consists of a single vertex v with label ℓ. In this case the following holds:

$$c^x_{i,j} = \begin{cases} \omega(v) & \text{if } i = j = \ell \\ \infty & \text{otherwise} \end{cases} \qquad \text{for } i,j \in [k]$$

$$a^x_{v,i} = \begin{cases} \omega(v) & \text{if } i = \ell \\ \infty & \text{otherwise} \end{cases} \qquad \text{for } v \in V(G^x), i \in [k]$$

Now, let $x \in V(T)$ be an internal node of the k-expression tree T marked with $\circ_R$ for some $R\colon [k] \to [k]$. Let $y \in V(T)$ be the unique child of x in T. Since we traverse T in a bottom-up manner, we have already computed the values $a^y_{v,i}$ for all $v \in V(G^y)$ and $i \in [k]$ and the values $c^x_{i,j}$ for all $i,j \in [k]$. Note, that $unlab(G^x) = unlab(G^y)$, which, in particular, implies that distances between vertices are identical in both graphs (though distances between label sets may be not, as these sets may be different).

▶ **Lemma 5.** *Let $x \in V(T)$ be an internal node of a k-expression tree T marked with $\circ_R$ for some $R\colon [k] \to [k]$ and let y be the child of x in T. Then $c^x_{i,j} = \min_{i' \in R^{-1}(i), j' \in R^{-1}(j)} c^y_{i',j'}$ for all $i,j \in [k]$.*

Note, that the computation of all $c^x_{i,j}$ can be realized in time $\mathcal{O}(k^2)$ by updating for every $c^y_{i,j}$ the corresponding value $c^x_{R(i),R(j)}$. The values $a^x_{v,i}$ can be similarly computed from the values at the child node:

▶ **Lemma 6.** *Let $x \in V(T)$ be an internal node of a k-expression tree T marked with $\circ_R$ for some $R\colon [k] \to [k]$ and let y be the unique child of x in T. Then $a^x_{v,i} = \min_{j \in R^{-1}(i)} a^y_{v,j}$ for all $v \in V(G^x)$ and $i \in [k]$.*

The running time for computing the values $a^x_{v,i}$ is $\mathcal{O}(nk)$ since we need to consider each value $a^y_{v,i}$ for any $v \in V(G^y)$ and $i \in [k]$ exactly once.

Finally, let $x \in V(T)$ be an internal node of the k-expression tree T marked with $\times_S$ for some $S \subseteq [k]^2$. Denote by $y \in V(T)$ and $z \in V(T)$ the two children of x in T, meaning that G^x combines the two labeled graphs G^y and G^z by introducing for each $(i,j) \in S$ a full join between the vertices in L^y_i and those in L^z_j. Thus, $V(G^x) = V(G^y) \dot\cup V(G^z)$ and one can partition the vertices of G^x into the $2k$ vertex sets $\{L^y_1, \ldots, L^y_k, L^z_1, \ldots, L^z_k\}$ and $L^x_i = L^y_i \dot\cup L^z_i$. To compute the desired distances between the label sets $\{L^y_1, \ldots, L^y_k, L^z_1, \ldots, L^z_k\}$, we construct an edge-weighted directed graph H^x that represents all the distances between the label sets in a graph with only $4k$ vertices.

For each label set L^a_i of G^x with $i \in [k]$ and $a \in \{y,z\}$ we create two vertices v^a_i and u^a_i. Let $V^a = \{v^a_i \mid i \in [k]\}$ resp. $U^a = \{u^a_i \mid i \in [k]\}$ for $a \in \{y,z\}$. We add a directed full join from V^y to U^y resp. from V^z to U^z with weight equal to the length of a shortest path between the two corresponding label sets. Finally, we connect vertices in U^y with vertices in

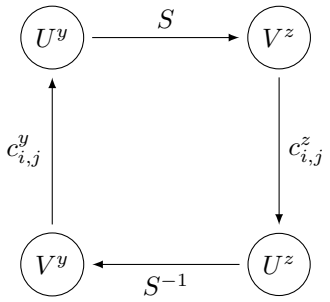


Figure 1 Construction of the auxiliary graph H^x. Each large node consists of k disjoint vertices corresponding to one of the k label sets in G^y resp. G^z. Between V^y and U^y there is a full join, each edge between the corresponding vertex of L^y_i and L^y_j is weighted by $c^y_{i,j}$, analogously for V^z and U^z. Vertices in U^y are only connected to those vertices in V^z for which the corresponding label sets are connected via S, analogously for U^z and V^y.

V^z, resp. vertices in U^z with vertices in V^y, if and only if the corresponding pair is contained in S, i.e., if there is a full join in G^x between the two corresponding label sets. See also Figure 1 for an illustration. Formally, we define the directed, edge-weighted graph H^x as follows.

▶ **Definition 7.** *Let $x \in V(T)$ be an internal node of a k-expression tree T marked with $\times_S$ for some $S \subseteq [k]^2$ and let $y \in V(T)$ and $z \in V(T)$ be the children of x. We define H^x as a directed, edge-weighted graph on $4k$ vertices created as follows:*

- *For each label set $L^a_i \in \{L^y_1, \ldots, L^y_k, L^z_1, \ldots, L^z_k\}$ we create two vertices v^a_i and u^a_i for $i \in [k]$ and $a \in \{y, z\}$.*
- *Add edges (v^y_i, u^y_j) with cost $c^y_{i,j}$ for all $i, j \in [k]$.*
- *Add edges (v^z_i, u^z_j) with cost $c^z_{i,j}$ for all $i, j \in [k]$.*
- *Add edges (u^y_i, v^z_j) with cost zero for all $(i, j) \in S$.*
- *Add edges (u^z_i, v^y_j) with cost zero for all $(j, i) \in S$.*

Note, that some edges may have cost ∞ as there is no path of the requested type exists. Next, we will see that H^x exhibits all the desired distances from G^x in a compact way.

▶ **Theorem 8.** *Let $x \in V(T)$ be an internal node of a k-expression tree T marked with $\times_S$ for some $S \subseteq [k]^2$ with children y and z. Let H^x be the graph as defined in Definition 7. Then the following holds:*

(1) *$dist_{H^x}(v^a_i, u^b_j) = dist^x(L^a_i, L^b_j)$ for all $a, b \in \{y, z\}$ and $i, j \in [k]$.*

(2) *$dist_{H^x}(u^a_i, u^b_j) = \min_{P \in \mathcal{P}} \omega(P) - \min_{v \in L^a_i} \omega(v)$ where $\mathcal{P}$ is the set of all paths in G^x starting in L^a_i, ending in L^b_j, and having the second vertex in $V(G^x) \setminus V(G^a)$.*

(3) *$dist_{H^x}(v^a_i, v^b_j) = \min_{P \in \mathcal{P}} \omega(P) - \min_{v \in L^b_j} \omega(v)$ where $\mathcal{P}$ is the set of all paths in G^x starting in L^a_i, ending in L^b_j, and having the penultimate vertex in $V(G^x) \setminus V(G^b)$.*

(4) *$dist_{H^x}(u^a_i, v^b_j) = \min_{P \in \mathcal{P}} \omega(P) - \min_{v \in L^a_i} \omega(v) - \min_{v \in L^b_j} \omega(v)$ where $\mathcal{P}$ is the set of all paths in G^x starting in L^a_i, ending in L^b_j, and having the second vertex in $V(G^x) \setminus V(G^a)$ and the penultimate vertex in $V(G^x) \setminus V(G^b)$.*

We prove Theorem 8 in two steps. We first prove that every path in H^x corresponds to some path in G^x. Later, we prove that also each optimal path between two label sets in G^x corresponds to some shortest path in H^x. We start with statement (1) of Theorem 8.

▶ **Lemma 9.** *Let $x \in V(T)$ be an internal node of the k-expression tree T marked with $\times_S$ for some $S \subseteq [k]^2$ with children y and z. Let H^x be the auxiliary graph as defined in Definition 7 and let P^* be an arbitrary v_i^y-u_j^z-path in H^x for some $i, j \in [k]$. Then there exists an L_i^y-L_j^z-path P in G^x with $\omega(P) = \omega_{H^x}(P^*)$.*

Proof. Due to the circular structure of H^x, each v_i^y-u_j^z-path in H^x will repeat the sequence $(v_p^y, u_q^y, v_r^z, u_s^z)$ for some $p, q, r, s \in [k]$ until reaching u_j^z at the end of a sequence. Thus, each v_i^y-u_j^z-path in H^x consists of 4ℓ vertices for $\ell \in \mathbb{N}$ and can be written as

$$P^* = (v_i^y = v_{p_1}^y, u_{q_1}^y, v_{r_1}^z, u_{s_1}^z, v_{p_2}^y, u_{q_2}^y, v_{r_2}^z, u_{s_2}^z, \ldots, v_{p_\ell}^y, u_{q_\ell}^y, v_{r_\ell}^z, u_{s_\ell}^z = u_j^z).$$

One can construct a path P in G^x from P^* as follows: For each edge $(v_{p_i}^y, u_{q_i}^y)$ in P^* of cost c_{p_i,q_i}^y pick a shortest path in G^y of total cost c_{p_i,q_i}^y and for each edge $(v_{r_i}^z, u_{s_i}^z)$ in P^* of cost c_{r_i,s_i}^y pick a shortest path in G^z of total cost c_{r_i,s_i}^z for each $i \in [\ell]$. Those paths always exist since c_{p_i,q_i}^y resp. c_{r_i,s_i}^y are defined as the cost of a shortest $L_{p_i}^y$-$L_{q_i}^y$-path in G^y, resp. as the cost of a shortest $L_{r_i}^z$-$L_{s_i}^z$ in G^z. Since each edge (u_{q_i}, v_{r_i}) only exists if and only if there is a full join between the sets $L_{q_i}^y$ and $L_{r_i}^z$, one can connect the last vertex of the path corresponding to the previous edge in P^* (that ends in some vertex in $L_{q_i}^y$) to the first vertex of the path corresponding to the following edge in P^* (that starts at some vertex in $L_{r_i}^z$). In the same manner one can argue that due to each edge $(u_{s_i}^z, v_{p_{i+1}}^y)$ one can connect the last vertex of the path corresponding to the edge $(v_{r_i}^z, u_{s_i}^z)$ with the first vertex of the path corresponding to the edge $(v_{p_{i+1}}^y, u_{q_{i+1}}^y)$. In both cases, the cost of the vertices is already accounted for in the resp. $c_{\cdot,\cdot}^{\cdot}$ value. Thus, each v_i^y-u_j^z-path in H^x corresponds to an L_i^y-L_j^z-path in G^x of same cost. ◀

Next, we generalize this argumentation to the following corollary:

▶ **Corollary 10.** *Let $x \in V(T)$ be an internal node of the k-expression tree T marked with $\times_S$ for some $S \subseteq [k]^2$ with children y and z. Let H^x be the auxiliary graph as defined in Definition 7. Then for every $i, j \in [k]$ and $a, b \in \{y, z\}$ the following holds:*

(1) *For any v_i^a-u_j^b-path P^* in H^x there exists an L_i^a-L_j^b-path P in G^x with $\omega_{H^x}(P^*) = \omega(P)$.*

(2) *For any u_i^a-u_j^b-path P^* in H^x there exists an L_i^a-L_j^b-path $P = (p_1, p_2, \ldots, p_\ell)$ in G^x with the property that $p_2 \in V(G^x) \setminus V(G^a)$ and $\omega_{H^x}(P^*) = \omega(P) - \omega(p_1)$.*

(3) *For any v_i^a-v_j^b-path P^* in H^x there exists an L_i^a-L_j^b-path $P = (p_1, \ldots, p_{\ell-1}, p_\ell)$ in G^x with the property that $p_{\ell-1} \in V(G^x) \setminus V(G^b)$ and $\omega_{H^x}(P^*) = \omega(P) - \omega(p_\ell)$.*

(4) *For any u_i^a-v_j^b-path P^* in H^x there exists an L_i^a-L_j^b-path $P = (p_1, p_2, \ldots, p_{\ell-1}, p_\ell)$ in G^x with the property that $p_2 \in V(G^x) \setminus V(G^a)$, $p_{\ell-1} \in V(G^x) \setminus V(G^b)$, and $\omega_{H^x}(P^*) = \omega(P) - \omega(p_1) - \omega(p_\ell)$.*

For any path in H^x that starts at some vertex u_i^a (resp. ends at some vertex v_j^b) one can find a corresponding path P in G^x with the property that the second vertex (resp. the penultimate vertex) is connected to *all* vertices of L_i^a (resp. L_j^b). Thus, one can extend any path that ends at *some* vertex in L_i^a by such a path (resp. one can prepend any path that starts in L_j^b by such a path). Hence, the cost of the first vertex (resp. last vertex) is neglected if the path starts in some vertex u_i^a or ends at some vertex v_j^b for $a, b \in \{x, y\}$. In general, every path that one can find in H^x corresponds to a path in G^x of essentially the same cost, possibly without the first or last vertex (which can be chosen as the minimum of the label set). This proves "$\leq$" in the equations of Theorem 8.

For the other direction, we will show that every optimal shortest path between two label sets in G^x is represented by a path in H^x.

▶ **Lemma 11.** *Let $x \in V(T)$ be an internal node of a k-expression tree T marked with $\times_S$ for some $S \subseteq [k]^2$ with children y and z. Let H^x be the auxiliary graph as defined in Definition 7. Let P be a shortest L_i^y-L_j^z-path in G^x for some $i, j \in [k]$. Then there exist a v_i^y-u_j^z-path P^* in H^x with $\omega_{H^x}(P^*) = \omega(P)$.*

Again, one can generalize the argumentation of Lemma 11 to the following corollary:

▶ **Corollary 12.** *Let $x \in V(T)$ be an internal node of the k-expression tree T marked with $\times_S$ for some $S \subseteq [k]^2$ with children y and z. Let H^x be the auxiliary graph as defined in Definition 7. Then for every $i, j \in [k]$ and $a, b \in \{y, z\}$ the following holds:*

(1) *For every shortest L_i^a-L_j^b-path P in G^x there exists a v_i^a-u_j^b-path P^* in H^x of cost $\omega_{H^x}(P^*) = \omega(P)$.*

(2) *For every shortest L_i^a-L_j^b-path P in G^x with the property that the second vertex is in $V(G^x) \setminus V(G^a)$ there exists a u_i^a-u_j^b-path P^* in H^x of cost $\omega_{H^x}(P^*) = \omega(P) - \min_{v \in L_i^a} \omega(v)$.*

(3) *For every shortest L_i^a-L_j^b-path P in G^x with the property that the penultimate vertex is in $V(G^x) \setminus V(G^b)$ there exists a v_i^a-v_j^b-path P^* in H^x of cost $\omega_{H^x}(P^*) = \omega(P) - \min_{v \in L_j^b} \omega(v)$.*

(4) *For every shortest L_i^a-L_j^b-path P in G^x with the property that the second vertex is in $V(G^x) \setminus V(G^a)$ and the penultimate vertex is in $V(G^x) \setminus V(G^b)$ there exists a u_i^a-v_j^b-path in H^x of cost $\omega_{H^x}(P^*) = \omega(P) - \min_{v \in L_i^a} \omega(v) - \min_{v \in L_j^b} \omega(v)$.*

Corollary 12 shows that every *shortest* L_i^a-L_j^b-path in G^x is represented in H^x for $i, j \in [k]$ and $a, b \in \{y, z\}$. Together with Corollary 10, this proves Theorem 8.

After the construction of the auxiliary graph H^x as defined in Definition 7, we compute and store the shortest distances for all pairs of vertices in H^x. With those values one can now compute the values $c_{i,j}^x$ and $a_{v,i}^x$ for $i, j \in [k]$ and $v \in V(G^x)$. Note that some of the values are only required in the second phase.

▶ **Corollary 13.** *Let $x \in V(T)$ be a node in the k-expression tree T marked with $\times_S$ for some $S \subseteq [k]^2$. For all $i, j \in [k]$ it holds that*

$$c_{i,j}^x = \min\left\{dist_{H^x}(v_i^y, u_j^y), dist_{H^x}(v_i^y, u_j^z), dist_{H^x}(v_i^z, u_j^y), dist_{H^x}(v_i^z, u_j^z)\right\}.$$

▶ **Corollary 14.** *Let $x \in V(T)$ be a node in the k-expression tree T marked with $\times_S$ for some $S \subseteq [k]^2$. Then for any $v \in V(G^y)$ and $i \in [k]$ it holds that $a_{v,i}^x = \min_{j \in [k], a \in \{y,z\}} \left\{a_{v,j}^y + dist_{H^x}(u_j^y, u_i^a)\right\}$.*

Second Phase. In this phase, we process the k-expression tree T in a top-down manner and use the local values that we have computed in the first phase to determine distances in the whole graph G.

Consider an internal node $x \in V(T)$ of the k-expression tree T marked with $\times_S$ for some $S \subseteq [k]^2$ and let y and z be the children of x in T. Let L_i^y resp. L_i^z denote the set of vertices with label i in G^y resp. G^z for $i \in [k]$. For an internal node x with children y and z we will compute for any vertex set L_i^y resp. L_i^z and every vertex $v \in V(G^x)$ the minimum cost of all paths in G that start in v and end in L_i^y resp. L_i^z with the property that the penultimate vertex is in $V(G) \setminus V(G^y)$, resp. in $V(G) \setminus V(G^z)$. Thus, the penultimate vertex is connected to *all* vertices of the vertex set L_i^y resp. L_i^z. It will therefore be convenient not to include the cost of the final vertex in these costs (cf. definition below). Note, that we consider the whole graph G in this step instead of just G^x.

Formally, for a node $x \in V(T)$ marked with $\times_S$ for some $S \subseteq [k]^2$ with children y and z we compute for every $v \in V(G^x)$ and $i \in [k]$ the following values:

- $d^x_{v,i,y} = \min_{P \in \mathcal{P}} \omega(P) - \min_{u \in L^y_i} \omega(u)$ where $\mathcal{P}$ is the set of all paths in G starting in v, ending in L^y_i, and having the penultimate vertex in $V(G) \setminus V(G^y)$.
- $d^x_{v,i,z} = \min_{P \in \mathcal{P}} \omega(P) - \min_{u \in L^z_i} \omega(u)$ where $\mathcal{P}$ is the set of all paths in G starting in v, ending in L^z_i, and having the penultimate vertex in $V(G) \setminus V(G^z)$.

For a node $x \in V(T)$ marked with $\circ_R$ for some $R\colon [k] \mapsto [k]$ and the child y, we only compute $d^x_{v,i,y}$. We start by computing those values for the root node. We can assume, w.l.o.g., that the root node has label $\times_S$ for some $S \subseteq [k]^2$.

▶ **Lemma 15.** *Let $r \in V(T)$ be the root node of the k-expression tree T marked with $\times_S$ for some $S \subseteq [k]^2$ and let y and z be the children of r. Let further H^r be the graph defined in Definition 7. Then, for any $v \in V(G^y)$ and for every $i \in [k]$ it holds that*

$$d^r_{v,i,y} = \min_{j \in [k]} \left\{a^y_{v,j} + dist_{H^r}(u^y_j, v^y_i)\right\} \quad \text{and} \quad d^r_{v,i,z} = \min_{j \in [k]} \left\{a^y_{v,j} + dist_{H^r}(u^y_j, v^z_i)\right\}.$$

Analogously, for any $v \in V(G^z)$ and for every $i \in [k]$ it holds that

$$d^r_{v,i,y} = \min_{j \in [k]} \left\{a^z_{v,j} + dist_{H^r}(u^z_j, v^y_i)\right\} \quad \text{and} \quad d^r_{v,i,z} = \min_{j \in [k]} \left\{a^z_{v,j} + dist_{H^r}(u^z_j, v^z_i)\right\}.$$

Next, we show how to propagate those values downwards in the k-expression tree, starting with a node marked with $\circ_R$ for some $R : [k] \mapsto [k]$.

▶ **Lemma 16.** *Let $x \in V(T)$ be an internal node of the k-expression tree T marked with $\circ_R$ for some $R\colon [k] \to [k]$. Let y be the unique child of x and w be the unique ancestor of x in T. Then $d^x_{v,i,y} = d^w_{v,R(i),x}$*

We now show the propagation for nodes x of T that are marked with $\times_S$. We start with one specific case and then conclude the general case as a corollary.

▶ **Lemma 17.** *Let $x \in V(T)$ be an internal node of the k-expression tree T that is marked with $\times_S$ for some $S \subseteq [k]^2$. Let y and z be the two children of x in T, let w be the unique ancestor of x in T, and let $v \in V(G^y)$ be arbitrary. Then $d^x_{v,i,y}$ is the minimum of the following three values:*

- $d^w_{v,i,x}$
- $\min_{j \in [k]} \left\{a^y_{v,j} + dist_{H^x}(u^y_j, v^y_i)\right\}$
- $\min_{j \in [k], c \in \{y,z\}} \left\{d^w_{v,j,x} + dist_{H^x}(v^c_j, v^y_i)\right\}$

Proof. After possibly adding nodes marked with $\circ_{id}$ to the k-expression tree, with id being the identity function, one can assume, that w is marked with $\circ_R$ for some $R\colon [k] \to [k]$ and that x is the only child of w.

Let $P = (p_1, \ldots, p_{n-1}, p_n)$ be a shortest v-L^x_i-path in G with penultimate vertex in $V(G) \setminus V(G^y)$, i.e., with $p_1 = v$, $p_n \in L^y_i$, and $p_{n-1} \in V(G) \setminus V(G^y)$; thus, $\omega(P) - \omega(p_n) = d^x_{v,i,y}$. We distinguish three cases:

Case 1: $p_{n-1} \in V(G) \setminus V(G^x)$. In this case, P is also a v-L^x_i-path with the property that the penultimate vertex is in $V(G) \setminus V(G^x)$; thus, $d^x_{v,i,y} \geq d^w_{v,i,x}$.

Case 2: $p_{n-1} \in V(G^z)$ and all vertices of P are in G^x. In this case, we can compute the value in the same way as done in Lemma 15 for the root node and get $d^x_{v,i,y} = \min_{j \in [k]} \{a^y_{v,j} + dist_{H^x}(u^y_j, v^y_i)\}$.

Case 3: $p_{n-1} \in V(G^z)$ and at least one vertex in P is in $V(G) \setminus V(G^x)$. Let p_ℓ be the last vertex of P that is in $V(G) \setminus V(G^x)$; clearly, $p_{\ell+1} \in V(G^x)$. We split the path P into the two subpaths $P_1 = (p_1, \ldots, p_\ell)$ and $P_2 = (p_{\ell+1}, \ldots, p_n)$. Let $j \in [k]$ such that

$p_{\ell+1} \in L_j^x$. Since p_ℓ is connected to $p_{\ell+1}$, the vertex p_ℓ is connected to every vertex in L_j^x. We extend P_1 by $p' = \arg\min_{u \in L_j^x} \omega(u)$ and denote the resulting path by P_1'. Now it holds by definition that $\omega(P_1') - \omega(p') \geq d^w_{v,j,x}$, as the penultimate vertex p_ℓ of P_1' is in $V \setminus V(G^x)$. Let further $c \in \{y, z\}$ such that $p_{\ell+1} \in L_j^c$, noting that it does not change its label at x. Then $\omega(P_2) - \omega(p_n) \geq dist_{H^x}(v_j^c, v_i^y)$ by Theorem 8, as P_2 is a path in G^x. Note, that $\omega(P) = \omega(P_1') + \omega(P_2) - \omega(p')$. Thus, in this case it holds that

$$\begin{aligned} d^x_{v,i,y} &= \omega(P) - \min_{u \in L_i^y} \omega(u) \\ &= \omega(P_1') - \omega(p') + \omega(P_2) - \min_{u \in L_i^y} \omega(u) \\ &\geq d^w_{v,j,x} + dist_{H^x}(v_j^c, v_i^y) + \omega(p_n) - \min_{u \in L_i^y} \omega(u) \\ &\geq d^w_{v,j,x} + dist_{H^x}(v_j^c, v_i^y) \\ &\geq \min_{j \in [k], c \in \{y,z\}} \left\{ d^w_{v,j,x} + dist_{H^x}(v_j^c, v_i^y) \right\} \end{aligned}$$

We have seen in the case analysis above that in each case $d^x_{v,i,y}$ is at least the value considered in the case; in particular, it is at least equal to their collective minimum value. On the other hand, for each case there is a path P fulfilling the definition of $d^x_{v,i,y}$ such that $\omega(P) - \min_{u \in L_i^y} \omega(u)$ equals the value of the considered case. Thus, $d^x_{v,i,y}$ is also at most equal to the minimum taken over all three cases. This completes the proof. ◀

▶ **Corollary 18.** *Let $x \in V(T)$ be an internal node of the k-expression tree T marked with $\times_S$ for some $S \subseteq [k]^2$. Let y and z be the unique children of x in T, w be the unique ancestor of x in T, and let $\alpha, \beta \in \{y, z\}$ be arbitrary. Then, for $v \in V(G^\alpha)$ the value $d^x_{v,i,\beta}$ is the minimum of the following three values:*

- $d^w_{v,i,x}$
- $\min_{j \in [k]} \left\{ a^\alpha_{v,j} + dist_{H^x}(u_j^\alpha, v_i^\beta) \right\}$
- $\min_{j \in [k], c \in \{y,z\}} \left\{ d^w_{v,j,x} + dist_{H^x}(v_j^c, v_i^\beta) \right\}$

Third Phase. In the third phase, we traverse the k-expression tree T one final time; the ordering is immaterial. We go over all nodes x with label $\times_S$ for some $S \subseteq [k]^2$ and compute for each pair of vertices (u, v) with $u \in V(G^y)$ and $v \in V(G^z)$ the shortest u-v-path in G, where y and z are the two children of x in T. Since the leaves of T correspond one-to-one to single-vertex graphs, one for each vertex of G, this procedure will consider every pair of vertices in G at some node $x \in V(T)$.

▶ **Lemma 19.** *Let $x \in V(T)$ be an internal node of the k-expression tree T marked with $\times_S$ for some $S \subseteq [k]^2$. Let y and z be the two children of x and let $u \in V(G^y)$ and $v \in V(G^z)$. Then $dist_G(u, v) = \min_{i \in [k]} \left\{ d^x_{u,i,z} + a^z_{v,i} \right\}$.*

Running time. First, we need to transform the clique-width k-expression into a NLC-width k-expression tree T, which can be done in linear time $\mathcal{O}(n + m)$ [15].

In the first traversal, we compute for every node $x \in V(T)$ the values $a^x_{v,i}$ for $v \in V(G^x)$ and $i \in [k]$. Thus, we compute at most $n \cdot k$ values, each in time $\mathcal{O}(k)$, which results in a running time of $\mathcal{O}(nk^2)$ per node of T. In the case of a node x with label $\times_S$ for some $S \subseteq [k]^2$ we first compute the auxiliary graph H^x in time $\mathcal{O}(|V(H)| + |E(H)|) = \mathcal{O}(k^2)$ and solve (edge-weighted) ALL-PAIRS SHORTEST PATHS on H^x in time $\mathcal{O}(k^3)$. After this, by using Corollary 13 resp. Corollary 14, we compute each $c^x_{i,j}$ in constant time resp. each $a^x_{v,i}$ in time $\mathcal{O}(k)$ resulting in a running time per node $x \in V(T)$ of $\mathcal{O}(k^3 + k^2 \cdot n)$.

In the second phase we perform a top-down traversal of T to compute the for each node x the values $d^x_{v,i,y}$ and $d^x_{v,i,z}$ for all $v \in G^x$ and $i \in [k]$. Again, we compute at most $n \cdot k$ values, each in time $\mathcal{O}(k)$, which results in a running time of $\mathcal{O}(nk^2)$ per node of T. Since there are $\mathcal{O}(n)$ nodes in the k-expression tree T, the total running time for Phase One and Phase Two is $\mathcal{O}(nk^3 + n^2k^2) = \mathcal{O}(n^2k^2)$.

In the last phase, we consider each pair (u, v) of vertices exactly once and compute each pairwise distance in time $\mathcal{O}(k)$. Thus, running time for the last phase is $\mathcal{O}(n^2k)$. In total, we obtain the claimed bound of $\mathcal{O}(k^2n^2)$.

4 Conclusion

We started the study of VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS in the FPT in P framework and obtained efficient parameterized algorithms with respect to clique-width and modular-width. The algorithm parameterized by modular-width is adaptive, i.e., even if the parameter reaches its upper bound of n, the algorithm is not worse than the best unparameterized algorithm, and even for $k \in \mathcal{O}(n^{1-\varepsilon})$ for any $\varepsilon > 0$, it outperforms the best unparameterized algorithm. The algorithm parameterized by the stronger parameter clique-width is truly subcubic if $\mathsf{cw} \in \mathcal{O}(n^{0.5-\varepsilon})$ for any $\varepsilon > 0$. It also permits us to solve DIAMETER in the same time $\mathcal{O}(\mathsf{cw}^2\, n^2)$, complementing the lower bound ruling out $\mathcal{O}(2^{o(\mathsf{cw})} \cdot n^{2-\varepsilon})$ for any $\varepsilon > 0$, due to Coudert et al. [4]. The algorithms only apply to the vertex-weighted case. Note also that the algorithm relative to clique-width assume to be given a suitable expression or decomposition, whereas the modular decomposition of a graph, and hence its modular-width, can be computed in linear time [24].

As mentioned in [16], considering edge-weighted graphs with (low) clique-width resp. low modular-width is hopeless, as one could modify an arbitrary input graph by adding all the missing edges with sufficiently large weights. Clearly, the shortest path lengths do not change, but the resulting graph is a clique and has constant clique-width and modular-width.

Apart from considering other parameters, one interesting open question is whether there is an *adaptive* algorithm for ALL-PAIRS SHORTEST PATHS parameterized by clique-width, e.g., can the running time be reduced to $\mathcal{O}(\mathsf{cw}\, n^2)$? This seems quite challenging, since even computing some variant of ALL-PAIRS SHORTEST PATHS for each node in the expression tree (on a graph with cw many nodes) results in a non-adaptive running time.

References

1. Matthias Bentert, Till Fluschnik, André Nichterlein, and Rolf Niedermeier. Parameterized aspects of triangle enumeration. *J. Comput. Syst. Sci.*, 103:61–77, 2019. doi:10.1016/j.jcss.2019.02.004.
2. Matthias Bentert and André Nichterlein. Parameterized complexity of diameter. In Pinar Heggernes, editor, *Algorithms and Complexity - 11th International Conference, CIAC 2019, Rome, Italy, May 27-29, 2019, Proceedings*, volume 11485 of *Lecture Notes in Computer Science*, pages 50–61. Springer, 2019. doi:10.1007/978-3-030-17402-6_5.
3. Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM J. Comput.*, 39(5):2075–2089, 2010. doi:10.1137/08071990X.
4. David Coudert, Guillaume Ducoffe, and Alexandru Popa. Fully polynomial FPT algorithms for some classes of bounded clique-width graphs. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2765–2784. SIAM, 2018. doi:10.1137/1.9781611975031.176.

5 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

6 Artur Czumaj and Andrzej Lingas. Finding a heaviest vertex-weighted triangle is not harder than matrix multiplication. *SIAM J. Comput.*, 39(2):431–444, 2009. doi:10.1137/070695149.

7 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

8 Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

9 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Michal Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Trans. Algorithms*, 14(3):34:1–34:45, 2018. doi:10.1145/3186898.

10 Michael L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.*, 5(1):83–89, 1976. doi:10.1137/0205006.

11 Archontia C. Giannopoulou, George B. Mertzios, and Rolf Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *CoRR*, abs/1506.01652, 2015. arXiv:1506.01652.

12 Yijie Han and Tadao Takaoka. An $\mathcal{O}(n^3 \log\log n/\log^2 n)$ time algorithm for all pairs shortest paths. *J. Discrete Algorithms*, 38-41:9–19, 2016. doi:10.1016/j.jda.2016.09.001.

13 Thore Husfeldt. Computing graph distances parameterized by treewidth and diameter. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPIcs*, pages 16:1–16:11. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.16.

14 Yoichi Iwata, Tomoaki Ogasawara, and Naoto Ohsaka. On the power of tree-depth for fully polynomial FPT algorithms. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 41:1–41:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.41.

15 Öjvind Johansson. Clique-decomposition, NLC-decomposition, and modular decomposition-relationships and results for random graphs. In *Congr. Numer.* Citeseer, 1998.

16 Stefan Kratsch and Florian Nelles. Efficient and adaptive parameterized algorithms on modular decompositions. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPIcs*, pages 55:1–55:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.55.

17 Stefan Kratsch and Florian Nelles. Efficient parameterized algorithms for computing all-pairs shortest paths. *arXiv e-prints*, page arXiv:2001.04908, January 2020. arXiv:2001.04908.

18 Andrzej Lingas and Dzmitry Sledneu. A combinatorial algorithm for all-pairs shortest paths in directed vertex-weighted graphs with applications to disc graphs. In Mária Bieliková, Gerhard Friedrich, Georg Gottlob, Stefan Katzenbeisser, and György Turán, editors, *SOFSEM 2012: Theory and Practice of Computer Science - 38th Conference on Current Trends in Theory and Practice of Computer Science, Špindlerův Mlýn, Czech Republic, January 21-27, 2012. Proceedings*, volume 7147 of *Lecture Notes in Computer Science*, pages 373–384. Springer, 2012. doi:10.1007/978-3-642-27660-6_31.

19 George B. Mertzios, André Nichterlein, and Rolf Niedermeier. Fine-grained algorithm design for matching. *CoRR*, abs/1609.08879, 2016. arXiv:1609.08879.

20 Seth Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theoretical Computer Science*, 312(1):47–74, 2004.

21 Seth Pettie and Vijaya Ramachandran. A shortest path algorithm for real-weighted undirected graphs. *SIAM Journal on Computing*, 34(6):1398–1431, 2005.

22 Asaf Shapira, Raphael Yuster, and Uri Zwick. All-pairs bottleneck paths in vertex weighted graphs. *Algorithmica*, 59(4):621–633, 2011. doi:10.1007/s00453-009-9328-x.

23 Susmita Susmita and Manish Pandey. Algorithms of all pair shortest path problem. *International Journal of Computer Applications*, 120(15):1–6, 2015.

24 Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 634–645. Springer, 2008. doi:10.1007/978-3-540-70575-8_52.

25 Egon Wanke. k-NLC graphs and polynomial algorithms. *Discrete Applied Mathematics*, 54(2-3):251–266, 1994. doi:10.1016/0166-218X(94)90026-4.

26 Stephen Warshall. A theorem on boolean matrices. In *Journal of the ACM*. Citeseer, 1962.

27 R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524.

28 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. doi:10.1145/3186893.

29 Raphael Yuster. Efficient algorithms on sets of permutations, dominance, and real-weighted APSP. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 950–957. SIAM, 2009. doi:10.1137/1.9781611973068.

Relational Width of First-Order Expansions of Homogeneous Graphs with Bounded Strict Width

Michał Wrona
Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Poland
http://www.tcs.uj.edu.pl/wrona
wrona@tcs.uj.edu.pl

Abstract

Solving the algebraic dichotomy conjecture for constraint satisfaction problems over structures first-order definable in countably infinite finitely bounded homogeneous structures requires understanding the applicability of local-consistency methods in this setting. We study the amount of consistency (measured by relational width) needed to solve CSP($\mathbb{A}$) for first-order expansions $\mathbb{A}$ of countably infinite homogeneous graphs $\mathcal{H} := (A; E)$, which happen all to be finitely bounded. We study our problem for structures $\mathbb{A}$ that additionally have bounded strict width, i.e., for which establishing local consistency of an instance of CSP($\mathbb{A}$) not only decides if there is a solution but also ensures that every solution may be obtained from a locally consistent instance by greedily assigning values to variables, without backtracking.

Our main result is that the structures $\mathbb{A}$ under consideration have relational width exactly $(2, \mathbb{L}_{\mathcal{H}})$ where $\mathbb{L}_{\mathcal{H}}$ is the maximal size of a forbidden subgraph of $\mathcal{H}$, but not smaller than 3. It beats the upper bound: $(2m, 3m)$ where $m = \max(\text{arity}(\mathbb{A}) + 1, \mathbb{L}, 3)$ and arity($\mathbb{A}$) is the largest arity of a relation in $\mathbb{A}$, which follows from a sufficient condition implying bounded relational width given in [10]. Since $\mathbb{L}_{\mathcal{H}}$ may be arbitrarily large, our result contrasts the collapse of the relational bounded width hierarchy for finite structures $\mathbb{A}$, whose relational width, if finite, is always at most $(2, 3)$.

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

Keywords and phrases Constraint Satisfaction, Homogeneous Graphs, Bounded Width, Strict Width, Relational Width, Computational Complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.39

Related Version A full version of the paper is available at https://arxiv.org/abs/2001.06676.

Funding The research was partially supported by NCN grant number 2014/14/A/ST6/00138.

1 Introduction

The constraint satisfaction problem (CSP) is one of the most important problems in theoretical and applied computer science and at the same time it is a general framework in which many other computational problems may be formalized. Given a number of constraints imposed on variables one asks if there is a global solution, i.e., a function assigning values to variables so that all the constraints are simultaneously satisfied. Boolean satisfiability and graph colouring are among the most prominent examples of NP-hard problems that can be formalized as CSPs and hence the CSP is NP-hard in general. Thus, one considers the problem CSP($\mathbb{A}$) parametrized by a relational structure (called also a constraint language, a language or a template) $\mathbb{A}$. (In this paper, $\mathbb{A}$ is always over a finite signature). A longstanding open problem in this area was to verify the Feder-Vardi [20] conjecture which states that for every finite $\mathbb{A}$ the problem CSP($\mathbb{A}$) is either in P or it is NP-complete. After over thirty years of work and a number of important partial results this so-called Dichotomy Conjecture was confirmed independently in [26] and [16]. In both cases the proof was carried out in the

so-called universal-algebraic approach to the complexity of CSPs [22, 17]. The approach not only provided appropriate tools but also suggested the delineation. This so-called algebraic dichotomy conjecture [17] saying that CSP($\mathbb{A}$) is hard under the condition that the algebra corresponding to $\mathbb{A}$ lacks interesting operations also has been confirmed in both proofs.

The universal-algebraic approach to finite-domain constraint satisfaction problems has been generalized to capture the computational complexity in many other similar settings. In particular, the complexity of CSP($\mathbb{A}$) depends on the algebra corresponding to $\mathbb{A}$ when $\mathbb{A}$ is ω-categorical [11], i.e., all countable models of the first-order theory of $\mathbb{A}$ are isomorphic. In particular, all structures first-order definable in (reducts of) (countably infinite) homogeneous structures over finite signatures are ω-categorical structure. (A structure is homogeneous if every isomorphism between its finite substructures may be extended to an automorphism of a structure.) Considering these infinite structures significantly broadens the class of problems that may be captured within the CSP framework. In particular, the order over rational numbers $(\mathbb{Q}, <)$, which is homogeneous, gives rise to CSP($\mathbb{Q}; <$) that can be seen as the digraph acyclicity problem. The latter cannot be expressed as the CSP over a finite template. Furthermore a number of problems of interest in qualitative reasoning may be captured by CSP($\mathbb{A}$) where $\mathbb{A}$ is a reduct of a homogeneous structure $\mathbb{B}$. It concerns constraint satisfaction problems in formalisms such as Allen's interval algebra or RCC-5, see [6] for a survey. Many of the homogeneous structures $\mathbb{B}$ of interest are finitely bounded, i.e., there exists a finite unique minimal set $\mathcal{F}_{\mathbb{B}}$ of finite structures over the signature of $\mathbb{B}$ such that a finite structure Δ embeds into $\mathbb{B}$ if and only if none of the structures in $\mathcal{F}_{\mathbb{B}}$ embeds into Δ. A dichotomy for algebras corresponding to reducts of countably infinite finitely bounded homogeneous structures was proved in [3]. As in the finite case, it suggests the delineation between polynomial-time solvable and NP-hard CSPs. Although the complexity dichotomy is still far from being obtained, the algebraic dichotomy conjecture for reducts of finitely bounded homogeneous structures is known to hold in the number of cases including the reducts of $(\mathbb{N}, =)$ [7], $(\mathbb{Q}, <)$ [8], the random partial order [23] or a countably infinite homogeneous graph [13, 14, 9].

Theoretical research on CSPs is focused not only on providing classifications of computational complexity but also on settling the limits of applicability of widely known algorithms or algorithmic techniques such as establishing local consistency. This method is used not only for finite CSP but is also considered to be the most important (if not the only) algorithmic technique for qualitative CSPs [25]. The algebraic characterization of finite structures $\mathbb{A}$ with bounded width [2], i.e., for which CSP($\mathbb{A}$) can be solved by establishing local consistency, is considered to be an important step towards solving the Feder-Vardi conjecture. Thus, in order to understand the complexity of CSPs for reducts $\mathbb{A}$ of finitely bounded homogeneous structures, we need to characterize $\mathbb{A}$ with bounded width and to understand how different notions of consistency relate to each other for templates under consideration. The focus of this paper is on the latter.

The amount of consistency needed to solve CSP($\mathbb{A}$) for $\mathbb{A}$ with bounded width is measured here [1] and here [15] by relational width. The relational width of $\mathbb{A}$ is a pair of numbers (k, l) with $k \leq l$ (for the exact definition we refer the reader to Section 4). The following question was of interest for finite structures.

▶ **Question 1.** *What is the exact relational width of $\mathbb{A}$ with bounded width?*

Question 1 for finite $\mathbb{A}$ was completely answered in [1] where it was proved that $\mathbb{A}$ with bounded width has always either relational width $(1, 1)$ or $(2, 3)$, see [15] for another proof. Both proofs rely, however, on the algebraic characterization of structures $\mathbb{A}$ with bounded width. Although the notion of bounded width has been generalized to ω-categorical structures [5], according to our knowledge, no algebraic characterization of bounded width for

such structures is within sight. Nevertheless the algebraic characterization of strict bounded width has been quite easily lifted from finite [20] to infinite domains [5]. (Again, for a detailed definition we refer the reader to Section 4.) A reduct of a finitely bounded homogeneous structure has bounded strict width if and only if it is preserved by so-called oligopotent quasi near-unanimity operation. This algebraic characterization gives us a hope to answer the following question analogous to Question 1.

▶ **Question 2.** *What is the exact relational width of reducts $\mathbb{A}$ of finitely bounded homogeneous structures with bounded strict width?*

In this paper we answer Question 2 for first-order expansions of countably infinite homogeneous graphs. We believe that our method may be used to provide the general answer in the near future. We note that the answer to Question 2 would be not only a nice theoretical result but should be also of particular interest for structures that give rise to constraint satisfaction problems in qualitative reasoning. In this context strict width is called *local-to-global consistency* and has been widely studied see, e.g., [19].

1.1 Our results

In contrast to all homogeneous structures, all countably infinite homogeneous graphs are well understood and have been classified in [24]. It happens that every such graph $\mathcal{H}$ is also finitely bounded, i.e., in each case there exists a finite unique minimal set of finite graphs $\mathcal{F}_{\mathcal{H}}$ such that a finite graph G embeds into $\mathcal{H}$ if and only if none of the graphs in $\mathcal{F}_{\mathcal{H}}$ embeds into G. We will write $\mathbb{L}_{\mathcal{H}}$ for the maximum of the number 3 and the size of the largest finite structure in $\mathbb{L}_{\mathcal{H}}$. Perhaps the best known example of a homogeneous graph is the random graph that is determined up to isomorphism by the two properties of being homogeneous and universal (i.e., it contains all countable graphs as induced subgraphs). Equivalently, the random graph is a unique countably infinite graph which has this extension property: for all disjoint finite subsets U, U' of the domain there exists an element v such that v is adjacent to all members of U and to none in U'. In this case the finite set of bounds consists of a single directed edge and a loop, and hence $\mathbb{L}_G$ for the random graph G is 3. Furthermore, the family of homogeneous graphs contains universal countable k-clique free graphs H_k with $k \geq 3$, called also Henson graphs, in which case $\mathcal{F}_{H_k}$ contains also a k-clique, and hence $\mathbb{L}_{H_k}$ is k or the graphs C_n^s that are disjoint sums of n cliques of size s where $1 \leq n, s \leq \omega$ and either n or s equals ω. Observe that $\mathcal{F}_{C_n^s}$ contains a graph on three vertices with two edges and one non-edge as well as a null graph over $n+1$ vertices in case n is finite or a $(s+1)$-clique in the case where s is finite. Thus, $\mathbb{L}_{C_n^s}$ is either $3, n+1$ or $s+1$. All remaining homogeneous graphs are the complements of graphs H_k or C_n^s. In this paper we prove the following.

▶ **Main Result.** *Let $\mathbb{A}$ be a first-order expansion of a countably infinite homogeneous graph $\mathcal{H}$ such that $\mathbb{A}$ has bounded strict width. Then $\mathbb{A}$ has relational width $(2, \mathbb{L}_{\mathcal{H}})$.*

In fact, we obtain a more general result. Some sufficient conditions implying that a first-order expansion of a homogeneous graph $\mathcal{H}$ has relational width $(2, \mathbb{L}_{\mathcal{H}})$ are given in Section 5. In particular, the conditions cover all languages under consideration preserved by binary canonical operations considered in [13, 14, 9] where an analysis of algebras corresponding to reducts of homogeneous graphs and the computational dichotomy is provided. Our result: relational width $(2, \mathbb{L}_{\mathcal{H}})$ beats the upper bound $(2m, 3m)$, where $m = \max(\text{arity}(\mathbb{A}) + 1, \mathbb{L}, 3)$ and $\text{arity}(\mathbb{A})$ is the largest arity of a relation in $\mathbb{A}$, that can be easily obtained from the proof of Theorem 4.10 in [10].

We believe that measuring relational width of structures with bounded width is interesting in its own rights. Nevertheless, our research has complexity consequences. As in the finite case, it was proved in [5] that CSP($\mathbb{A}$) for an ω-categorical $\mathbb{A}$ with strict width k may be solved by establishing $(k, k+1)$-consistency and hence in time $O(n^{k+1})$ where n is the number of variables in an instance. Our main result implies that such CSP($\mathbb{A}$) for a first-order expansion $\mathbb{A}$ of a homogeneous graph $\mathcal{H}$ may be solved by establishing $(2, \mathbb{L}_{\mathcal{H}})$-minimality, and hence in time $O(n^m)$ where $m = \max(\mathbb{L}_{\mathcal{H}}, \text{arity}(\mathbb{A}))$.

1.2 Outline of the paper

We start with general preliminaries in Section 2. Then we review canonical operations providing tractability for reducts of homogeneous graphs, Section 3. Bounded (relational) width, strict width and other notions related to local consistency are provided in Section 4. There we also give a number of examples explaining the applicability of our main result. The proof of the main result is divided into Section 5 and Section 6. In the former one, we give a number of sufficient conditions implying relational width $(2, \mathbb{L}_{\mathcal{H}})$, while in the latter one we show that the sufficient conditions are satisfied whenever a first-order expansion of $\mathcal{H}$ has bounded strict width. In Section 5 we additionally show that the sufficient condition are also satisfied by first-order expansions of homogeneous graphs preserved by the studied binary canonical operations. As a consequence, we obtain that all tractable (whose CSP is solvable in polynomial time) reducts of $\mathcal{H}$ where $\mathcal{H}$ is $C_1^\omega, C_\omega^1, C_\omega^\omega$ or H_k with $k \geq 3$ have bounded relational width $(2, \mathbb{L}_{\mathcal{H}})$ and hence can be solved by establishing $(2, \mathbb{L}_{\mathcal{H}})$-minimality.

2 Preliminaries

We write $t = (t[1], ..., t[n])$ for a tuple of elements and $[n]$ to denote the set $\{1, \dots, n\}$.

2.1 Relations, languages and formulas

In this paper we consider first-order expansions $\mathbb{A} := (A; E, R_1, \dots, R_k)$ over a finite signature τ of homogeneous graphs, called also (constraint) languages or templates, where all $R_1, \dots, R_k$ have a first-order definition in $(A; E)$. We assume that $\mathbb{A}$ constains $=$ and N whenever N is pp-definable in $\mathbb{A}$. Relations E and N refer always to a homogeneous graph $\mathcal{H}$ known from the context. For the sake of presentation we usually do not distinguish between a relation symbol R in the signature of $\mathbb{A}$ and the relation $R^{\mathbb{A}}$ and use the former symbol for both. We often write $O, O_1, O_2, \dots$ for elements of $\{E, N, =\}$ and $\underline{\underline{\text{E}}}, \underline{\underline{\text{N}}}, \underline{\underline{\text{O}}}, \underline{\underline{\text{O}_1}}, \underline{\underline{\text{O}_2}}$ to denote relations $(E \cup =), (N \cup =), (O \cup =), (O_1 \cup =), (O_2 \cup =)$, respectively.

For a structure $\mathbb{A}$ over domain A and a tuple $t \in A^k$, the orbit of t in $\mathbb{A}$ is the relation $\{(\alpha(t[1]), \dots, \alpha(t[k])) \mid \alpha \in \text{Aut}(\mathbb{A})\}$ where $\text{Aut}(\mathbb{A})$ is the set of automorphisms of $\mathbb{A}$. In particular, E, N and $=$ are orbits of pairs, called also orbitals. We would like to note that all structures considered in this paper are ω-categorical. By a theorem proved independently by Ryll-Nardzewski, Engeler and Svenonius, a structure $\mathbb{A}$ is ω-categorical if and only if its automorphism group is oligomorphic, i.e., for every n the number of orbits of n-tuples is finite. See [21] for a textbook on model theory.

A primitive-positive (pp-)formula is a first-order formula built exclusively out of existential quantifiers $\exists$, conjunction $\wedge$ and atomic formulas $R(x_1, \dots, x_k)$ where R is a k-ary relation symbol and $x_1, \dots, x_k$ are variables, not necessarily pairwise different.

2.2 The universal-algebraic approach

We say that an operation $f : A^n \to A$ is a polymorphism of an m-ary relation R iff for any m-tuples $t_1, \ldots, t_n \in R$, it holds that the tuple $(f(t_1[1], \ldots, t_n[1]), \ldots, f(t_1[m], \ldots, t_n[m]))$ is also in R. We write $f(t_1, \ldots, t_n)$ as a shorthand for $(f(t_1[1], \ldots, t_n[1]), \ldots, f(t_1[m], \ldots, t_n[m]))$. An operation f is a polymorphism of $\mathbb{A}$ if it is a polymorphism of every relation in $\mathbb{A}$. If $f : A^n \to A$ is a polymorphism of $\mathbb{A}, R$, we say that f preserves $\mathbb{A}, R$, otherwise that f violates $\mathbb{A}, R$. A set of polymorphisms of an ω-categorical structure $\mathbb{A}$ forms an algebraic object called an oligomorphic locally closed clone [4], which in particular contains an oligomorphic permutation group [18].

▶ **Theorem 1** ([11]). *Let $\mathbb{A}$ be a countable ω-categorical structure. Then R is preserved by the polymorphisms of $\mathbb{A}$ if and only if it has a primitive-positive definition in $\mathbb{A}$, i.e., a definition via a primitive-positive formula.*

We say that a set of operations F generates a set of operations G if every $g \in G$ is in the smallest locally-closed clone containing F. We wite $\overline{\mathrm{Aut}(\mathbb{A})}$ to denote the clone generated by the automorphisms of the structure $\mathbb{A}$. An operation f of an oligomorphic clone F is called oligopotent if $\{g\}$ where $g(x) := f(x, \ldots, x)$ is generated by the permutations in F. We say that a k-ary operation f is a weak near-unanimity operation if $f(y, x, \ldots, x) = f(x, y, x, \ldots, x) = \cdots = f(x, \ldots, x, y)$ for all $x, y \in A$ and that f is a quasi near-unanimity operation (short, qnu-operation) if it is a weak near-unanimity and it additionally satisfies $f(x, \ldots, x) = f(x, \ldots, x, y)$ for all $x, y \in A$. We say that a k-ary operation f is a weak near-unanimity operation modulo $\overline{\mathrm{Aut}(\mathbb{A})}$ if there exist $e_1, \ldots, e_k \in \overline{\mathrm{Aut}(\mathbb{A})}$ such that: $e_1(f(y, x, \ldots, x)) = e_2(f(x, y, x, \ldots, x)) = \cdots = e_k(f(x, \ldots, x, y))$ for all $x, y \in A$.

2.3 The constraint satisfaction problem

We define the CSP to be a computational problem whose instance $\mathcal{I}$ is a triple $(\mathcal{V}, \mathcal{C}, A)$ where $\mathcal{V} = \{v_1, \ldots, v_n\}$ is a set of variables, $\mathcal{C}$ is a set of constraints each of which is of the form $((v_{i_1}, \ldots, v_{i_k}), R)$ where $\{v_{i_1}, \ldots, v_{i_k}\} \subseteq \mathcal{V}$ is *the scope* of the constraint and $R \subseteq A^k$. The question is whether there is a solution $\mathbf{s} : \mathcal{V} \to A$ to $\mathcal{I}$ satisfying $(\mathbf{s}(v_{i_1}), \ldots, \mathbf{s}(v_{i_k})) \in R$ for all $((v_{i_1}, \ldots, v_{i_k}), R) \subseteq \mathcal{C}$. Further, we define $\mathrm{CSP}(\mathbb{A})$ for a constraint language $\mathbb{A}$ to be the CSP restricted to instances where all relations come from $\mathbb{A}$. [1]

We define the projection of $((v_{i_1}, \ldots, v_{i_k}), R)$ to the set $\{w_1, \ldots, w_l\} \subseteq \{v_{i_1}, \ldots, v_{i_k}\}$ to be the constraint $(\{w_1, \ldots, w_l\}, R')$ where the relation R' is given by $(R'(w_1, \ldots, w_l) \equiv \exists x_1 \ldots \exists x_m \ R(v_{i_1}, \ldots, v_{i_k}))$ and $\{x_1, \ldots, x_m\} = \{v_{i_1}, \ldots, v_{i_k}\} \setminus \{w_1, \ldots, w_m\}$. Let $W \subseteq \mathcal{V}$. An assignment $\mathbf{a} : W \to A$ is a partial solution to $\mathcal{I}$ if $\mathbf{a}$ satisfies all projections of constraints in $\mathcal{I}$ to variables in W.

It is very well known that adding pp-definable relations to the template does not change the complexity of the problem.

▶ **Proposition 2.** *Let $\mathbb{A} = (A; R_1, \ldots, R_l)$ be a relational structure, and let R be a relation that has a primitive-positive definition in $\mathbb{A}$. Then $\mathrm{CSP}(\mathbb{A})$ and $\mathrm{CSP}(A, R, R_1, \ldots, R_l)$ are log-space equivalent.*

[1] Equivalently, one defines an instance of $\mathrm{CSP}(\mathbb{A})$ as a conjunction φ of atomic formulae over the signature of $\mathbb{A}$. Then the question is whether φ is satisfiable in $\mathbb{A}$.

B_1	=	E	N
=	=	E	N
E	E	E	E
N	N	E	N

B_2	=	E	N
=	=	E	E
E	E	E	E
N	E	E	E

B_3	=	E	N
=	=	N	N
E	N	E	N
N	N	N	N

Figure 1 Three examplary binary behaviours: B_1, B_2, and B_3.

2.4 Efficient entailment

We say that a formula φ_1 entails a formula φ_2 both over free variables $x_1, \dots, x_n$ if $(\forall x_1 \cdots \forall x_n \ (\varphi_1(x_1, \dots, x_n) \implies \varphi_2(x_1, \dots, x_n)))$ is a valid sentence. Furthermore, we say that an n-ary relation R entails φ over free variables $x_1, \dots, x_n$ if $R(x_1, \dots, x_n)$ entails φ. These definitions are quite standard but for the purposes of this paper we need a stronger notion of entailment.

▶ **Definition 3.** *We say that a quaternary relation R efficiently entails $\psi := (S_1(x_1, x_2) \implies S_2(x_3, x_4))$ where S_1, S_2 are binary relations if R entails ψ and R contains*

1. *a tuple t_1 such that $(t_1[1], t_1[2]) \in S_1$ and $(t_1[3], t_1[4]) \in S_2$, and*
2. *a tuple t_2 such that $(t_2[1], t_2[2]) \notin S_1$ and $(t_2[3], t_2[4]) \notin S_2$.*

We say that a quaternary relation R is a $[(S_1(x_1, x_2) \implies S_2(x_3, x_4)), (\varphi)]$-relation if R efficiently entails $(S_1(x_1, x_2) \implies S_2(x_3, x_4))$ and entails φ or that a quaternary relation R is a $[(S_1(x_1, x_2) \implies S_2(x_3, x_4))]$-relation if R efficiently entails $(S_1(x_1, x_2) \implies S_2(x_3, x_4))$.

3 Canonical Operations over Reducts of Homogeneous Graphs

The polymorphisms that appear in the complexity classifications of CSPs of reducts of homogeneous graphs display some regularities in the sense defined below.

Let $f : A^k \to A$, and let G be a permutation group on A. We say that f is canonical with respect to G if for all $m \in N, \alpha_1, \dots, \alpha_k \in G$ and m-tuples $a_1, \dots, a_k$, there exists $\beta \in G$ such that $\beta f(\alpha_1(a_1), \dots, \alpha_k(a_k)) = f(a_1, \dots, a_k)$. Equivalently, this means that f induces an operation $\xi^{\text{typ}}(f)$, called a k-ary behaviour, on orbits of m-tuples under G, by defining $\xi^{\text{typ}}(f)(O_1, \dots, O_k)$ as the orbit of $f(a_1, \dots, a_k)$ where a_i is any m-tuple in O_i. In what follows we are mainly interested in operations that are canonical with respect to $\text{Aut}(\mathcal{H})$ where $\mathcal{H}$ is a homogeneous graph. Therefore we usually say simply canonical. See [12] for a survey on canonical operations. Three simple binary behaviors are presented in Figure 1. According to Definition 6, a binary injection f such that $\xi^{\text{typ}}(f)$ is

- B_1 is said to be of behavior max and balanced,
- B_2 is said to be E-constant,
- B_3 is said to be of type min and N-dominated.

We introduce the following notation. Let $R_1, \dots, R_k \subseteq A^2$ be binary relations. We write $R_1 \cdots R_k$ for the binary relation on A^k defined so that: $R_1 \cdots R_k(a_1, a_2)$ holds for k-tuples $a_1, a_2 \in A^k$ if and only if $R_i(a_1[i], a_2[i])$ holds for all $i \in [k]$. Here, we can find the list of all binary behaviours of interest.

▶ **Definition 4.** *Let (A, E) be a countably infinite homogeneous graph. We say that a binary injective operation $f : A^2 \to A$ is*

- balanced *if for all $a, b \in A^2$ we have that $\text{E}{=}(a, b)$ and $={\text{E}}(a, b)$ implies $E(f(a), f(b))$ as well as $\text{N}{=}(a, b)$ and $={\text{N}}(a, b)$ implies $N(f(a), f(b))$,*

- *E-dominated (N-dominated) if for all* $a, b \in A^2$ *with* $\neq=(a,b)$ *or* $=\neq(a,b)$ *we have that* $E(f(a), f(b))$ *(*$N(f(a), f(b))$*);*
- *of behaviour* min *if for all* $a, b \in A^2$ *with* $\neq\neq(a,b)$ *we have* $E(f(a), f(b))$ *iff* $\mathrm{EE}(a,b)$*;*
- *of behaviour* max *if for all* $a, b \in A^2$ *with* $\neq\neq(a,b)$ *we have* $N(f(a), f(b))$ *iff* $\mathrm{NN}(a,b)$*;*
- *of behaviour* projection *if there exists* $i \in [2]$ *such that for all* $a, b \in A^2$ *with* $\neq\neq(a,b)$ *we have* $E(f(a), f(b))$ *iff* $E(a[i], b[i])$*,*
- *of behaviour* xor *if for all* $a, b \in A^2$ *with* $\neq\neq(a,b)$ *the relation* $E(f(a), f(b))$ *holds iff* $\mathrm{EN}(a,b)$ *or* $\mathrm{NE}(a,b)$ *holds;*
- *of behaviour* xnor *if for all* $a, b \in A^2$ *with* $\neq\neq(a,b)$ *the relation* $E(f(a), f(b))$ *holds iff* $\mathrm{EE}(a,b)$ *or* $\mathrm{NN}(a,b)$ *holds;*
- E-constant *if the image of* f *is a clique,*
- N-constant *if the image of* f *is an independent set.*

We now turn to ternary behaviours of interest.

▶ **Definition 5.** *Let* $(A; E)$ *be a countably infinite homogeneous graph. We say that a ternary injective operation* $f : A^3 \to A$ *is of behaviour*

- majority *if for all* $a, b \in D^3$ *satisfying* $\neq\neq\neq(a,b)$ *we have that* $E(f(a), f(b))$ *if and only if* $EEE(a,b), EEN(a,b), ENE(a,b)$*, or* $NEE(a,b)$*,*
- minority *if for all* $a, b \in D^3$ *satisfying* $\neq\neq\neq(a,b)$ *we have that* $N(f(a), f(b))$ *if and only if* $NNN(a,b), EEN(a,b), ENE(a,b)$*, or* $NEE(a,b)$*.*

Furthermore, let B *be a binary behavior. A ternary function is hyperplanely of behaviour* B *if the binary functions* $(x,y) \to f(x,y,c), (x,z) \to f(x,c,z)$*, and* $(y,z) \to f(c,y,z)$ *have behavior* B *for all* $c \in D$*.*

4 Consistency and Minimality

This section is devoted to the formal introduction of consistency and width notions. The main algorithm we are interested in is based on establishing minimality.

▶ **Definition 6.** *Let* $l \geq k > 0$ *be natural numbers. An instance* $\mathcal{I} = (\mathcal{V}, \mathcal{C}, A)$ *of the CSP is* (k,l)*-minimal if:*

1. *Every at most* l*-element set of variables is contained in the scope of some constraint in* $\mathcal{I}$*.*
2. *For every set* W *with* $|W| \leq k$ *and every pair of constraints* C_1 *and* C_2 *in* $\mathcal{C}$ *whose scopes contain* W*, the projections of the constraints* C_1 *and* C_2 *to* W *are the same.*

We say that $\mathcal{I}$ *is trivial if it contains a constraint with an empty relation. Otherwise, we say that* $\mathcal{I}$ *is non-trivial.*

As in the finite case, one may transform an instance $\mathcal{I}$ into an equivalent instance, i.e. with the same set of solutions by simply introducing at most $O(|\mathcal{V}|^l)$ new constraints so that the first condition in Definition 6 was satisfied and then by repeatedly removing orbits of tuples from constraints until the second condition is satisfied. Similarly to the finite CSP we have the following.

▶ **Proposition 7.** *Let* $\mathbb{A}$ *be an* ω*-categorical relational structure. Then for every instance* $\mathcal{I}$ *of* $\mathrm{CSP}(\mathbb{A})$ *and* $l \geq k > 0$ *there exists an instance* $\mathcal{I}'$ *of the CSP with the same sets of solutions as* $\mathcal{I}$ *such that* $\mathcal{I}'$ *is* (k,l)*-minimal.*

For fixed (k,l) *and* $\mathbb{A}$*, the process of establishing* (k,l)*-minimality, i.e., transfoming* $\mathcal{I}$ *into* $\mathcal{I}'$ *takes time* $O(|\mathcal{V}|^m)$ *where* $m = \max(l, arity(\mathbb{A}))$ *is the maximum of* l *and the greatest arity of a relation in* $\mathbb{A}$*. If* $\mathcal{I}'$ *is trivial, then both* $\mathcal{I}$ *and* $\mathcal{I}'$ *have no solutions.*

We are now ready to define the relational width.

▶ **Definition 8.** *We say that $\mathbb{A}$ has relational width (k,l) if and only if $\mathcal{I}$ has a solution provided any (k,l)-minimal instance of the CSP equivalent to I is non-trivial. We say that $\mathbb{A}$ has (relational) bounded width* [2] *if there exist (k,l) such that $\mathbb{A}$ has relational width (k,l).*

Finite structures with bounded relational width admit an algebraic characterization [2]. It is known that a finite structure $\mathbb{A}$ has bounded (relational) width if and only if it has a four-ary polymorphism f and a ternary polymorphism g that are weak near-unanimity operations and such that $f(y,x,x,x) = g(y,x,x)$ for all $x, y \in A$. We have a similar sufficient condition for reducts of finitely bounded homogeneous structures.

▶ **Theorem 9** ([10])**.** *Let $\mathbb{A}$ be a finite-signature reduct of a finitely bounded homogeneous structure $\mathbb{B}$. Suppose that $\mathbb{A}$ has a four-ary polymorphism f and a ternary polymorphism g that are canonical with respect to $\mathrm{Aut}(\mathbb{B})$ and are weak near-unanimity operations modulo $\overline{\mathrm{Aut}(\mathbb{B})}$, and such that there are operations $e_1, e_2 \in \overline{\mathrm{Aut}(\mathbb{B})}$ with $e_1(f(y,x,x,x)) = e_2(g(y,x,x))$ for all $x, y \in A$. Then $\mathrm{CSP}(\mathbb{A})$ has bounded relational width.*

A slight change in the proof of the above theorem gives us the upper bound for relational width of infinite structures under consideration.

▶ **Corollary 10.** *Let $\mathbb{A}$ be a finite-signature reduct of a finitely bounded homogeneous structure $\mathbb{B}$. Suppose that $\mathbb{A}$ has a four-ary polymorphism f and a ternary polymorphism g that are canonical with respect to $\mathrm{Aut}(\mathbb{B})$, that are weak near-unanimity operations modulo $\overline{\mathrm{Aut}(\mathbb{A})}$, and such that there are operations $e_1, e_2 \in \overline{\mathrm{Aut}(\mathbb{B})}$ with $e_1(f(y,x,x,x)) = e_2(g(y,x,x))$ for all $x, y \in A$. Then $\mathbb{A}$ has relational width $(2m, 3m)$ where $m = \max(arity(\mathbb{A}) + 1, arity(\mathbb{B}) + 1, \mathbb{L}_{\mathbb{B}}, 3)$.*

We now use Corollary 10 to provide the upper bound of the relational width for reducts of homogeneous graphs preserved by binary canonical operations considered in [13, 14, 9].

▶ **Proposition 11.** *Let $\mathbb{A}$ be a reduct of a countably infinite homogeneous graph $\mathcal{H}$ preserved by a binary injection:*

1. *of behaviour max which is either balanced or E-dominated, or*
2. *of behaviour min which is either balanced or N-dominated, or*
3. *which is E-constant, or*
4. *which is N-constant.*

Then it has relational width $(2m, 3m)$ where $m = \max(arity(\mathbb{A}) + 1, \mathbb{L}_{\mathcal{H}}, 3)$.

In Section 6.1, we use our approach to show that the exact relational width of structures under consideration in Proposition 11 is $(2, \mathbb{L}_{\mathcal{H}})$. The same is proved for first-order expansions of homogeneous graphs with bounded strict width.

Strict width is defined as follows. A (k,l)-minimal instance $\mathcal{I}$ of the CSP is called globally consistent, if every partial solution of $\mathcal{I}$ can be extended to a total solution of $\mathcal{I}$.

▶ **Definition 12.** *We say that $\mathbb{A}$ has strict width k if for some $l \geq k \geq 2$ all instances of $\mathrm{CSP}(\mathbb{A})$ that are (k,l)-minimal are globally consistent. We say that $\mathbb{A}$ has bounded strict width if it has strict width k for some k.* [3]

We have the following algebraic characterization of ω-categorical structures with bounded strict width.

---

[2] We note that the definition of bounded width provided in [5] is equivalent to ours.

[3] Our definition of strict width slightly varies from a definition in [5] but again both definitions are equivalent.

▶ **Theorem 13** ([5, 4]). *Let $\mathbb{A}$ be an ω-categorical language. Then the following are equivalent.*

1. *$\mathbb{A}$ has strict width k.*
2. *$\mathbb{A}$ has an oligopotent $(k+1)$-ary quasi near-unanimity operation as a polymorphism.*

For a $(2,k)$-minimal instance over variables $\mathcal{V} = \{v_1, \dots, v_n\}$ we write $\mathcal{I}_{i,j}$ with $i,j \in [n]$ to denote a subset of $\{E, N, =\}$ such that the projection of all constraints having v_i, v_j in its scope to $\{v_i, v_j\}$ equals $\bigcup_{O \in \mathcal{I}_{i,j}} O$. We will say that an instance is *simple* if $|\mathcal{I}_{i,j}| = 1$ for all $i,j \in [n]$.

We will now show that a simple non-trivial $(2, \mathbb{L}_{\mathcal{H}})$-instance of CSP($\mathbb{A}$) for a first-order expansion $\mathbb{A}$ of a homogeneous graph $\mathcal{H}$ always has a solution and that this amount of consistency is necessary.

▶ **Observation 14.** *Let $\mathcal{I}$ be a simple non-trivial $(2, \mathbb{L}_{\mathcal{H}})$-minimal instance of the CSP equivalent to an instance of* CSP($\mathcal{H}'$) *where $\mathcal{H}'$ is the expansion of $\mathcal{H}$ containing all orbitals pp-definable in $\mathcal{H}$. Then $\mathcal{I}$ has a solution.*

On the other hand, for every homogeneous graph $\mathcal{H}$ there exists a simple non-trivial $(1, \mathbb{L}_{\mathcal{H}})$-minimal instance $\mathcal{I}_1$ equivalent to an instance of CSP($\mathcal{H}'$) *and a simple non-trivial $(2, \mathbb{L}_{\mathcal{H}} - 1)$-minimal instance $\mathcal{I}_2$ equivalent to an instance of* CSP($\mathcal{H}'$) *that have no solutions.*

Proof. We start from proving the first part of the observation. Define Δ to be a finite structure over the domain consisting of variables $\{v_1, \dots, v_n\}$ in $\mathcal{I}$ and the signature $\tau \subseteq \{E, N, =\}$ such that $(v_i, v_j) \in R^{\Delta}$ for $i,j \in [n]$ and $R \in \tau$ if φ_I contains a constraint $((v_i, v_j), R)$. Since $\mathcal{I}$ is $(2,3)$-minimal, we have the following.

▶ **Observation 15.** *The binary relation $\sim := \{(v_i, v_j) \mid \mathcal{I}_{i,j} \subseteq \{=\}\} \cup \bigcup_{i \in [n]} \{(v_i v_i)\}$ is an equivalence relation.*

We claim that there is an embedding from $\Delta / \sim$ to $\mathcal{H}'$. Assume the contrary. Since $\mathcal{H}$ is finitely bounded, there exists G over variables $\{w_1, \dots, w_l\}$ in $\mathcal{F}_{\mathcal{H}'}$ such that G embeds into $\Delta / \sim$ and $l \leq \mathbb{L}_{\mathcal{H}}$. Since $\mathcal{I}$ is $(2, \mathbb{L}_{\mathcal{H}})$-minimal, there is a constraint C in $\mathcal{I}$ whose scope contains $\{w_1, \dots, w_l\}$ and the corresponding relation is empty. It contradicts with the assumption that $\mathcal{I}$ is non-trivial. Thus, $\Delta / \sim$ embeds into $\mathcal{H}$, and in consequence $\mathcal{I}$ has a solution. It completes the proof of the first part of the observation.

For the second part of the observation, we select $\mathcal{I}_1$ to be $\{((v_1, v_2), E), ((v_1, v_2), N)\}$. Indeed, every subset of variables of $\mathcal{I}_1$ is in the scope of some constraint. The projection of each constraint to $\{v_1\}$ or $\{v_2\}$ is the set of all vertices in $\mathcal{H}$. It follows that $\mathcal{I}_1$ is $(1, \mathbb{L}_{\mathcal{H}})$-consistent. Clearly $\mathcal{I}_1$ has no solutions.

We now turn to $\mathcal{I}_2$. If $\mathbb{L}_{\mathcal{H}} > 3$ and $G = ([n], E)$ is a forbidden subgraph of size $n = \mathbb{L}_{\mathcal{H}}$ consider an instance $\mathcal{I}_2'$ over variables $\{v_1, \dots, v_n\}$ containing a constraint $((v_i, v_j), E)$ if $(i,j) \in E^G$ and a constraint $((v_i, v_j), N)$ if $(i,j) \notin E^G$. Let $\mathcal{I}_2$ be a $(2, \mathbb{L}_{\mathcal{H}} - 1)$-minimal instance of the CSP equivalent to $\mathcal{I}_2'$. By the minimality of $\mathcal{F}_{\mathcal{H}}$, we have that no induced subgraph of G is in $\mathcal{F}_{\mathcal{H}}$. It follows that $\mathcal{I}_2$ is non-trivial but, clearly, $\mathcal{I}_2$ has no solutions. If $\mathbb{L}_{\mathcal{H}} = 3$, then we select $\mathcal{I}_2$ to be an instance such that $\mathcal{C} = \{((v_1, v_2), =), ((v_2, v_3), =), ((v_1, v_3), E)$. It is again straightforward to check that $\mathcal{I}_2$ is $(2,2)$-minimal. Yet, it has no solutions. It completes the proof of the observation. ◀

We complete this section by giving some examples of first-order expansions of homogeneous graphs with bounded strict width.

▶ **Proposition 16.** *Let $\mathbb{A}$ be a first-order expansion of the random graph $\mathcal{H} = (A; E)$ such that every relation in $\mathbb{A}$ is pp-definable as a conjunction of clauses of the form:*

$$(x_1 \neq y_1 \vee \dots \vee x_k \neq y_1 \vee R(y_1, y_2) \vee y_2 \neq z_1 \vee \dots \vee y_2 \neq z_l),$$

where $R \in \{E, N\}$. Then $\mathbb{A}$ has bounded strict-width.

And here comes another example.

▶ **Proposition 17.** *The constraint language* $\mathbb{A} = (A; E, N, R)$ *where* $(A; E)$ *is* C_2^ω *and* $R(x_1, x_2, x_3) \equiv ((E(x_1, x_2) \wedge N(x_2, x_3)) \vee (N(x_1, x_2) \wedge E(x_2, x_3)))$ *has bounded strict width.*

5 Conditions Sufficient for Low Relational Width

In order to show that a non-trivial $(2, \mathbb{L}_\mathcal{H})$-minimal instance $\mathcal{I}$ of CSP($\mathbb{A}$) for first-order expansions $\mathbb{A}$ of a homogeneous graph $\mathcal{H}$ has a solution, we always use one scheme. We take advantage of the fact that certain quaternary and ternary relations are not pp-definable in $\mathbb{A}$ and we carefully narrow down $\mathcal{I}_{i,j}$ for $i, j \in [n]$ so that we end up with a simple non-trivial instance $\mathcal{I}'$ which is a 'subinstance' of $\mathcal{I}$ in the following sense: for every $C = ((x_1, \dots, x_r), R)$ in $\mathcal{I}$ we have $C' = ((x_1, \dots, x_r), R') \in \mathcal{I}'$ such that $R' \subseteq R$. Since $\mathcal{I}'$ is simple, by Observation 14, it has a solution. This solution is clearly a solution to the orginal instance $\mathcal{I}$.

We shrink an instance of CSP($\mathbb{A}$) using one of three different sets of relations presented in the three lemmas below.

▶ **Lemma 18.** *Let* $\{O_1, O_2\}$ *be* $\{E, N\}$ *and* $\mathbb{A}$ *be a first-order expansion of a homogeneous graph* $\mathcal{H}$ *such that none of the following types of relations is pp-definable in* $\mathbb{A}$*:*

1. $[(O_1(x_1, x_2) \implies \underline{\underline{\mathrm{O}_2}}(x_3, x_4))]$*-relations,*
2. $[(O_1(x_1, x_2) \implies x_3 = x_4)]$*-relations, and*
3. $[(O_2(x_1, x_2) \implies x_3 = x_4), (\underline{\underline{\mathrm{O}_2}}(x_1, x_2) \wedge \underline{\underline{\mathrm{O}_2}}(x_3, x_4))]$*-relations.*

Then $\mathbb{A}$ *has relational width* $(2, \mathbb{L}_\mathcal{H})$*.*

Before we discuss the 'shrinking' strategy that stands behind Lemma 18, consider a non-trivial instance $\mathcal{I}$ of some $\mathbb{A}$ under consideration in the lemma and a constraint $((x_1, \dots, x_r), R)$ for which there are i_1, j_1, i_2, j_2 such that $v_{i_1}, v_{j_1}, v_{i_2}, v_{j_2} \in \{x_1, \dots, x_r\}$ and $O_1 \in \mathcal{I}_{i_1,j_1}$, $O_1 \in \mathcal{I}_{i_2,j_2}$. Since $\mathcal{I}$ is non-trivial and $(2, 3)$-minimal the relation $(R'(x_1, \dots, x_r) \equiv (R(x_1, \dots, x_r) \wedge O_1(v_{i_1}, v_{j_1})))$ is non-empty. But also $(R''(x_1, \dots, x_r) \equiv (R(x_1, \dots, x_r) \wedge O_1(v_{i_1}, v_{j_1}) \wedge O_1(v_{i_2}, v_{j_2})))$ is non-empty. Indeed, otherwise since $O_1 \cup \underline{\underline{\mathrm{O}_2}} = A^2$, the structure $\mathbb{A}$ would define a relation from Item 1 or Item 2. Generalizing the argument, one can easily transform $\mathcal{I}$ to a non-trivial $\mathcal{I}'$ where every $\mathcal{I}'_{i,j} = \{O_1\}$ whenever $\mathcal{I}_{i,j}$ contains O_1. Using a similar reasoning and Item 3, and taking care of some details, we have to skip here, one can then transform $\mathcal{I}'$ to $\mathcal{I}''$ so that $\mathcal{I}''_{i,j} = O_2$ whenever $\mathcal{I}'_{i,j}$ contains O_2. Since $\mathcal{I}''$ is simple and non-trivial, we can use Observation 14 to argue that both $\mathcal{I}''$ and $\mathcal{I}$ has a solution.

The next lemma considers a specific situation where $\mathcal{H}$ is a disjoint sum of ω edges and languages under consideration are preserved by oligopotent qnu-operations.

▶ **Lemma 19.** *Let* $\mathbb{A}$ *be a first-order expansion of* C_ω^2 *preserved by an oligopotent qnu-operation and such that none of the following types of relations is pp-definable in* $\mathbb{A}$*:*

1. $[(N(x_1, x_2) \implies \underline{\mathrm{E}}(x_3, x_4))]$*-relations,*
2. $[(N(x_1, x_2) \implies E(x_3, x_4)), (\underline{\mathrm{E}}(x_3, x_4))]$*-relations,*
3. $[(N(x_1, x_2) \implies x_3 = x_4)]$*-relations,*
4. $[(O_1(x_1, x_2) \implies O_2(x_3, x_4)), (\underline{\mathrm{E}}(x_1, x_2) \wedge N(x_2, x_3) \wedge \underline{\mathrm{E}}(x_3, x_4))]$*-relations where the set* $\{O_1, O_2\}$ *equals* $\{E, =\}$*.*

Then $\mathbb{A}$ *has relational width* $(2, 3)$*.*

The shrinking strategy for Lemma 19 is as follows. We start with a non-trivial $(2, 3)$-minimal instance $\mathcal{I}$ and use Items 1–3 to transform it into a non-trivial $(2, 3)$-minimal $\mathcal{I}'$ such that $\mathcal{I}'_{i,j} = \{N\}$ whenever $\mathcal{I}_{i,j}$ contains N. Since $\underline{\mathrm{E}}$ fo-definable in C_ω^2 is transitive and

$\mathcal{I}'$ is $(2,3)$-minimal, it is easy to show that the graph over variables $\{v_1,\dots,v_n\}$ and edges $\mathcal{I}'_{i,j}$ with $i,j\in[n]$ is a disjoint union of components $K_1,\dots,K_\kappa$ such that for all $k\in[\kappa]$ and all $v_i,v_j\in K_k$ it holds that $\mathcal{I}'_{i,j}\subseteq\{E,=\}$ and whenever v_i,v_j are in different components, then $\mathcal{I}'_{i,j}=\{N\}$. Now, any $\mathcal{I}'_{K_i}$ – the instance $\mathcal{I}'$ restricted to variables in K_i, which is in fact an instance of CSP(Δ) for Δ over two-elements (some edge in C^2_ω, different for every $i\in[\kappa]$) preserved by a near-unanimity operation, is shown to have a solution $\mathbf{s}_i$. It follows by the characterization of relational width for finite structure. In order to prove that solution $\mathbf{s}:=\bigcup_{i\in[\kappa]}\mathbf{s}_i$ is the solution to $\mathcal{I}'$ and hence to $\mathcal{I}$ we use the fact that relations from Item 4 are not pp-definable in $\mathbb{A}$.

Finally, we turn to the case where $\mathcal{H}$ is a disjoint sum of two infinite cliques and the structures $\mathbb{A}$ have oligopotent qnu-operations as polymorphisms.

▶ **Lemma 20.** *Let $\mathbb{A}$ be a first-order expansion of C^ω_2 preserved by an oligopotent qnu-operation and such that $\mathbb{A}$ pp-defines neither $\underline{\underline{\mathrm{N}}}$ nor $[(O(x_1,x_2)\to x_3=x_4)]$ for any $O\in\{E,N\}$. Then $\mathbb{A}$ has relational width $(2,3)$.*

Clearly any tuple over C^ω_2 takes some of its values from one equivalence class in C^ω_2 and the remaining values from the other class. In order to prove Lemma 20, we consider a non-trivial $(2,3)$-minimal instance $\mathcal{I}$ of CSP($\mathbb{A}$) but this time we also assume without loss of generality that there are no i,j with $\mathcal{I}_{i,j}=\{=\}$. Since $\mathbb{A}$ does not define $\underline{\underline{\mathrm{N}}}$ we have that for all $i,j\in[n]$ the set $\mathcal{I}_{i,j}$ contains E. Then we transform $\mathcal{I}$ to $\mathcal{I}_B$ of CSP(Δ) where Δ is over the domain $\{0,1\}$ by replacing any tuple t in any relation in any constraint in $\mathcal{I}$ by a tuple over $\{0,1\}$ so that all values in one equivalence class are replaced by 0 and all values in the other equivalence class are replaced by 1. Since Δ is preserved by a near-unanimity operation, and hence has bounded relational width we have that the $(2,3)$-minimal $\mathcal{I}_B$ has a solution $\mathbf{s}_B:\{v_1,\dots,v_m\}\to\{0,1\}$. We use $\mathbf{s}_B$ to transform $\mathcal{I}$ to $\mathcal{I}'$ so that we set $\mathcal{I}'_{i,j}$ to $\{N\}$ whenever $\mathbf{s}_B(v_i)\neq\mathbf{s}_B(v_j)$. No $[(N(x_1,x_2)\to x_3=x_4)]$-relations are pp-definable in $\mathbb{A}$, and hence we have that $\mathcal{I}'_{i,j}$ for any $i,j\in[n]$ contains E. Since $\mathbb{B}$ pp-defines no $[(E(x_1,x_2)\to x_3=x_4)]$-relations we may transform $\mathcal{I}'$ into $\mathcal{I}''$ so that $\mathcal{I}''_{i,j}$ is E whenever $\mathcal{I}'_{i,j}\neq\{N\}$. Thus, $\mathcal{I}''$ is a simple non-trivial $(2,3)$-minimal instance. It follows by Observation 14 that both $\mathcal{I}''$ and $\mathcal{I}$ have a solution. Again, we skipped many details but our goal was rather to convey some intuitions that stand behind the proofs of the lemmas in this section.

6 Constraint Languages with Low Relational Width

In this section we employ lemmas from Section 5 to provide the exact characterization of relational width of first-order expansions of homogeneous graphs with bounded strict width and first-order expansions of homogeneous graphs preserved by binary canonical operations from Proposition 11. In order to prove the former, we also show which quaternary relations of interest are violated by ternary injections used in the complexity classification (see Subsection 6.2). To rule out some other relations, we have to use oligopotent qnu-operations directly (see Subsection 6.3).

6.1 Binary Injections and Low Relational Width

We start with first-order expansions $\mathbb{A}$ of homogeneous graphs $\mathcal{H}$ whose tractability has been shown in Proposition 8.22 in [13], Proposition 6.2 in [14] as well as Proposition 37 and Theorem 62 in [9].

▶ **Lemma 21.** *Let $\mathbb{A}$ be a first-order expansion of a countably infinite homogeneous graph $\mathcal{H}$ preserved by a binary injection:*

1. *of behaviour max which is either balanced or E-dominated, or*
2. *of behaviour min which is either balanced or N-dominated, or*
3. *which is E-constant, or*
4. *which is N-constant.*

Then $\mathbb{A}$ has relational width $(2, \mathbb{L}_{\mathcal{H}})$.

The above lemma gives an opportunity to reformulate the dichotomy results for reducts $\mathbb{A}$ of $C^1_\omega, C^\omega_1, C^\omega_\omega$ and H_k for any $k \geq 3$.

▶ **Corollary 22.** *Let $\mathbb{A}$ be a reduct of a homomorphism graph $\mathcal{H}$ which is $C^1_\omega, C^\omega_1, C^\omega_\omega$ or H_k for any $k \geq 3$. Then either* $\mathrm{CSP}(\mathbb{A})$ *is NP-complete or $\mathbb{A}$ has relational width $(2, \mathbb{L}_{\mathcal{H}})$.*

Proof. We have that any tractable first-order expansion of $(\mathbb{N}; =, \neq)$ is preserved by a binary injection [7]. It follows that every tractable reduct of C^ω_1 is either preserved by a constant operation or is a first-order expansion of C^ω_1 and preserved by a binary injection which is of behaviour max and E-dominated. A similar reasoning holds for reducts of C^1_ω with a difference that we replace E with N. Further, every reduct of C^ω_ω is either homomorphically equivalent to a reduct of $(\mathbb{N}; =)$ or pp-defines both E and N, see Theorem 4.5 [14]. In the former case we are done, while in the latter a tractable $\mathbb{A}$ is preserved by a binary injection of behaviour min and balanced, Corollary 7.5 in [14]. The corollary follows by Lemma 21. By Proposition 15 and Lemma 17 in [9], a tractable reduct of H_k with $k \geq 3$ is either homomorphically equivalent to a reduct of $(\mathbb{N}; =)$ or pp-defines both E and N. In the former case we are done while in the latter, we have that $\mathrm{CSP}(\mathbb{A})$ is in P when it is preserved by a binary injection of behaviour min and N-dominated (see Theorem 38 in [9]). Again, the corollary follows by Lemma 21. ◀

6.2 Types of Relations violated by Ternary Canonical Operations

Here we look at quaternary relations of interest violated by canonical ternary operations. We start with ternary injections of behaviour majority.

▶ **Lemma 23.** *Let $\mathbb{A}$ be a reduct of a countably infinite homogeneous graph preserved by a ternary injection of behaviour majority which additionally is:*

- *hyperplanely balanced and of behaviour projection, or*
- *hyperplanely E-constant, or hyperplanely N-constant, or*
- *hyperplanely of behaviour max and E-dominated, or*
- *hyperplanely of behaviour min and N-dominated.*

Then $\mathbb{A}$ pp-defines no $[(O(x_1, x_2) \implies x_3 = x_4)]$-relations with $O \in \{E, N\}$.

We continue with ternary injections of behaviour minority.

▶ **Lemma 24.** *Let $\mathbb{A}$ be a reduct of a countably infinite homogeneous graph preserved by a ternary injection of behaviour minority which additionally is:*

- *hyperplanely balanced and of behaviour projection,*
- *hyperplanely of behaviour projection and E-dominated, or*
- *hyperplanely of behaviour projection and N-dominated, or*
- *hyperplanely balanced of behaviour xnor, or*
- *hyperplanely balanced of behaviour xor.*

Then $\mathbb{A}$ does not pp-define $[(O(x_1, x_2) \implies x_3 = x_4)]$-relations with $O \in \{E, N\}$.

We are already in the position to prove that another large family of CSP($\mathbb{A}$) under consideration may be solved by establishing minimality.

▶ **Corollary 25.** *Let $\mathbb{A}$ be a first-order expansion of C_2^ω preserved by a canonical ternary injection of behaviour minority which is hyperplanely balanced of behaviour xnor and an oligopotent qnu-operation. Then $\mathbb{A}$ has relational width $(2,3)$.*

Proof. By Lemma 24, the structure $\mathbb{A}$ does not pp-define $[(O(x_1,x_2) \implies x_3 = x_4)]$ with $O \in \{E,N\}$. Since a canonical ternary injection of behaviour minority which is hyperplanely balanced of behaviour xnor does not preserve $\underline{\underline{\mathrm{N}}}$, the result follows by appealing to Lemma 20. ◀

The third lemma of this subsection takes care of the third kind of ternary operations that occurrs in complexity classifications of CSPs for reducts of homogeneous graphs.

▶ **Lemma 26.** *Let $\mathbb{A}$ be a reduct of a countably infinite homogeneous graph preserved by a ternary canonical operation h with $h(N,\cdot,\cdot) = h(\cdot,N,\cdot) = h(\cdot,\cdot,N) = N$ and which behaves like a minority on $\{E,=\}$, i.e., h satisfies the behaviour B such that $B(E,E,E) = B(E,=,=) = B(=,E,=) = B(=,=,E) = E$ and $B(=,=,=) = B(=,E,E) = B(E,=,E) = B(E,E,=)$ equals $=$. Then $\mathbb{A}$ pp-defines none of the following types of relations:*

- *$[(N(x_1,x_2) \implies \underline{\underline{\mathrm{E}}}(x_3,x_4))]$-relations,*
- *$[(N(x_1,x_2) \implies x_3 = x_4)]$-relations,*
- *$[(N(x_1,x_2) \implies E(x_3,x_4)), (\underline{\underline{\mathrm{E}}}(x_3,x_4))]$-relations.*

6.3 Types of Relations violated by Oligopotent QNUs

Here we provide a list of quaternary relations of interest violated by ternary canonical operations and oligopotent qnu-operations. We start with the case where the considered homogeneous graph is the random graph.

▶ **Lemma 27.** *Let $\mathbb{A}$ be a first-order expansion of the random graph preserved by a ternary injection of behaviour majority which additionally satisfies one of the conditions in Lemma 23 or of behaviour minority which additionally satisfies one of the conditions in Lemma 24, and an oligopotent qnu-operation. Then $\mathbb{A}$ pp-defines at most one of the following:*

1. *either a $[(E(x_1,x_2) \implies \underline{\underline{\mathrm{N}}}(x_3,x_4))]$-relation or*
2. *a $[(N(x_1,x_2) \implies \underline{\underline{\mathrm{E}}}(x_3,x_4))]$-relation.*

Here comes the corollary.

▶ **Corollary 28.** *Let $\mathbb{A}$ be a first-order expansion of the random graph preserved by a ternary injection of behaviour majority which additionally satisfies one of the conditions in Lemma 23 or of behaviour minority which additionally satisfies one of the conditions in Lemma 24 and an oligopotent qnu-operation. Then $\mathbb{A}$ has relational width $(2,3)$.*

Proof. By appeal to Lemma 27, it follows that there are $\{O_1,O_2\} = \{E,N\}$ such that $\mathbb{A}$ does not pp-define a $[(O_1(x_1,x_2) \implies \underline{\underline{O_2}}(x_3,x_4))]$-relation. By Lemmas 23 and 24, $\mathbb{A}$ pp-defines neither $[(O_1(x_1,x_2) \implies x_3 = x_4)]$-relations nor $[(O_2(x_1,x_2) \implies x_3 = x_4)]$-relations. Since $\mathbb{L}_\mathcal{H}$ in the case where $\mathcal{H}$ is the random graph equals 3, the result follows by Lemma 18. ◀

We now turn to the case where the considered homogeneous graph is the disjoint union of ω edges.

▶ **Lemma 29.** *Let $\mathbb{A}$ be a first-order expansion of C^2_ω preserved by a ternary canonical operation h with $h(N,\cdot,\cdot) = h(\cdot,N,\cdot) = h(\cdot,\cdot,N) = N$ and which behaves like a minority on $\{E,=\}$ and an oligopotent qnu-operation. Then $\mathbb{A}$ pp-defines neither*

- *a $[(E(x_1,x_2) \implies (x_3 = x_4)), (\underline{\underline{\mathrm{E}}}(x_1,x_2) \wedge N(x_2,x_3) \wedge \underline{\underline{\mathrm{E}}}(x_3,x_4))]$-relation nor*
- *a $[((x_1 = x_2) \implies E(x_3,x_4)), (\underline{\underline{\mathrm{E}}}(x_1,x_2) \wedge N(x_2,x_3) \wedge \underline{\underline{\mathrm{E}}}(x_3,x_4))]$-relation.*

Then we provide another similar lemma.

▶ **Lemma 30.** *Let $\mathbb{A}$ be a first-order expansion of C^2_ω preserved by a ternary canonical operation h with $h(N,\cdot,\cdot) = h(\cdot,N,\cdot) = h(\cdot,\cdot,N) = N$ and which behaves like a minority on $\{E,=\}$ and an oligopotent qnu-operation. Then $\mathbb{A}$ pp-defines neither*

- *a $[(E(x_1,x_2) \implies E(x_3,x_4)), (\underline{\underline{\mathrm{E}}}(x_1,x_2) \wedge N(x_2,x_3) \wedge \underline{\underline{\mathrm{E}}}(x_3,x_4))]$-relation nor*
- *a $[((x_1 = x_2) \implies (x_3 = x_4)), (\underline{\underline{\mathrm{E}}}(x_1,x_2) \wedge N(x_2,x_3) \wedge \underline{\underline{\mathrm{E}}}(x_3,x_4))]$-relation.*

Then comes the corollary.

▶ **Corollary 31.** *Let $\mathbb{A}$ be a first-order expansion of C^2_ω preserved by a ternary canonical operation h with $h(N,\cdot,\cdot) = h(\cdot,N,\cdot) = h(\cdot,\cdot,N) = N$ and which behaves like a minority on $\{E,=\}$ and an oligopotent qnu-operation. Then $\mathbb{A}$ has relational width $(2,3)$.*

Proof. By Lemmas 26, 29 and 30 none of the types of relations mentioned in Lemma 19 is pp-definable in $\mathbb{A}$. Appealing to Lemma 19 completes the proof of the corollary. ◀

6.4 The Main Result

Here we prove our main result.

▶ **Theorem 32.** *Let $\mathbb{A}$ be a first-order expansion of a countably infinite homogeneous graph $\mathcal{H}$ which has bounded strict width. Then $\mathbb{A}$ has relational width $(2, \mathbb{L}_\mathcal{H})$.*

Proof. By the classification of Lachlan and Woodrow [24], we have that $\mathcal{H}$ is either the random graph, a Henson graph H_k with a forbidden k-clique where $k \geq 3$, a disjoint set of n cliques of size s denoted by C^s_n or a complement of either C^s_n or H_k. The case where $\mathcal{H}$ is C^ω_1, C^1_ω, C^ω_ω or H_k with $k \geq 3$ follows by Corollary 22. If C^s_n is such that $3 \leq n < \omega$ or $3 \leq s < \omega$, then by Theorem 60 in [9], a first-order expansion $\mathbb{A}$ of C^s_n is either homomorphically equivalent to a reduct of $(\mathbb{N};=)$ or is not preserved by an oligopotent qnu-operation and we are done. If $\mathbb{A}$ is a first-order expansion of C^ω_2, then by Theorem 61 in [9] either it is homomorphically equivalent to a reduct of $(\mathbb{N};=)$ or is not preserved by an oligopotent qnu-operation or pp-defines both E and N and is preserved by a canonical ternary injection of behaviour minority which is hyperplanely balanced of behaviour xnor and then $\mathbb{A}$ has relational width $(2,3)$ by Corollary 25. If $\mathbb{A}$ is a first-order expansion of C^2_ω, then by Theorem 62 in [9], we have that either $\mathbb{A}$ is homomorphically equivalent to a reduct of $(\mathbb{N};=)$, or it is not preserved by an oligopotent qnu-operation or it pp-defines both E and N and is preserved by a canonical binary injection of behaviour min that is N-dominated or a ternary canonical operation h with $h(N,\cdot,\cdot) = h(\cdot,N,\cdot) = h(\cdot,\cdot,N) = N$ and which behaves like a minority on $\{E,=\}$. In the former case the language $\mathbb{A}$ has relational width $(2,3)$ by Lemma 21, in the latter by Corollary 31.

The remaining case is where $\mathbb{A}$ is a first-order expansion of the random graph G preserved by an oligopotent qnu-operation. By Theorem 6.1 in [13] we have that a first-order expansion of G is either homorphically equivalent to a reduct of $(\mathbb{N};=)$ and then we are done or pp-defines both E and N in which case, by Theorem 9.3 in [13], we have that:

- $\mathbb{A}$ is preserved by a binary injection of behaviour max which is either balanced or E-dominated, by a binary injection of behaviour min which is either balanced or N-dominated, by a binary injection which is E-constant, or a binary injection which is N-constant, and then the theorem follows by Lemma 21, or
- $\mathbb{A}$ is preserved by a ternary injection of behaviour majority which additionally satisfies one of the conditions in Lemma 23 or of behaviour minority which additionally satisfies one of the conditions in Lemma 24, and then the theorem holds by Corollary 28.

It completes the proof of the theorem. ◀

7 Summary and Future Work

In this paper we proved in particular that:

1. every first-order expansion of a homogeneous graph $\mathcal{H}$ preserved by a canonical binary operation considered in [13, 14, 9] and
2. every first-order expansion of a homogeneous graph $\mathcal{H}$ with bounded strict width has relational width exactly $(2, \mathbb{L}_{\mathcal{H}})$.

A nice consequence of the former result is that all tractable reducts of $C^1_\omega, C^\omega_1, C^\omega_\omega$ and H_k with $k \geq 3$ have relational width exactly $(2, \mathbb{L}_{\mathcal{H}})$, and thus all tractable CSP($\mathbb{A}$) may be solved by establishing $(2, \mathbb{L}_{\mathcal{H}})$-minimality. Nevertheless, we find the latter result to be the main result of this paper. It is for the following reason.

Our general strategy is that we show that constraint languages $\mathbb{A}$ under consideration do not express "too many implications", i.e., quaternary relations that efficiently entail formulas of the form $(R_1(x_1, x_2) \implies R_2(x_3, x_4))$, see definitions in Section 2.4 and lemmas in Section 5 and then use these facts in order to find a strategy of how to shrink a non-trivial $(2, \mathbb{L}_{\mathcal{H}})$-minimal instance of the CSP so that it became a simple instance. In this paper, in order to show that certain relations are not pp-definable in $\mathbb{A}$ we employ in particular some binary and ternary canonical operations. We believe that it is not in fact necessary and theorems analogous to Theorem 32 may be obtained for large families of constraint languages using only the fact that structures $\mathbb{A}$ under consideration are preserved by oligopotent qnu-operations. Thereby we believe that Question 2 may be answered in full generality.

References

1 Libor Barto. The collapse of the bounded width hierarchy. *J. Log. Comput.*, 26(3):923–943, 2016.

2 Libor Barto and Marcin Kozik. Constraint satisfaction problems solvable by local consistency methods. *J. ACM*, 61(1):3:1–3:19, 2014. An extended abstract appeared in the Proceedings of the Symposium on Foundations of Computer Science (FOCS'09). doi:10.1145/2556646.

3 Libor Barto and Michael Pinsker. The algebraic dichotomy conjecture for infinite domain constraint satisfaction problems. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, (LICS)*, pages 615–622, 2016.

4 Manuel Bodirsky and Hubert Chen. Oligomorphic clones. *Algebra Universalis*, 57(1):109–125, 2007.

5 Manuel Bodirsky and Víctor Dalmau. Datalog and constraint satisfaction with infinite templates. *J. Comput. Syst. Sci.*, 79(1):79–100, 2013. doi:10.1016/j.jcss.2012.05.012.

6 Manuel Bodirsky and Peter Jonsson. A model-theoretic view on qualitative constraint reasoning. *J. Artif. Intell. Res. (JAIR)*, 58:339–385, 2017.

7 Manuel Bodirsky and Jan Kára. The complexity of equality constraint languages. *Theory Comput. Syst.*, 43(2):136–158, 2008.

8 Manuel Bodirsky and Jan Kára. The complexity of temporal constraint satisfaction problems. *Journal of the ACM*, 57(2):1–41, 2009. An extended abstract appeared in the Proceedings of the Symposium on Theory of Computing (STOC'08).

9 Manuel Bodirsky, Barnaby Martin, Michael Pinsker, and András Pongrácz. Constraint satisfaction problems for reducts of homogeneous graphs. *CoRR*, abs/1602.05819, 2016. An extended abstract appeared in the Proceedings of the International Colloquium on Automata, Languages, and Programming (ICALP'16).

10 Manuel Bodirsky and Antoine Mottet. A dichotomy for first-order reducts of unary structures. *Logical Methods in Computer Science*, 14(2), 2018. A preliminary version of the paper appeared in the proceedings of LICS'16.

11 Manuel Bodirsky and Jaroslav Nesetril. Constraint satisfaction with countable homogeneous templates. *J. Log. Comput.*, 16(3):359–373, 2006.

12 Manuel Bodirsky and Michael Pinsker. Reducts of Ramsey structures. *AMS Contemporary Mathematics, vol. 558 (Model Theoretic Methods in Finite Combinatorics)*, pages 489–519, 2011.

13 Manuel Bodirsky and Michael Pinsker. Schaefer's theorem for graphs. *J. ACM*, 62(3):19:1–19:52, 2015. An extended abstract appeared in the Proceedings of the Symposium on Theory of Computing (STOC'11).

14 Manuel Bodirsky and Michał Wrona. Equivalence constraint satisfaction problems. In *Proceedings of Computer Science Logic*, volume 16 of *LIPICS*, pages 122–136. Dagstuhl Publishing, September 2012.

15 Andrei A. Bulatov. Bounded relational width. manuscript, 2009.

16 Andrei A. Bulatov. A dichotomy theorem for nonuniform csps. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 319–330. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.37.

17 Andrei A. Bulatov, Andrei A. Krokhin, and Peter G. Jeavons. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34:720–742, 2005. A preliminary version of the paper appeared in the proceedings of the Symposium on Logic in Computer Science (ICALP'00).

18 Peter J. Cameron. *Oligomorphic Permutation Groups*. Cambridge University Press, Cambridge, 1990.

19 Rina Dechter. From local to global consistency. *Artificial Intelligence*, 55(1):87–108, 1992.

20 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. A preliminary version appeared in the proceedings of the Symposium on Theory of Computing (STOC'93).

21 Wilfrid Hodges. *A shorter model theory*. Cambridge University Press, Cambridge, 1997.

22 Peter Jeavons, David Cohen, and Marc Gyssens. Closure properties of constraints. *Journal of the ACM*, 44(4):527–548, 1997.

23 Michael Kompatscher and Van Trung Pham. A complexity dichotomy for poset constraint satisfaction. In *34th Symposium on Theoretical Aspects of Computer Science, STACS'17*, pages 47:1–47:12, 2017.

24 A. H. Lachlan and Robert E. Woodrow. Countable ultrahomogeneous undirected graphs. *Trans. Amer. Math. Soc.*, 262:51–94, 1980.

25 Jochen Renz. Implicit constraints for qualitative spatial and temporal reasoning. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR'12*, 2012.

26 Dmitriy Zhuk. A proof of CSP dichotomy conjecture. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 331–342. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.38.

Succinct Population Protocols for Presburger Arithmetic

Michael Blondin
Département d'informatique, Université de Sherbrooke, Sherbrooke, Canada
michael.blondin@usherbrooke.ca

Javier Esparza
Fakultät für Informatik, Technische Universität München, Garching bei München, Germany
esparza@in.tum.de

Blaise Genest
Univ Rennes, CNRS, IRISA, France
blaise.genest@irisa.fr

Martin Helfrich
Fakultät für Informatik, Technische Universität München, Garching bei München, Germany
helfrich@in.tum.de

Stefan Jaax
Fakultät für Informatik, Technische Universität München, Garching bei München, Germany
jaax@in.tum.de

Abstract

In [5], Angluin et al. proved that population protocols compute exactly the predicates definable in Presburger arithmetic (PA), the first-order theory of addition. As part of this result, they presented a procedure that translates any formula φ of quantifier-free PA with remainder predicates (which has the same expressive power as full PA) into a population protocol with $2^{\mathcal{O}(\text{poly}(|\varphi|))}$ states that computes φ. More precisely, the number of states of the protocol is exponential in both the bit length of the largest coefficient in the formula, and the number of nodes of its syntax tree.

In this paper, we prove that every formula φ of quantifier-free PA with remainder predicates is computable by a leaderless population protocol with $\mathcal{O}(\text{poly}(|\varphi|))$ states. Our proof is based on several new constructions, which may be of independent interest. Given a formula φ of quantifier-free PA with remainder predicates, a first construction produces a succinct protocol (with $\mathcal{O}(|\varphi|^3)$ leaders) that computes φ; this completes the work initiated in [8], where we constructed such protocols for a fragment of PA. For large enough inputs, we can get rid of these leaders. If the input is not large enough, then it is small, and we design another construction producing a succinct protocol with *one* leader that computes φ. Our last construction gets rid of this leader for small inputs.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Theory of computation → Automata over infinite objects; Theory of computation → Logic and verification

Keywords and phrases Population protocols, Presburger arithmetic, state complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.40

Related Version A full version of this paper is available at [7], `https://arxiv.org/abs/1910.04600`.

Funding *Javier Esparza, Martin Helfrich, Stefan Jaax*: Supported by an ERC Advanced Grant (787367: PaVeS).
Michael Blondin: Supported by a Quebec–Bavaria project funded by the Fonds de recherche du Québec (FRQ), by a Discovery Grant from the Natural Sciences and Engineering Research Council of Canada (NSERC), and by the Fonds de recherche du Québec – Nature et technologies (FRQNT).

37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020).
Editors: Christophe Paul and Markus Bläser; Article No. 40; pp. 40:1–40:15

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

1 Introduction

Population protocols [3, 4] are a model of distributed computation by indistinguishable, mobile finite-state agents, intensely investigated in recent years (see e.g. [2, 10]). Initially introduced to model networks of passively mobile sensors, they have also been applied to the analysis of chemical reactions under the name of chemical reaction networks (see e.g. [14]).

In a population protocol, a collection of agents, called a *population*, randomly interact in pairs to decide whether their initial configuration satisfies a given property, e.g. whether there are initially more agents in some state A than in some state B. Since agents are indistinguishable and finite-state, their configuration at any time moment is completely characterized by the mapping that assigns to each state the number of agents that currently populate it. A protocol is said to *compute a predicate* if for every initial configuration where the predicate holds, the agents eventually reach consensus 1, and they eventually reach consensus 0 otherwise.

In a seminal paper, Angluin et al. proved that population protocols compute exactly the predicates definable in Presburger arithmetic (PA) [5]. As part of the result, for every Presburger predicate Angluin et al. construct a leaderless protocol that computes it. The construction uses the quantifier elimination procedure for PA: every Presburger formula φ can be transformed into an equivalent boolean combination of *threshold predicates* of the form $\boldsymbol{\alpha} \cdot \boldsymbol{x} > \beta$ and *remainder predicates* of the form $\boldsymbol{\alpha} \cdot \boldsymbol{x} \equiv \beta \pmod{m}$, where $\boldsymbol{\alpha}$ is an integer vector, and β, m are integers [12]. Slightly abusing language, we call the set of these boolean combinations *quantifier-free Presburger arithmetic* (QFPA)[1]. Using that PA and QFPA have the same expressive power, Angluin et al. first construct protocols for all threshold and remainder predicates, and then show that the predicates computed by protocols are closed under negation and conjunction.

The construction of [5] is simple and elegant, but it produces large protocols. Given a formula φ of QFPA, let n be the number of bits of the largest coefficient of φ in absolute value, and let m be the number of atomic formulas of φ, respectively. The number of states of the protocols of [5] grows exponentially in both n and m. In terms of $|\varphi|$ (defined as the sum of the number of variables, n, and m) they have $\mathcal{O}(2^{\text{poly}(|\varphi|)})$ states. This raises the question of whether *succinct protocols* with $\mathcal{O}(\text{poly}(|\varphi|))$ states exist for every formula φ of QFPA. We give an affirmative answer by proving that every formula of QFPA has a succinct and leaderless protocol.

Succinct protocols are the state-complexity counterpart of *fast protocols*, defined as protocols running in polylogarithmic parallel time in the size of the population. Angluin et al. showed that every predicate has a fast protocol with a leader [6], but Alistarh et al., based on work by Doty and Soloveichik [9], proved that in the leaderless case some predicates need linear parallel time [1]. Our result shows that, unlike for time complexity, succinct protocols can be obtained for every QFPA formula in both the leaderless case and the case with leaders.

The proof of our result overcomes a number of obstacles. Designing succinct leaderless protocols is particularly hard for inputs with very few input agents, because there are less resources to simulate leaders. So we produce two completely different families of protocols, one for small inputs with $\mathcal{O}(|\varphi|^3)$ agents, and a second for large inputs with $\Omega(|\varphi|^3)$ agents, and combine them appropriately.

[1] Remainder predicates cannot be directly expressed in Presburger arithmetic without quantifiers.

Large inputs. The family for large inputs is based on our previous work [8]. However, in order to obtain leaderless protocols we need a new succinct construction for boolean combinations of atomic predicates. This obstacle is overcome by designing new protocols for threshold and remainder predicates that work under *reversible dynamic initialization*. Intuitively, agents are allowed to dynamically "enter" and "leave" the protocol through the initial states (dynamic initialization). Further, every interaction can be undone (reversibility), until a certain condition is met, after which the protocol converges to the correct output for the current input. We expect protocols with reversible dynamic initialization to prove useful in other contexts, since they allow a protocol designer to cope with "wrong" non-deterministic choices.

Small inputs. The family of protocols for small inputs is designed from scratch. We exploit that there are few inputs of small size. So it becomes possible to design one protocol for each possible size of the population, and combine them appropriately. Once the population size is fixed, it is possible to design agents that check if they have interacted with all other agents. This is used to simulate the *concatenation operator* of sequential programs, which allows for boolean combinations and succinct evaluation of linear combinations.

Relation to previous work. In [8], we designed succinct protocols with leaders for systems of linear equations. More precisely, we constructed a protocol with $\mathcal{O}((m+k)(n+\log m))$ states and $\mathcal{O}(m(n+\log m))$ leaders that computes a given predicate $A\boldsymbol{x} \geq \boldsymbol{c}$, where $A \in \mathbb{Z}^{m\times k}$ and n is the number of bits of the largest entry in A and $\boldsymbol{c}$, in absolute value. Representing $A\boldsymbol{x} \geq \boldsymbol{c}$ as a formula φ of QFPA, we obtain a protocol with $\mathcal{O}(|\varphi|^2)$ states and $\mathcal{O}(|\varphi|^2)$ leaders that computes φ. However, in [8] no succinct protocols for formulas with remainder predicates are given, and the paper makes extensive use of leaders.

Organization. Sections 2 and 3 introduce basic notation and definitions. Section 4 presents the main result. Sections 5 and 6 present the constructions of the protocols for large and small inputs, respectively. Section 7 presents conclusions. For space reasons, several proofs are only sketched. Detailed proofs are given in the full version of this paper [7].

2 Preliminaries

Notation. We write $\mathbb{Z}$ to denote the set of integers, $\mathbb{N}$ to denote the set of non negative integers $\{0,1,\ldots\}$, $[n]$ to denote $\{1,2,\ldots,n\}$, and $\mathbb{N}^E$ to denote the set of all multisets over E, i.e. unordered vectors with components labeled by E. The *size* of a multiset $\boldsymbol{v} \in \mathbb{N}^E$ is defined as $|\boldsymbol{v}| \stackrel{\text{def}}{=} \sum_{e\in E} \boldsymbol{v}(e)$. The set of all multisets over E with size $s \geq 0$ is $E^{\langle s\rangle} \stackrel{\text{def}}{=} \{\boldsymbol{v} \in \mathbb{N}^E : |\boldsymbol{v}| = s\}$. We sometimes write multisets using set-like notation, e.g. $\{\!\!\{a, 2\cdot b\}\!\!\}$ denotes the multiset $\boldsymbol{v}$ such that $\boldsymbol{v}(a) = 1$, $\boldsymbol{v}(b) = 2$ and $\boldsymbol{v}(e) = 0$ for every $e \in E \setminus \{a,b\}$. The empty multiset $\{\!\!\{\}\!\!\}$ is instead denoted $\mathbf{0}$ for readability. For every $\boldsymbol{u},\boldsymbol{v} \in \mathbb{N}^E$, we write $\boldsymbol{u} \geq \boldsymbol{v}$ if $\boldsymbol{u}(e) \geq \boldsymbol{v}(e)$ for every $e \in E$. Moreover, we write $\boldsymbol{u}+\boldsymbol{v}$ to denote the multiset $\boldsymbol{w} \in \mathbb{N}^E$ such that $\boldsymbol{w}(e) \stackrel{\text{def}}{=} \boldsymbol{u}(e) + \boldsymbol{v}(e)$ for every $e \in E$. The multiset $\boldsymbol{u} \ominus \boldsymbol{v}$ is defined analogously with $-$ instead of $+$, provided that $\boldsymbol{u} \geq \boldsymbol{v}$.

Presburger arithmetic. *Presburger arithmetic* (PA) is the first-order theory of $\mathbb{N}$ with addition, i.e. $\mathsf{FO}(\mathbb{N},+)$. For example, the PA formula $\psi(x,y,z) = \exists x' \exists z' (x = x'+x') \wedge (y = z+z') \wedge \neg(z' = 0)$ states that x is even and that $y > z$. It is well-known that for every formula

of PA there is an equivalent formula of quantifier-free Presburger arithmetic (QFPA) [13], the theory with syntax given by the grammar

$$\varphi(\boldsymbol{v}) ::= \boldsymbol{a} \cdot \boldsymbol{v} > b \mid \boldsymbol{a} \cdot \boldsymbol{v} \equiv_c b \mid \varphi(\boldsymbol{v}) \wedge \varphi(\boldsymbol{v}) \mid \varphi(\boldsymbol{v}) \vee \varphi(\boldsymbol{v}) \mid \neg\varphi(\boldsymbol{v})$$

where $\boldsymbol{a} \in \mathbb{Z}^X$, $b \in \mathbb{Z}$, $c \in \mathbb{N}_{\geq 2}$, and $\equiv_c$ denotes equality modulo c. For example, the formula $\psi(x, y, z)$ above is equivalent to $(x \equiv_2 0) \wedge (y - z \geq 1)$. Throughout the paper, we refer to any formula of QFPA, or the predicate $\mathbb{N}^X \to \{0, 1\}$ it denotes, as a *predicate*. Predicates of the form $\boldsymbol{a} \cdot \boldsymbol{v} > b$ and $\boldsymbol{a} \cdot \boldsymbol{v} \equiv_c b$ are *atomic*, and they are called *threshold* and *remainder* predicates respectively. The *max-norm* $\|\varphi\|$ of a predicate φ is the largest absolute value among all coefficients occurring within φ. The *length* $\mathrm{len}(\varphi)$ of a predicate φ is the number of boolean operators occurring within φ. The *bit length* of a predicate φ, over variables X, is defined as $|\varphi| \stackrel{\text{def}}{=} \mathrm{len}(\varphi) + \log\|\varphi\| + |X|$. We lift these definitions to sets of predicates in the natural way: given a finite set P of predicates, we define its *size* $\mathrm{size}(P)$ as the number of predicates in P, its *length* as $\mathrm{len}(P) \stackrel{\text{def}}{=} \sum_{\varphi \in P} \mathrm{len}(\varphi)$, its *norm* as $\|P\| \stackrel{\text{def}}{=} \max\{\|\varphi\| : \varphi \in P\}$, and its *bit length* as $|P| \stackrel{\text{def}}{=} \mathrm{size}(P) + \mathrm{len}(P) + \log\|P\| + |X|$. Note that $\mathrm{len}(P) = 0$ iff P only contains atomic predicates.

3 Population protocols

A *population protocol* is a tuple $\mathcal{P} = (Q, T, L, X, I, O)$ where

- Q is a finite set whose elements are called *states*;
- $T \subseteq \{(\boldsymbol{p}, \boldsymbol{q}) \in \mathbb{N}^Q \times \mathbb{N}^Q : |\boldsymbol{p}| = |\boldsymbol{q}|\}$ is a finite set of *transitions* containing the set $\{(\boldsymbol{p}, \boldsymbol{p}) : \boldsymbol{p} \in \mathbb{N}^Q, |\boldsymbol{p}| = 2\}$;
- $L \in \mathbb{N}^Q$ is the *leader multiset*;
- X is a finite set whose elements are called *input variables*;
- $I \colon X \to Q$ is the *input mapping*;
- $O \colon Q \to \{0, 1, \bot\}$ is the *output mapping*.

For readability, we often write $t \colon \boldsymbol{p} \mapsto \boldsymbol{q}$ to denote a transition $t = (\boldsymbol{p}, \boldsymbol{q})$. Given $\Delta \geq 2$, we say that t is Δ-*way* if $|\boldsymbol{p}| \leq \Delta$.

In the standard syntax of population protocols T is a subset of $\mathbb{N}^2 \times \mathbb{N}^2$, and $O \colon Q \to \{0, 1\}$. These differences are discussed at the end of this section.

Inputs and configurations. An *input* is a multiset $\boldsymbol{v} \in \mathbb{N}^X$ such that $|\boldsymbol{v}| \geq 2$, and a *configuration* is a multiset $C \in \mathbb{N}^Q$ such that $|C| \geq 2$. Intuitively, a configuration represents a population of agents where $C(q)$ denotes the number of agents in state q. The *initial configuration* $C_{\boldsymbol{v}}$ *for input* $\boldsymbol{v}$ is defined as $C_{\boldsymbol{v}} \stackrel{\text{def}}{=} L + \{\!\{\boldsymbol{v}(x) \cdot I(x) : x \in X\}\!\}$.

The *support* and *b-support* of a configuration C are respectively defined as $[\![C]\!] \stackrel{\text{def}}{=} \{q \in Q : C(q) > 0\}$ and $[\![C]\!]_b = \{q \in [\![C]\!] : O(q) = b\}$. The *output* of a configuration C is defined as $O(C) \stackrel{\text{def}}{=} b$ if $[\![C]\!]_b \neq \emptyset$ and $[\![C]\!]_{\neg b} = \emptyset$ for some $b \in \{0, 1\}$, and $O(C) \stackrel{\text{def}}{=} \bot$ otherwise. Loosely speaking, if $O(q) = \bot$ then agents in state q have no output, and a population has output $b \in \{0, 1\}$ if all agents *with output* have output b.

Executions. A transition $t = (\boldsymbol{p}, \boldsymbol{q})$ is *enabled* in a configuration C if $C \geq \boldsymbol{p}$, and *disabled* otherwise. Because of our assumption on T, every configuration enables at least one transition. If t is enabled in C, then it can be *fired* leading to configuration $C' \stackrel{\text{def}}{=} C \ominus \boldsymbol{p} + \boldsymbol{q}$, which we denote $C \xrightarrow{t} C'$. For every set of transitions S, we write $C \xrightarrow{S} C'$ if $C \xrightarrow{t} C'$ for some $t \in S$.

We denote the reflexive and transitive closure of $\xrightarrow{S}$ by $\xrightarrow{S^*}$. If S is the set of all transitions of the protocol under consideration, then we simply write $\to$ and $\xrightarrow{*}$.

An execution is a sequence of configurations $\sigma = C_0C_1\cdots$ such that $C_i \to C_{i+1}$ for every $i \in \mathbb{N}$. We write σ_i to denote configuration C_i. The *output* of an execution σ is defined as follows. If there exist $i \in \mathbb{N}$ and $b \in \{0,1\}$ such that $O(\sigma_i) = O(\sigma_{i+1}) = \cdots = b$, then $O(\sigma) \stackrel{\text{def}}{=} b$, and otherwise $O(\sigma) \stackrel{\text{def}}{=} \perp$.

Computations. An execution σ is *fair* if for every configuration D the following holds:

if $|\{i \in \mathbb{N} : \sigma_i \xrightarrow{*} D\}|$ is infinite, then $|\{i \in \mathbb{N} : \sigma_i = D\}|$ is infinite.

In other words, fairness ensures that an execution cannot avoid a configuration forever. We say that a population protocol *computes* a predicate $\varphi\colon \mathbb{N}^X \to \{0,1\}$ if for every $\boldsymbol{v} \in \mathbb{N}^X$ and every fair execution σ starting from $C_{\boldsymbol{v}}$, it is the case that $O(\sigma) = \varphi(\boldsymbol{v})$. Two protocols are *equivalent* if they compute the same predicate. It is known that population protocols compute precisely the Presburger-definable predicates [5, 11].

▶ **Example 1.** Let $\mathcal{P}_n = (Q, T, \mathbf{0}, \{x\}, I, O)$ be the protocol where $Q \stackrel{\text{def}}{=} \{0, 1, 2, 3, \ldots, 2^n\}$, $I(x) \stackrel{\text{def}}{=} 1$, $O(a) = 1 \iff a = 2^n$, and T contains a transition, for each $a, b \in Q$, of the form $(a, b) \mapsto (0, a+b)$ if $a+b < 2^n$, and $(a, b) \mapsto (2^n, 2^n)$ if $a+b \geq 2^n$. It is readily seen that $\mathcal{P}_n$ computes $\varphi(x) \stackrel{\text{def}}{=} (x \geq 2^n)$. Intuitively, each agent stores a number, initially 1. When two agents meet, one of them stores the sum of their values and the other one stores 0, with sums capping at 2^n. Once an agent reaches this cap, all agents eventually get converted to 2^n.

Now, consider the protocol $\mathcal{P}'_n = (Q', T', \mathbf{0}, \{x\}, I', O')$, where $Q' \stackrel{\text{def}}{=} \{0, 2^0, 2^1, \ldots, 2^n\}$, $I'(x) \stackrel{\text{def}}{=} 2^0$, $O'(a) = 1 \iff a = 2^n$, and T' contains a transition for each $0 \leq i < n$ of the form $(2^i, 2^i) \mapsto (0, 2^{i+1})$, and a transition for each $a \in Q'$ of the form $(a, 2^n) \mapsto (2^n, 2^n)$. Using similar arguments as above, it follows that $\mathcal{P}'_n$ also computes φ, but more succinctly: While $\mathcal{P}_n$ has $2^n + 1$ states, $\mathcal{P}'_n$ has only $n + 1$ states.

Types of protocols. A protocol $\mathcal{P} = (Q, T, L, X, I, O)$ is

- *leaderless* if $|L| = 0$, and *has* $|L|$ *leaders* otherwise;
- Δ*-way* if all its transitions are Δ-way;
- *simple* if there exist $\mathtt{f}, \mathtt{t} \in Q$ such that $O(\mathtt{f}) = 0$, $O(\mathtt{t}) = 1$ and $O(q) = \perp$ for every $q \in Q \setminus \{\mathtt{f}, \mathtt{t}\}$ (i.e., the output is determined by the number of agents in $\mathtt{f}$ and $\mathtt{t}$.)

Protocols with leaders and leaderless protocols compute the same predicates [5]. Every Δ-way protocol can be transformed into an equivalent 2-way protocol with a polynomial increase in the number of transitions [8]. Finally, every protocol can be transformed into an equivalent simple protocol with a polynomial increase in the number of states [7].

4 Main result

The main result of this paper is the following theorem:

▶ **Theorem 2.** *Every predicate φ of QFPA can be computed by a leaderless population protocol $\mathcal{P}$ with $\mathcal{O}(\text{poly}(|\varphi|))$ states. Moreover, $\mathcal{P}$ can be constructed in polynomial time.*

To prove Theorem 2, we first provide a construction that uses $\ell \in \mathcal{O}(|\varphi|^3)$ leaders. If there are at least $|\boldsymbol{v}| \geq \ell$ input agents $\boldsymbol{v}$ (*large inputs*), we will show how the protocol can be made leaderless by having agents encode both their state and the state of some leader. Otherwise, $|\boldsymbol{v}| < \ell$ (*small inputs*), and we will resort to a special construction, with a single

leader, that only works for populations of bounded size. We will show how the leader can be simulated collectively by the agents. Hence, we will construct succinct protocols computing φ for large and small inputs, respectively. Formally, we prove:

▶ **Lemma 3.** *Let φ be a predicate over variables X. There exist $\ell \in \mathcal{O}(|\varphi|^3)$ and leaderless protocols $\mathcal{P}_{\geq \ell}$ and $\mathcal{P}_{<\ell}$ with $\mathcal{O}(\text{poly}(|\varphi|))$ states such that:*

(a) *$\mathcal{P}_{\geq \ell}$ computes predicate $(|\boldsymbol{v}| \geq \ell) \to \varphi(\boldsymbol{v})$, and*

(b) *$\mathcal{P}_{<\ell}$ computes predicate $(|\boldsymbol{v}| < \ell) \to \varphi(\boldsymbol{v})$.*

Theorem 2 follows immediately from the lemma: it suffices to take the conjunction of both protocols, which only yields a quadratic blow-up on the number of states, using the classical product construction [3]. The rest of the paper is dedicated to proving Lemma 3. Parts (a) and (b) are shown in Sections 5 and 6, respectively.

In the remainder of the paper, whenever we claim the existence of some protocol $\mathcal{P}$, we also claim polynomial-time constructibility of $\mathcal{P}$ without mentioning it explicitly.

5 Succinct protocols for large populations

We show that, for every predicate φ, there exists a constant $\ell \in \mathcal{O}(|\varphi|^3)$ and a succinct protocol $\mathcal{P}_{\geq \ell}$ computing $(|\boldsymbol{v}| \geq \ell) \to \varphi(\boldsymbol{v})$. Throughout this section, we say that $n \in \mathbb{N}$ is *large* if $n \geq \ell$, and that a protocol *computes φ for large inputs* if it computes $(|\boldsymbol{v}| \geq \ell) \to \varphi(\boldsymbol{v})$.

We present the proof in a top-down manner, by means of a chain of statements of the form "$A \leftarrow B$, $B \leftarrow C$, $C \leftarrow D$, and D". Roughly speaking, and using notions that will be defined in the forthcoming subsections:

- Section 5.1 introduces protocols with helpers, a special class of protocols with leaders. The section shows: φ is computable for large inputs by a succinct leaderless protocol (A), if it is computable for large inputs by a succinct protocol with helpers (B).
- Section 5.2 defines protocols that simultaneously compute a set of predicates. The section proves: (B) holds if the set P of atomic predicates occurring within φ is simultaneously computable for large inputs by a succinct protocol with helpers (C).
- Section 5.3 introduces protocols with reversible dynamic initialization. The section shows: (C) holds if each atomic predicate of P is computable for large inputs by a succinct protocol with helpers and reversible dynamic initialization (D).
- Section 5.4 shows that (D) holds by exhibiting succinct protocols with helpers and reversible dynamic initialization that compute atomic predicates for large inputs.

5.1 From protocols with helpers to leaderless protocols

Intuitively, a protocol with helpers is a protocol with leaders satisfying an additional property: adding more leaders does not change the predicate computed by the protocol. Formally, let $\mathcal{P} = (Q, T, L, X, I, O)$ be a population protocol computing a predicate φ. We say that $\mathcal{P}$ is a protocol *with helpers* if for every $L' \succeq L$ the protocol $\mathcal{P}' = (Q, T, L', X, I, O)$ also computes φ, where $L' \succeq L \stackrel{\text{def}}{=} \forall q \in Q\colon (L'(q) = L(q) = 0 \vee L'(q) \geq L(q) > 0)$. If $|L| = \ell$, then we say that $\mathcal{P}$ is a protocol *with ℓ helpers*.

▶ **Theorem 4.** *Let $\mathcal{P} = (Q, T, L, X, I, O)$ be a Δ-way population protocol with ℓ-helpers computing some predicate φ. There exists a 2-way leaderless population protocol with $\mathcal{O}(\ell \cdot |X| + (\Delta \cdot |T| + |Q|)^2)$ states that computes $(|\boldsymbol{v}| \geq \ell) \to \varphi(\boldsymbol{v})$.*

Proof sketch. By [8, Lemma 3], $\mathcal{P}$ can be transformed into a 2-way population protocol (with helpers[2]) computing the same predicate φ, and with at most $|Q|+3\Delta\cdot|T|$ states. Thus, we assume $\mathcal{P}$ to be 2-way in the rest of the sketch.

For simplicity, assume $X=\{x\}$ and $L=\{\!\{3\cdot q, 5\cdot q'\}\!\}$; that is, $\mathcal{P}$ has 8 helpers, and initially 3 of them are in state q, and 5 are in q'. We describe a leaderless protocol $\mathcal{P}'$ that simulates $\mathcal{P}$ for every input $\boldsymbol{v}$ such that $|\boldsymbol{v}|\geq|L|=\ell$. Intuitively, $\mathcal{P}'$ runs in two phases:

- In the first phase each agent gets assigned a number between 1 and 8, ensuring that each number is assigned to at least one agent (this is the point at which the condition $|\boldsymbol{v}|\geq\ell$ is needed). At the end of the phase, each agent is in a state of the form (x,i), meaning that the agent initially represented one unit of input for variable x, and that it has been assigned number i. To achieve this, initially every agent is placed in state $(x,1)$. Transitions are of the form $\{\!\{(x,i),(x,i)\}\!\}\mapsto\{\!\{(x,i+1),(x,i)\}\!\}$ for every $1\leq i\leq 7$. The transitions guarantee that all but one agent is promoted to $(x,2)$, all but one to $(x,3)$, etc. In other words, one agent is "left behind" at each step.
- In the second phase, an agent's state is a multiset: agents in state (x,i) move to state $\{\!\{I(x),q\}\!\}$ if $1\leq i\leq 3$, and to state $\{\!\{I(x),q'\}\!\}$ if $4\leq i\leq 8$. Intuitively, after this move each agent has been assigned two jobs: simultaneously simulate a regular agent of $\mathcal{P}$ starting at state x, *and* a helper of L starting at state q or q'. Since in the first phase each number is assigned to at least one agent, $\mathcal{P}'$ has at least 3 agents simulating helpers in state q, and at least 5 agents simulating helpers in state q'. There may be many more helpers, but this is harmless, because, by definition, additional helpers do not change the computed predicate.

  The transitions of $\mathcal{P}'$ are designed according to this double role of the agents of $\mathcal{P}'$. More precisely, for all multisets $\boldsymbol{p},\boldsymbol{q},\boldsymbol{p}',\boldsymbol{q}'$ of size two, $\{\!\{\boldsymbol{p},\boldsymbol{q}\}\!\}\mapsto\{\!\{\boldsymbol{p}',\boldsymbol{q}'\}\!\}$ is a transition of $\mathcal{P}'$ iff $(\boldsymbol{p}+\boldsymbol{q})\to(\boldsymbol{p}'+\boldsymbol{q}')$ in $\mathcal{P}$. ◀

5.2 From multi-output protocols to protocols with helpers

A *k-output population protocol* is a tuple $\mathcal{Q}=(Q,T,L,X,I,O)$ where $O\colon[k]\times Q\to\{0,1,\perp\}$ and $\mathcal{Q}_i\stackrel{\text{def}}{=}(Q,T,L,X,I,O_i)$ is a population protocol for every $i\in[k]$, where O_i denotes the mapping such that $O_i(q)\stackrel{\text{def}}{=}O(i,q)$ for every $q\in Q$. Intuitively, since each $\mathcal{Q}_i$ only differs by its output mapping, $\mathcal{Q}$ can be seen as a single population protocol whose executions have k outputs. More formally, $\mathcal{Q}$ *computes* a set of predicates $P=\{\varphi_1,\varphi_2,\ldots,\varphi_k\}$ if $\mathcal{Q}_i$ computes φ_i for every $i\in[k]$. Furthermore, we say that $\mathcal{Q}$ is *simple* if $\mathcal{Q}_i$ is simple for every $i\in[k]$. Whenever the number k is irrelevant, we use the term *multi-output population protocol* instead of k-output population protocol.

▶ **Proposition 5.** *Assume that every finite set A of* atomic *predicates is computed by some $|A|$-way multi-output protocol with $\mathcal{O}(|A|^3)$ helpers and states, and $\mathcal{O}(|A|^5)$ transitions. Every QFPA predicate φ is computed by some simple $|\varphi|$-way protocol with $\mathcal{O}(|\varphi|^3)$ helpers and states, and $\mathcal{O}(|\varphi|^5)$ transitions.*

Proof sketch. Consider a binary tree decomposing the boolean operations of φ. We design a protocol for φ by induction on the height of the tree.

The case where the height is 0, and φ is atomic, is trivial. We sketch the induction step for the case where the root is labeled with $\wedge$, that is $\varphi=\varphi_1\wedge\varphi_2$, the other cases are similar. By induction hypothesis, we have simple protocols $\mathcal{P}_1,\mathcal{P}_2$ computing φ_1,φ_2,

[2] Lemma 3 of [8] deals with leaders and not the more specific case of helpers. Nonetheless, computation under helpers is preserved as the input mapping of $\mathcal{P}$ remains unchanged in the proof of the lemma.

respectively. Let $\mathtt{t}_j, \mathtt{f}_j$ be the output states of $\mathcal{P}_j$ for $j \in \{1,2\}$ such that $O_j(\mathtt{t}_j) = 1$ and $O_j(\mathtt{f}_j) = 0$. We add two new states $\mathtt{t}, \mathtt{f}$ (the output states of the new protocol) and an additional helper starting in state $\mathtt{f}$. To compute $\varphi_1 \wedge \varphi_2$ we add the following transitions for every $b_1 \in \{\mathtt{t}_1, \mathtt{f}_1\}, b_2 \in \{\mathtt{t}_2, \mathtt{f}_2\}$, and $b \in \{\mathtt{t}, \mathtt{f}\}$: $\wr b_1, b_2, b \wr \mapsto \wr b_1, b_2, \mathtt{t} \wr$ if $b_1 = \mathtt{t}_1 \wedge b_2 = \mathtt{t}_2$, and $\wr b_1, b_2, b \wr \mapsto \wr b_1, b_2, \mathtt{f} \wr$ otherwise. The additional helper computes the conjunction as desired. ◀

5.3 From reversible dynamic initialization to multi-output protocols

Let $P = \{\varphi_1, \ldots, \varphi_k\}$ be a set of $k \geq 2$ atomic predicates of arity $n \geq 1$ over a set $X = \{x_1, \ldots, x_n\}$ of variables. We construct a multi-output protocol $\mathcal{P}$ for P of size $\mathrm{poly}(|\varphi_1| + \cdots + |\varphi_k|)$.

Let $\mathcal{P}_1, \ldots, \mathcal{P}_k$ be protocols for $\varphi_1, \ldots, \varphi_k$. Observe that $\mathcal{P}$ cannot be a "product protocol" that executes $\mathcal{P}_1, \ldots, \mathcal{P}_k$ synchronously. Indeed, the states of such a $\mathcal{P}$ are tuples $(q_1, \ldots, q_k)$ of states of $\mathcal{P}_1, \ldots, \mathcal{P}_k$, and so $\mathcal{P}$ would have exponential size in k. Further, $\mathcal{P}$ cannot execute $\mathcal{P}_1, \ldots, \mathcal{P}_k$ asynchronously in parallel, because, given an input $\boldsymbol{x} \in \mathbb{N}^n$, it must dispatch $k \cdot \boldsymbol{x}$ agents ($\boldsymbol{x}$ to the input states of each $\mathcal{P}_j$), but it only has $\boldsymbol{x}$. Such a $\mathcal{P}$ would need $(k-1)|\boldsymbol{x}|$ helpers, which is not possible, because a protocol of size $\mathrm{poly}(|\varphi_1| + \cdots + |\varphi_k|)$ can only use $\mathrm{poly}(|\varphi_1| + \cdots + |\varphi_k|)$ helpers, whatever the input $\boldsymbol{x}$.

The solution is to use a more sophisticated parallel asynchronous computation. Consider two copies of inputs, denoted $\overline{X} = \{\overline{x}_1, \ldots, \overline{x}_n\}$ and $\underline{X} = \{\underline{x}_1, \ldots, \underline{x}_n\}$. For each predicate φ over X, consider predicate $\tilde{\varphi}$ over $\overline{X} \cup \underline{X}$ satisfying $\tilde{\varphi}(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}}) = \varphi(k\overline{\boldsymbol{x}} + \underline{\boldsymbol{x}})$ for every $(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}}) \in \mathbb{N}^{\overline{X} \cup \underline{X}}$. We obtain $\tilde{\varphi}(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}}) = \varphi(\boldsymbol{x})$ whenever $k\overline{\boldsymbol{x}} + \underline{\boldsymbol{x}} = \boldsymbol{x}$, e.g. for $\overline{\boldsymbol{x}} := \lfloor \frac{\boldsymbol{x}}{k} \rfloor$ and $\underline{\boldsymbol{x}} := \boldsymbol{x} \bmod k$. With this choice, $\mathcal{P}$ needs to dispatch a total of $k\,(|\overline{\boldsymbol{x}} + \underline{\boldsymbol{x}}|) \leq |\boldsymbol{x}| + n \cdot (k-1)^2$ agents to compute $\tilde{\varphi}_1(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}}), \ldots, \tilde{\varphi}_k(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}})$. That is, $n \cdot (k-1)^2$ helpers are sufficient to compute $\mathcal{P}$. Formally, we define $\tilde{\varphi}$ in the following way:

$$\text{For } \varphi(\boldsymbol{x}) = \left(\sum_{i=1}^{n} \alpha_i x_i > \beta\right), \text{ we define } \tilde{\varphi}(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}}) := \left(\sum_{i=1}^{n} (k \cdot \alpha_i)\overline{x}_i + \alpha_i \underline{x}_i > \beta\right)$$

and similarly for modulo predicates. For instance, if $\varphi(x_1, x_2) = 3x_1 - 2x_2 > 6$ and $k = 4$, then $\tilde{\varphi}(\overline{x}_1, \underline{x}_1, \overline{x}_2, \underline{x}_2) = 12\overline{x}_1 + 3\underline{x}_1 - 8\overline{x}_2 - 2\underline{x}_2 > 6$. As required, $\tilde{\varphi}(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}}) = \varphi(k\overline{\boldsymbol{x}} + \underline{\boldsymbol{x}})$.

Let us now describe how the protocol $\mathcal{P}$ computes $\tilde{\varphi}_1(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}}), \ldots, \tilde{\varphi}_k(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}})$. Let $\tilde{\mathcal{P}}_1, \ldots, \tilde{\mathcal{P}}_k$ be protocols computing $\tilde{\varphi}_1, \ldots, \tilde{\varphi}_k$. Let $X = \{\mathtt{x_1}, \ldots, \mathtt{x_n}\}$ be the input states of $\mathcal{P}$, and let $\overline{\mathtt{x}}_1^{\mathtt{j}}, \ldots, \overline{\mathtt{x}}_{\mathtt{n}}^{\mathtt{j}}$ and $\underline{\mathtt{x}}_1^{\mathtt{j}}, \ldots, \underline{\mathtt{x}}_{\mathtt{n}}^{\mathtt{j}}$ be the input states of $\tilde{\mathcal{P}}_j$ for every $1 \leq j \leq k$. Protocol $\mathcal{P}$ repeatedly chooses an index $1 \leq i \leq n$, and executes one of these two actions: (a) take k agents from $\mathtt{x_i}$, and dispatch them to $\overline{\mathtt{x}}_1^{\mathtt{1}}, \ldots, \overline{\mathtt{x}}_{\mathtt{i}}^{\mathtt{k}}$ (one agent to each state); or (b) take one agent from $\mathtt{x_i}$ and $(k-1)$ helpers, and dispatch them to $\underline{\mathtt{x}}_{\mathtt{i}}^{\mathtt{1}}, \ldots, \underline{\mathtt{x}}_{\mathtt{i}}^{\mathtt{k}}$. The index and the action are chosen nondeterministically. Notice that if for some input $\mathtt{x}_i$, all ℓ agents of $\mathtt{x_i}$ are dispatched, then $k\overline{\mathtt{x}}_{\mathtt{i}}^{\mathtt{j}} + \underline{\mathtt{x}}_{\mathtt{i}}^{\mathtt{j}} = \ell$ for all j. If all agents of $\mathtt{x_i}$ are dispatched for every $1 \leq i \leq n$, then we say that the *dispatch is correct*.

The problem is that, because of the nondeterminism, the dispatch may or may not be correct. Assume, e.g., that $k = 5$ and $n = 1$. Consider the input $x_1 = 17$, and assume that $\mathcal{P}$ has $n \cdot (k-1)^2 = 16$ helpers. $\mathcal{P}$ may correctly dispatch $\overline{x}_1 = \lfloor \frac{17}{5} \rfloor = 3$ agents to each of $\overline{\mathtt{x}}_{\mathtt{1}}^{\mathtt{1}}, \ldots, \overline{\mathtt{x}}_{\mathtt{5}}^{\mathtt{1}}$ and $\underline{x}_1 = (17 \bmod 5) = 2$ to each of $\underline{\mathtt{x}}_{\mathtt{1}}^{\mathtt{1}}, \ldots, \underline{\mathtt{x}}_{\mathtt{5}}^{\mathtt{1}}$; this gives a total of $(3+2) \cdot 5 = 25$ agents, consisting of the 17 agents for the input plus 8 helpers. However, it may also wrongly dispatch 2 agents to each of $\overline{\mathtt{x}}_{\mathtt{1}}^{\mathtt{1}}, \ldots, \overline{\mathtt{x}}_{\mathtt{5}}^{\mathtt{1}}$ and 4 agents to each of $\underline{\mathtt{x}}_{\mathtt{1}}^{\mathtt{1}}, \ldots, \underline{\mathtt{x}}_{\mathtt{5}}^{\mathtt{1}}$, with a total of $(2+4) \cdot 5 = 30$ agents, consisting of 14 input agents plus 16 helpers. In the second case, each $\mathcal{P}_j$ wrongly computes $\tilde{\varphi}_j(2,4) = \varphi_j(2 \cdot 5 + 4) = \varphi_j(14)$, instead of the correct value $\varphi_j(17)$.

To solve this problem we ensure that $\mathcal{P}$ can always recall agents already dispatched to $\tilde{\mathcal{P}}_1, \ldots, \tilde{\mathcal{P}}_k$ as long as the dispatch is not yet correct. This allows $\mathcal{P}$ to "try out" dispatches until it dispatches correctly, which eventually happens by fairness. For this we design $\mathcal{P}$ so that (i) the atomic protocols $\tilde{\mathcal{P}}_1, \ldots, \tilde{\mathcal{P}}_k$ can work with inputs agents that arrive over time (*dynamic initialization*), and (ii) $\tilde{\mathcal{P}}_1, \ldots, \tilde{\mathcal{P}}_k$ can always return to their initial configuration and send agents back to $\mathcal{P}$, unless the dispatch is correct (*reversibility*). To ensure that $\mathcal{P}$ stops redistributing after dispatching a correct distribution, it suffices to replace each reversing transition $\boldsymbol{p} \mapsto \boldsymbol{q}$ by transitions $\boldsymbol{p} + \wr \mathtt{x_i} \wr \mapsto \boldsymbol{q} + \wr \mathtt{x_i} \wr$, one for each $1 \leq i \leq n$: All these transitions become disabled when $\mathtt{x_1}, \ldots, \mathtt{x_n}$ are not populated.

Reversible dynamic initialization. Let us now formally introduce the class of *protocols with reversible dynamic initialization* that enjoys all properties needed for our construction. A simple protocol with *reversible dynamic initialization* (*RDI-protocol* for short) is a tuple $\mathcal{P} = (Q, T_\infty, T_\dagger, L, X, I, O)$, where $\mathcal{P}_\infty = (Q, T_\infty, L, X, I, O)$ is a simple population protocol, and $T_\dagger$ is the set of transitions making the system reversible, called the *RDI-transitions*.

Let $T \stackrel{\text{def}}{=} T_\infty \cup T_\dagger$, and let $\mathsf{In} \stackrel{\text{def}}{=} \{\mathsf{in}_x : x \in X\}$ and $\mathsf{Out} \stackrel{\text{def}}{=} \{\mathsf{out}_x : x \in X\}$ be the sets of *input* and *output transitions*, respectively, where $\mathsf{in}_x \stackrel{\text{def}}{=} (\mathbf{0}, \wr I(x) \wr)$ and $\mathsf{out}_x \stackrel{\text{def}}{=} (\wr I(x) \wr, \mathbf{0})$. An *initialization sequence* is a finite execution $\pi \in (T \cup \mathsf{In} \cup \mathsf{Out})^*$ from the *initial configuration* L' with $L' \succeq L$. The *effective input* of π is the vector $\boldsymbol{w}$ such that $\boldsymbol{w}(x) \stackrel{\text{def}}{=} |\pi|_{\mathsf{in}_x} - |\pi|_{\mathsf{out}_x}$ for every $x \in X$. Intuitively, a RDI-protocol starts with helpers only, and is dynamically initialized via the input and output transitions.

Let $\mathtt{f}, \mathtt{t} \in Q$ be the unique states of $\mathcal{P}$ with $O(\mathtt{f}) = 0$ and $O(\mathtt{t}) = 1$. For every configuration C, let $[C] \stackrel{\text{def}}{=} \{C' : C'(\mathtt{f}) + C'(\mathtt{t}) = C(\mathtt{f}) + C(\mathtt{t}) \text{ and } C'(q) = C(q) \text{ for all } q \in Q \setminus \{\mathtt{f}, \mathtt{t}\}\}$. Intuitively, all configurations $C' \in [C]$ are equivalent to C in all but the output states.

An RDI-protocol is required to be *reversible*, that is for every initialization sequence π with effective input $\boldsymbol{w}$, and such that $L' \xrightarrow{\pi} C$ for some $L' \succeq L$, the following holds:

- if $C \xrightarrow{T^*} D$ and $D' \in [D]$, then $D' \xrightarrow{T^*} C'$ for some $C' \in [C]$, and
- $C(I(x)) \leq \boldsymbol{w}(x)$ for all $x \in X$.

Intuitively, an RDI-protocol can never have more agents in an input state than the effective number of agents it received via the input and output transitions. Further, an RDI-protocol can always reverse all sequences that do not contain input or output transitions. This reversal does not involve the states $\mathtt{f}$ and $\mathtt{t}$, which have a special role as output states. Since RDI-protocols have a default output, we need to ensure that the default output state is populated when dynamic initialization ends, and reversal for $\mathtt{f}$ and $\mathtt{t}$ would prevent that.

An RDI-protocol $\mathcal{P}$ *computes* φ if for every initialization sequence π with effective input $\boldsymbol{w}$ such that $L' \xrightarrow{\pi} C$ for some $L' \succeq L$, the standard population protocol $\mathcal{P}_\infty$ computes $\varphi(\boldsymbol{w})$ from C (that is with $T_\dagger$ disabled). Intuitively, if the dynamic initialization terminates, the RDI-transitions $T_\dagger$ become disabled, and then the resulting standard protocol $\mathcal{P}_\infty$ converges to the output corresponding to the dynamically initialized input.

▶ **Theorem 6.** *Assume that for every atomic predicate φ, there exists a $|\varphi|$-way RDI-protocol with $\mathcal{O}(|\varphi|)$ helpers, $\mathcal{O}(|\varphi|^2)$ states and $\mathcal{O}(|\varphi|^3)$ transitions that computes φ. For every finite set P of atomic predicates, there exists a $|P|$-way simple* multi-output *protocol, with $\mathcal{O}(|P|^3)$ helpers and states, and $\mathcal{O}(|P|^5)$ transitions, that computes P.*

5.4 Atomic predicates under reversible dynamic initialization

Lastly, we show that atomic predicates are succinctly computable by RDI-protocols:

▶ **Theorem 7.** *Every atomic predicate φ over variables X can be computed by a simple $|\varphi|$-way population protocol with reversible dynamic initialization that has $\mathcal{O}(|\varphi|)$ helpers, $\mathcal{O}(|\varphi|^2)$ states, and $\mathcal{O}(|\varphi|^3)$ transitions.*

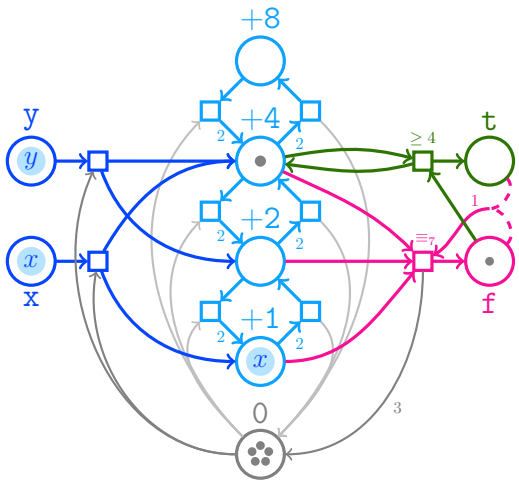


Figure 1 Partial representation of the protocol computing $5x + 6y \geq 4 \pmod 7$ as a Petri net, where places (circles), transitions (squares) and tokens (smaller filled circles) represent respectively states, transitions and agents. Non-helper agents remember their input variable (labeled here within tokens). The depicted configuration is obtained from input $x = 2$, $y = 1$ by firing the bottom leftmost transition (dark blue).

The protocols for arbitrary threshold and remainder predicates satisfying the conditions of Theorem 7, and their correctness proofs, are given in [7]. Note that the threshold protocol is very similar to the protocol for linear inequalities given in Section 6 of [8]. Thus, as an example, we will instead describe how to handle the remainder predicate $5x - y \equiv_7 4$. Note, that the predicate can be rewritten as $(5x + 6y \geq 4 \pmod 7) \wedge (5x + 6y \not\geq 5 \pmod 7)$. As we can handle negations and conjunctions separately in Section 5.2, we will now explain the protocol for $\varphi \stackrel{\text{def}}{=} 5x + 6y \geq 4 \pmod 7$. The protocol is partially depicted in Figure 1 using Petri net conventions for the graphical representation.

The protocol has an *input state* $\mathtt{x}$ for each variable $x \in X$, *output states* $\mathtt{f}$ and $\mathtt{t}$, a *neutral state* $\mathtt{0}$, and *numerical states* of the form $+2^{\mathtt{i}}$ for every $0 \leq i \leq n$, where n is the smallest number such that $2^n > \|\varphi\|$. Initially, (at least) one helper is set to $\mathtt{f}$ and (at least) $2n$ helpers set to $\mathtt{0}$. In order to compute $5x + 6y \geq 4 \pmod 7$ for $x := r$ and $y := s$, we initially place r and s agents in the states $\mathtt{x}$ and $\mathtt{y}$, i.e., the agents in state $\mathtt{x}$ encode the number r in unary, and similarly for $\mathtt{y}$. The blue transitions on the left of Figure 1 "convert" each agents in input states to a binary representation of their corresponding coefficient. In our example, agents in state $\mathtt{x}$ are converted to $\boldsymbol{a}(x) = 5 = 0101_2$ by putting one agent in $\mathtt{4}$ and another one in $\mathtt{1}$. Since two agents are needed to encode 5, the transition "recruits" one helper from state $\mathtt{0}$. Observe that, since the inputs can be arbitrarily large, but a protocol can only use a constant number of helpers, the protocol must reuse helpers in order to convert all agents in input states. This happens as follows. If two agents are in the same power of two, say $+2^{\mathtt{i}}$, then one of them can be "promoted" to $+2^{\mathtt{i}+1}$, while the other one moves to

state `0`, "liberating" one helper. This allows the agents to represent the overall value of all converted agents in the most efficient representation. That is, from any configuration, one can always reach a configuration where there is at most one agent in each place $2^0, \ldots, 2^{n-1}$, there are at most the number of agents converted from input places in place 2^n, and hence there are at least n agents in place 0, thus ready to convert some agent from the input place. Similar to promotions, "demotions" to smaller powers of two can also happen. Thus, the agents effectively shift through all possible binary representations of the overall value of all converted agents. The $\equiv_7$ transition in Figure 1 allows 3 agents in states `4`, `2` and `1` to "cancel out" by moving to state `0`, and it moves the output helper to `f`. Furthermore, there are RDI-transitions that allow to revert the effects of conversion and cancel transitions. These are not shown in Figure 1.

We have to show that this protocol computes φ under reversible dynamic initialization. First note, that while dynamic initialization has not terminated, all transitions have a corresponding reverse transition. Thus, it is always possible to return to wrong initial configurations. However, reversing the conversion transitions can create more agents in input states than the protocol effectively received. To forbid this, each input agent is "tagged" with its variable (see tokens in Figure 1). Therefore, in order to reverse a conversion transitions, the original input agent is needed. This implies, that the protocol is reversible.

Next, we need to argue that the protocol without the RDI-transitions computes φ once the dynamic initialization has terminated. The agents will shift through the binary representations of the overall value. Because of fairness, the $\equiv_7$ transition will eventually reduce the overall value to at most 6. There is a ≥ 4-transition which detects the case where the final value is at least 4 and moves the output helper from `f` to state `t`. Notice that whenever transition $\equiv_7$ occurs, we reset the output by moving the output helper to state `f`.

6 Succinct protocols for small populations

We show that for every predicate φ and constant $\ell = \mathcal{O}(|\varphi|^3)$, there exists a succinct protocol $\mathcal{P}_{<\ell}$ that computes the predicate $(|\boldsymbol{v}| < \ell) \to \varphi(\boldsymbol{v})$. In this case, we say that $\mathcal{P}_{<\ell}$ *computes* φ *for small inputs*. Further, we say that a number $n \in \mathbb{N}$ (resp. an input $\boldsymbol{v}$) is small with respect to φ if $n \leq \ell$ (resp. $|\boldsymbol{v}| \leq \ell$). We present the proof strategy in a top-down manner.

- Section 6.1 proves: There is a succinct leaderless protocol $\mathcal{P}$ that computes φ for small inputs (A), if for every small n some succinct protocol $\mathcal{P}_n$ computes φ for all inputs of size n (B). Intuitively, constructing a succinct protocol for all small inputs reduces to the simpler problem of constructing a succinct protocol for all small inputs of a fixed size.
- Section 6.2 introduces halting protocols. It shows: There is a succinct protocol that computes φ for inputs of size n, if for every *atomic* predicate ψ of φ some halting succinct protocol computes ψ for inputs of size n (C). Thus, constructing protocols for arbitrary predicates reduces to constructing *halting* protocols for atomic predicates.
- Section 6.3 proves (C). Given a threshold or remainder predicate φ and a small n, it shows how to construct a succinct halting protocol that computes φ for inputs of size n.

6.1 From fixed-sized protocols with one leader to leaderless protocols

We now define when a population protocol computes a predicate *for inputs of a fixed size*. Intuitively, it should compute the correct value for every initial configurations of this size; for inputs of other sizes, the protocol may converge to the wrong result, or may not converge.

▶ **Definition 8.** *Let φ be a predicate and let $i \geq 2$. A protocol $\mathcal{P}$* computes *φ for inputs of size i, denoted "$\mathcal{P}$ computes $(\varphi \mid i)$", if for every input $\boldsymbol{v}$ of size i, every fair execution of $\mathcal{P}$ starting at $C_{\boldsymbol{v}}$ stabilizes to $\varphi(\boldsymbol{v})$.*

We show that if, for each small number i, some succinct protocol computes $(\varphi \mid i)$, then there is a single succinct protocol that computes φ for all small inputs.

▶ **Theorem 9.** *Let φ be a predicate over a set of variables X, and let $\ell \in \mathbb{N}$. Assume that for every $i \in \{2, 3, \ldots, \ell - 1\}$, there exists a protocol with at most one leader and at most m states that computes $(\varphi \mid i)$. Then, there is a leaderless population protocol with $\mathcal{O}(\ell^4 \cdot m^2 \cdot |X|^3)$ states that computes $(\boldsymbol{x} < \ell) \to \varphi(\boldsymbol{x})$.*

Proof sketch. Fix a predicate φ and $\ell \in \mathbb{N}$. For every $2 \leq i < \ell$, let $\mathcal{P}_i$ be a protocol computing $(\varphi \mid i)$. We describe the protocol $\mathcal{P} = (Q, T, X, I, O)$ that computes $(\boldsymbol{x} \geq \ell) \vee \varphi(\boldsymbol{x}) \equiv (\boldsymbol{x} < \ell) \to \varphi(\boldsymbol{x})$. The input mapping I is the identity. During the computation, agents never forget their initial state – that is, all successor states of an agent are annotated with their initial state. The protocol initially performs a leader election. Each provisional leader stores how many agents it has "knocked out" during the leader election in a counter from 0 to $\ell - 1$. After increasing the counter to a given value $i < \ell$, it resets the state of i agents and itself to the corresponding initial state of $\mathcal{P}_{i+1}$, annotated with X, and initiates a simulation of $\mathcal{P}_{i+1}$. When the counter of an agent reaches $\ell - 1$, the agent knows that the population size must be $\geq \ell$, and turns the population into a permanent 1-consensus. Now, if the population size i is smaller than ℓ, then eventually a leader gets elected who resets the population to the initial population of $\mathcal{P}_i$. Since $\mathcal{P}_i$ computes $(\varphi \mid i)$, the simulation of $\mathcal{P}_i$ eventually yields the correct output. ◀

6.2 Computing boolean combinations of predicates for fixed-size inputs

We want to produce a population protocol $\mathcal{P}$ for a boolean combination φ of atomic predicates $(\varphi_i)_{i \in [k]}$ for which we have population protocols $(\mathcal{P}_i)_{i \in [k]}$. As in Section 5.3, we cannot use a standard "product protocol" that executes $\mathcal{P}_1, \ldots, \mathcal{P}_k$ synchronously because the number of states would be exponential in k. Instead, we want to simulate the *concatenation* of $(\mathcal{P}_i)_{i \in [k]}$. However, this is only possible if for all $i \in [k]$, the executions of $\mathcal{P}_i$ eventually "halt", i.e. some agents are eventually certain that the output of the protocol will not change anymore, which is not the case in general population protocols. For this reason we restrict our attention to "halting" protocols.

▶ **Definition 10.** *Let $\mathcal{P}$ be a simple protocol with output states* `f` *and* `t`*. We say that $\mathcal{P}$ is a* halting protocol *if every configuration C reachable from an initial configuration satisfies:*

- $C(\mathtt{f}) = 0 \vee C(\mathtt{t}) = 0$,
- $C \xrightarrow{*} C' \wedge C(q) > 0 \Rightarrow C'(q) > 0$ *for every $q \in \{\mathtt{f}, \mathtt{t}\}$ and every configuration C'.*

Intuitively, a halting protocol is a simple protocol in which states `f` and `t` behave like "final states": If an agent reaches $q \in \{\mathtt{f}, \mathtt{t}\}$, then the agent stays in q forever. In other words, the protocol reaches consensus 0 (resp. 1) iff an agent ever reaches `f` (resp. `t`).

▶ **Theorem 11.** *Let $k, i \in \mathbb{N}$. Let φ be a boolean combination of atomic predicates $(\varphi_j)_{j \in [k]}$. Assume that for every $j \in [k]$, there is a simple halting protocol $\mathcal{P}_j = (Q_j, L_j, X, T_j, I_j, O_j)$ with one leader computing $(\varphi_j \mid i)$. Then there exists a simple halting protocol $\mathcal{P}$ that computes $(\varphi \mid i)$, with one leader and $\mathcal{O}(|X| \cdot (\mathrm{len}(\varphi) + |Q_1| + \ldots + |Q_k|))$ states.*

Proof sketch. We only sketch the construction for $\varphi = \varphi_1 \wedge \varphi_2$. The main intuition is that, since $\mathcal{P}_1$ and $\mathcal{P}_2$ are halting, we can construct a protocol that, given an input $\boldsymbol{v}$, first simulates $\mathcal{P}_1$ on $\boldsymbol{v}$, and, after $\mathcal{P}_1$ halts, either halts if $\mathcal{P}_1$ converges to 0, or simulates $\mathcal{P}_2$ on $\boldsymbol{v}$ if $\mathcal{P}_1$ converges to 1. Each agent remembers in its state the input variable it corresponds to, in order to simulate $\mathcal{P}_2$ on $\boldsymbol{v}$. ◀

6.3 Computing atomic predicates for fixed-size inputs

We describe a halting protocol that computes a given threshold predicate for fixed-size inputs.

▶ **Theorem 12.** *Let $\varphi(\boldsymbol{x}, \boldsymbol{y}) \stackrel{def}{=} \boldsymbol{\alpha} \cdot \boldsymbol{x} - \boldsymbol{\beta} \cdot \boldsymbol{y} > 0$. For every $i \in \mathbb{N}$, there exists a halting protocol with one leader and $\mathcal{O}(i^2(|\varphi| + \log i)^3)$ states that computes $(\varphi \mid i)$.*

We first describe a sequential algorithm Greater-Sum$(\boldsymbol{x}, \boldsymbol{y})$, that for every input $\boldsymbol{x}, \boldsymbol{y}$ satisfying $|\boldsymbol{x}| + |\boldsymbol{y}| = i$ decides whether $\boldsymbol{\alpha} \cdot \boldsymbol{x} - \boldsymbol{\beta} \cdot \boldsymbol{y} > 0$ holds. Then we simulate Greater-Sum by means of a halting protocol with i agents.

Since each agent can only have $\mathcal{O}(\log i + \log |\varphi|)$ bits of memory (the logarithm of the number of states), Greater-Sum must use at most $\mathcal{O}(i \cdot (\log i + \log |\varphi|))$ bits of memory, otherwise it cannot be simulated by the agents. Because of this requirement, Greater-Sum cannot just compute, store, and then compare $\boldsymbol{\alpha} \cdot \boldsymbol{x}$ and $\boldsymbol{\beta} \cdot \boldsymbol{y}$; this uses too much memory.

Greater-Sum calls procedures $Probe_1(j)$ and $Probe_2(j)$ that return the j-th bits of $\boldsymbol{\alpha x}$ and $\boldsymbol{\beta y}$, respectively, where $j = 1$ is the most significant bit. Since $|\boldsymbol{x}| \leq i$, and the largest constant in $\boldsymbol{\alpha}$ is at most $||\varphi||$, we have $\boldsymbol{\alpha} \cdot \boldsymbol{x} \leq i \cdot ||\varphi||$, and so $\boldsymbol{\alpha} \cdot \boldsymbol{x}$ has at most $m \stackrel{\text{def}}{=} |\varphi| + \lfloor \log(i) \rfloor + 1$ bits, and the same holds for $\boldsymbol{\beta y}$. So we have $1 \leq j \leq m$. Let us first describe Greater-Sum, and then $Probe_1(j)$; the procedure $Probe_2(j)$ is similar.

Greater-Sum$(\boldsymbol{x}, \boldsymbol{y})$ loops through $j = 1, \ldots, m$. For each j, it calls $Probe_1(j)$ and $Probe_2(j)$. If $Probe_1(j) > Probe_2(j)$, then it answers $\varphi(\boldsymbol{x}, \boldsymbol{y}) = 1$, otherwise it moves to $j + 1$. If Greater-Sum reaches the end of the loop, then it answers $\varphi(\boldsymbol{x}, \boldsymbol{y}) = 0$. Observe that Greater-Sum only needs to store the current value of j and the bits returned by $Probe_1(j)$ and $Probe_2(j)$. Since $j \leq m$, Greater-Sum only needs $\mathcal{O}(\log(|\varphi| + \log i))$ bits of memory.

$Probe_1(j)$ uses a decreasing counter $k = m, \ldots, j$ to successively compute the bits $b_1(k)$ of $\boldsymbol{\alpha} \cdot \boldsymbol{x}$, starting at the least significant bit. To compute $b_1(k)$, the procedure stores the carry $c_k \leq i$ of the computation of $b_1(k+1)$; it then computes the sum $s_k := c_k + \boldsymbol{\alpha}(k) \cdot \boldsymbol{x}$ (where $\boldsymbol{\alpha}(k)$ is the k-th vector of bits of $\boldsymbol{\alpha}$), and sets $b_k := s_k \bmod 2$ and $c_{k-1} := s_k \div 2$. The procedure needs $\mathcal{O}(\log(|\varphi| + \log i))$ bits of memory for counter k, $\log(i) + 1$ bits for encoding s_k, and $\mathcal{O}(\log(i))$ bits for encoding c_k. So it only uses $\mathcal{O}(\log(|\varphi| + \log i))$ bits of memory.

Let us now simulate Greater-Sum$(\boldsymbol{x}, \boldsymbol{y})$ by a halting protocol with one leader agent. Intuitively, the protocol proceeds in rounds corresponding to the counter k. The leader stores in its state the value j and the current values of the program counter, of counter k, and of variables b_k, s_k, and c_k. The crucial part is the implementation of the instruction $s_k := c_k + \boldsymbol{\alpha}(k) \cdot \boldsymbol{x}$ of $Probe_1(j)$. In each round, the leader adds input agents one by one. As the protocol only needs to work for populations with i agents, it is possible for each agent to know if it already interacted with the leader in this round, and for the leader to count the number of agents it has interacted with this round, until it reaches i to start the next round.

7 Conclusion and further work

We have proved that every predicate φ of quantifier-free Presburger arithmetic (QFPA) is computed by a leaderless protocol with poly$(|\varphi|)$ states. Further, the protocol can be computed in polynomial time. The number of states of previous constructions was exponential

both in the bit-length of the coefficients of φ, and in the number of occurrences of boolean connectives. Since QFPA and PA have the same expressive power, every computable predicate has a succinct leaderless protocol. This result completes the work initiated in [8], which also constructed succinct protocols, but only for some predicates, and with the help of leaders.

It is known that protocols with leaders can be exponentially faster than leaderless protocols. Indeed, every QFPA predicate is computed by a protocol with leaders whose expected time to consensus is polylogarithmic in the number of agents [6], while every leaderless protocol for the majority predicate needs at least linear time in the number of agents [1]. Our result shows that, if there is also an exponential gap in state-complexity, then it must be because some family of predicates have protocols with leaders of logarithmic size, while all leaderless families need polynomially many states. The existence of such a family is an open problem.

The question of whether protocols with $\text{poly}(|\varphi|)$ states exist for every PA formula φ, possibly with quantifiers, also remains open. However, it is easy to prove that no algorithm for the construction of protocols from PA formulas runs in time $2^{p(n)}$ for any polynomial p.

▶ **Theorem 13.** *For every polynomial p, every algorithm that accepts a formula φ of PA as input, and returns a population protocol computing φ, runs in time $2^{\omega(p(|\varphi|))}$.*

Therefore, if PA also has succinct protocols, then they are very hard to find.

Our succinct protocols for QFPA have slow convergence (in the usual parallel time model, see e.g. [2]), since they often rely on exhaustive exploration of a number of alternatives, until the right one is eventually hit. The question of whether every QFPA predicate has a succinct *and* fast protocol is very challenging, and we leave it open for future research.

References

1 Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. Time-space trade-offs in population protocols. In *Proc. Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2560–2579. SIAM, 2017.

2 Dan Alistarh and Rati Gelashvili. Recent algorithmic advances in population protocols. *SIGACT News*, 49(3):63–73, 2018. doi:10.1145/3289137.3289150.

3 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *Proc. 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 290–299, 2004. doi:10.1145/1011767.1011810.

4 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

5 Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *Proc. 25th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 292–299, 2006. doi:10.1145/1146381.1146425.

6 Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21(3):183–199, 2008.

7 Michael Blondin, Javier Esparza, Blaise Genest, Martin Helfrich, and Stefan Jaax. Succinct population protocols for Presburger arithmetic, 2019. arXiv:1910.04600.

8 Michael Blondin, Javier Esparza, and Stefan Jaax. Large flocks of small birds: On the minimal size of population protocols. In *Proc. 35th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 16:1–16:14, 2018. doi:10.4230/LIPIcs.STACS.2018.16.

9 David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. *Distributed Computing*, 31(4):257–271, 2018.

10 Robert Elsässer and Tomasz Radzik. Recent results in population protocols for exact majority and leader election. *Bulletin of the EATCS*, 126, 2018.

11 Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. *Acta Informatica*, 54(2):191–215, 2017. doi:10.1007/s00236-016-0272-3.

12 Christoph Haase. A survival guide to Presburger arithmetic. *SIGLOG News*, 5(3):67–82, 2018.

13 Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. *Comptes rendus du I^{er} Congrès des mathématiciens des pays slaves*, pages 192–201, 1929.

14 David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *Natural Computing*, 7(4):615–633, 2008.

A Sub-Quadratic Algorithm for the Longest Common Increasing Subsequence Problem

Lech Duraj
Theoretical Computer Science, Faculty of Mathematics and Computer Science,
Jagiellonian University, Kraków, Poland
duraj@tcs.uj.edu.pl

Abstract

The Longest Common Increasing Subsequence problem (LCIS) is a natural variant of the celebrated Longest Common Subsequence (LCS) problem. For LCIS, as well as for LCS, there is an $\mathcal{O}\left(n^2\right)$-time algorithm and a SETH-based conditional lower bound of $\mathcal{O}\left(n^{2-\varepsilon}\right)$. For LCS, there is also the Masek-Paterson $\mathcal{O}\left(n^2/\log n\right)$-time algorithm, which does not seem to adapt to LCIS in any obvious way. Hence, a natural question arises: does any (slightly) sub-quadratic algorithm exist for the Longest Common Increasing Subsequence problem? We answer this question positively, presenting a $\mathcal{O}\left(n^2/\log^a n\right)$-time algorithm for $a = \frac{1}{6} - o(1)$. The algorithm is not based on memorizing small chunks of data (often used for logarithmic speedups, including the "Four Russians Trick" in LCS), but rather utilizes a new technique, bounding the number of significant symbol matches between the two sequences.

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

Keywords and phrases longest common increasing subsequence, log-shaving, matching pairs

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.41

Related Version A full version of the paper is available at `https://arxiv.org/abs/1902.06864`.

Funding *Lech Duraj*: This work was partially supported by Polish National Science Center grant 2016/21/B/ST6/ 02165.

Acknowledgements I would like to thank Grzegorz Guśpiel, Grzegorz Herman and Adam Polak for helpful discussions and proofreading the paper, as well as the reviewers for many insightful comments and suggestions.

1 Introduction

The Longest Common Increasing Subsequence problem (LCIS) is a variant of the well-known and extensively studied Longest Common Sequence (LCS) problem. The LCS is formulated as follows: given two integer sequences $A = (A[1], \ldots, A[n])$ and $B = (B[1], \ldots, B[n])$, determine another sequence C which is a subsequence of both A and B, of maximal possible length. In the LCIS variant, we require C to be a strictly increasing subsequence.

For LCS, a simple algorithm working in $\mathcal{O}\left(n^2\right)$-time was published in 1974 by Wagner and Fischer [27]. The complexity was later brought down to $\mathcal{O}\left(\frac{n^2}{\log n}\right)$ (for constant alphabet size) by Masek and Paterson, using a technique informally called the "Four Russians trick" [21]. Some improvements have been made since then (in particular, [14] shaves another logarithm, down to $\mathcal{O}\left(\frac{n^2 \log\log n}{\log^2 n}\right)$ even with arbitrary alphabet size), but no truly sub-quadratic, $\mathcal{O}\left(n^{2-\epsilon}\right)$-time algorithm has been found. There is even substantial evidence that a better algorithm might in fact not exist: it was shown by Abboud, Backurs and Vassilevska-Williams [1], as well as by Bringmann and Künnemann [8] that a truly sub-quadratic algorithm for LCS would yield a $2^{\delta n}$-time algorithm for SAT, with some $\delta < 1$, thus refuting the Strong Exponential Time Hypothesis (which states, roughly speaking, that such an algorithm is

impossible [16, 17]). Hence, if we believe that SETH is true, then we must accept that no fast algorithms for LCS will ever be found. It is worth noting that in recent years several other SETH-based quadratic-time bounds were also shown, e.g., [6] and [24].

As for LCIS, it is arguably one of the most interesting variants of LCS: neither of these problems seems to be reducible to the other (unless we count the reduction of LCIS to 3-sequence LCS [18], which does not seem strong enough to have meaningful consequences). Therefore no algorithm or hardness result for LCS can be easily translated to a corresponding result for LCIS. The "obvious" dynamic programming algorithm for LCIS is $\mathcal{O}\left(n^3\right)$, the first $\mathcal{O}\left(n^2\right)$-time algorithm was given in [31], and possibly the simplest one was explicitly stated in [32]. A conditional lower bound was proven in [13]: it turns out that, as for LCS, any $\mathcal{O}\left(n^{2-\varepsilon}\right)$-time algorithm for LCIS would refute the Strong Exponential Time Hypothesis. The proof is based, like the one in [1], on a reduction from the Orthogonal Vectors problem (introduced in [28]), but the reduction itself needs a quite different gadget construction. It is also worth mentioning that the problem of Longest Common Weakly Increasing Subsequence, similar to LCIS but with only weak monotonicity required, also has a conditional quadratic lower bound [23]. LCWIS, unlike LCIS, is also non-trivial when restricted to constant-size alphabets [19, 12]. It still remains an open question whether LCWIS admits a sub-quadratic algorithm for any alphabet size greater than 3.

The LCIS problem itself has been studied quite extensively, and other algorithms have been proposed: Sakai [25] found an algorithm which can retrieve the LCIS in linear space, Kutz et al. [19] presented an algorithm that works in (roughly speaking) $\mathcal{O}\left(n \cdot d\right)$ time, where d is the output size (i.e. the length of LCIS). Chan et al. [11] proved that LCIS can be found in $\mathcal{O}\left(r \log\log n\right)$, where r is the number of *matching pairs* of symbols (i.e. the pairs (x, y) with $A[x] = B[y]$). These algorithms work much faster for some specific cases (for example, they are sub-quadratic for "random" inputs with a reasonable notion of "randomness"), but no algorithm that achieves $o\left(n^2\right)$ worst-case complexity has been given so far. Arguably, one of the reasons is that the "Four Russians Trick" does not seem to adapt to current dynamic-programming LCIS algorithms – at least, not in any easy way. In light of known conditional lower bound of this problem, we can only hope for complexity similar to $\mathcal{O}\left(\frac{n^2}{\log^a n}\right)$ for some $a > 0$, but achieving this would seem interesting enough. "The Art of Shaving Logs", as called by Timothy M. Chan [9], has already been practised for a variety of problems [7, 10, 29, 20, 15, 30], sometimes yielding surprising results – for example, some remarkable consequences in circuit complexity [3, 2]. Therefore, it appears natural to ask the question: *Is there any slightly sub-quadratic (i.e. $o\left(n^2\right)$-time) algorithm for LCIS?*

This paper gives a positive answer to this question, by presenting an $\mathcal{O}\left(\frac{n^2(\log\log n)^2}{\log^{1/6} n}\right)$-time algorithm for LCIS. Our algorithm iterates over matching pairs of symbols (as the one in [11]), but to achieve sub-quadratic time, a new „log-shaving" technique is introduced: we do not try to precompute the results for small chunks of data, as in LCS algorithms. Instead, we choose a useful subset of matching pairs – so-called *significant pairs* – prove that there are $o\left(n^2\right)$ such pairs, and adapt the algorithm to exploit this fact.

2 Basic notions and paper outline

Let A and B be the input sequences – for most of the paper, it is convenient to allow A and B to have different lengths. Later, for the final complexity results, we will assume $|A| = |B|$. We use array-like notation for elements of A and B, i.e. $A = (A[1], A[2], \ldots)$, $B = (B[1], B[2], \ldots)$. We will refer to the elements of A and B as *symbols*, remembering that

the symbols are in fact integers, and thus can be compared with each other. Also, we may assume that all those integers are positive and not exceeding $\mathcal{O}(|A| + |B|)$ – if not, we can rename all the elements to be in range $\{1, 2, \ldots, |A| + |B|\}$, while preserving their relative order.

▶ **Definition 1** (Matching pair). *A pair of indices (x, y) for some $1 \leq x \leq |A|$, $1 \leq y \leq |B|$ is a* matching pair *if $A[x] = B[y]$. For $\sigma = A[x] = B[y]$, we can say that (x, y) is a σ-matching pair, or simply a σ-pair. We also say that σ is the symbol of (x, y) and sometimes write $\sigma = symbol(x, y)$.*

▶ **Definition 2** (Orders on pairs). *Let (x, y) and (x', y') be matching pairs.*

(1) *We say that $(x, y) \leq (x', y')$ if $x \leq x'$ and $y \leq y'$.*

(2) *We say that $(x, y) \prec (x', y')$ if $x < x'$, $y < y'$ and $symbol(x, y) < symbol(x', y')$.*

▶ **Definition 3** (Common increasing subsequence). *A common increasing sequence of A and B is a sequence of matching pairs $(x_1, y_1), \ldots, (x_s, y_s)$ such that $(x_1, y_1) \prec \ldots \prec (x_s, y_s)$.*

Our main problem is to find the longest possible common increasing subsequence. Sometimes, we wish to consider only some prefixes of A and B, for which we will need the following two definitions:

▶ **Definition 4.** *For any $x \leq |A|$ and $y \leq |B|$, we define $lcis(x, y)$ as the maximal possible length of a common increasing subsequence that ends with some (x', y') with $x' \leq x$ and $y' \leq y$. In other words $lcis(x, y)$ is the length of the longest common increasing subsequence of $A[1..x]$ and $B[1..y]$.*

▶ **Definition 5.** *For any matching pair (x, y), we define $lcis^{\rightarrow}(x, y)$ as the maximal possible length of a common increasing subsequence that ends with (x, y).*

▶ Remark 6. The value of $lcis(x, y)$ is equal to $\max_{(x',y') \leq (x,y)} lcis^{\rightarrow}(x', y')$. In particular, for any $(x', y') \leq (x, y)$ we have $lcis^{\rightarrow}(x', y') \leq lcis(x, y)$.

A sequence realizing $lcis^{\rightarrow}(x, y)$ must have some pair (x', y') as the next-to-last element (providing that $lcis^{\rightarrow}(x, y) \geq 2$). Clearly, $lcis^{\rightarrow}(x', y') = lcis^{\rightarrow}(x, y) - 1$. We call such a pair the *predecessor* of (x, y). There may be multiple candidates for the predecessor, so we break the ties first by y, then by x. Formally:

▶ **Definition 7** (Predecessor). *For a matching pair (x, y) the predecessor $\pi(x, y)$ is a matching pair $(x', y') \prec (x, y)$ such that:*

(1) *$lcis^{\rightarrow}(x', y') = lcis^{\rightarrow}(x, y) - 1$,*

(2) *(x', y') has the minimal possible y' of all pairs satisfying (1),*

(3) *(x', y') has the minimal possible x' of all pairs satisfying (1) and (2).*

An example is shown in Figure 1 below:

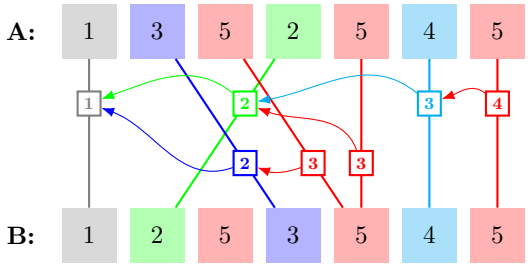


Figure 1 An example: for two sequences $A = (1, 3, 5, 2, 5, 4, 5)$ and $B = (1, 2, 5, 3, 5, 4, 5)$ some matching pairs are shown. A pair (x, y) is labeled with $lcis^{\rightarrow}(x, y)$ and an arrow leads from (x, y) to $\pi(x, y)$. Some pairs were omitted for clarity.

▶ **Definition 8.** *For a matching pair (x, y) with $lcis^{\rightarrow}(x, y) > k$, the k-th predecessor $\pi^k(x, y)$ is defined inductively as (x, y) for $k = 0$ and $\pi(\pi^{k-1}(x, y))$ for $k \geq 1$. In particular, $\pi^1(x, y) = \pi(x, y)$.*

The algorithm for LCIS in [11] iterates over all matching pairs in A and B. There may be, however, as many as $\Theta(n^2)$ of them – it is easy to construct an example of such sequences by including a lot of equal elements. Observe, though, that some of the matching pairs may not really matter in the solution – for example, if $A[x+1] = A[x]$, then a matching pair $(x+1, y)$ for any y is as good as (x, y), and we could drop $A[x+1]$ from A altogether. We generalize this observation to form the notion of a *significant pair*, which is the central concept of this paper, allowing us to construct the desired faster algorithm for LCIS.

▶ **Definition 9** (Significant pair). *Let (x, y) be a σ-pair, i.e. $\sigma = A[x] = B[y]$. We say that (x, y) is a significant pair if for every σ-pair $(x', y') \leq (x, y)$, if $(x', y') \neq (x, y)$ then $lcis^{\rightarrow}(x', y') < lcis^{\rightarrow}(x, y)$.*

Again, we include an example to make this important definition more clear:

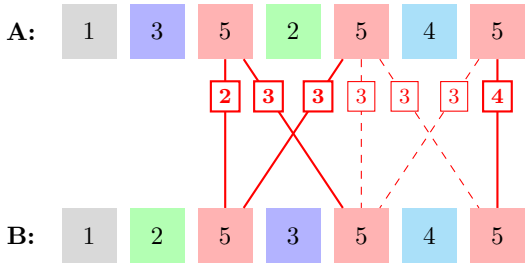


Figure 2 An example of two sequences A and B with some matching pairs. A pair (x, y) is labeled with $lcis^{\rightarrow}(x, y)$; the significant pairs are drawn with solid lines, while the insignificant ones – with dashed lines. Some pairs were ommitted for clarity.

▷ Claim 10. If (x, y) is a matching pair, then $\pi(x, y)$, if exists, is a significant pair.

Proof. Easy from the tie-breaking rule in the predecessor definition: let $(x', y') = \pi(x, y)$. If (x', y') is not significant, then there is a better candidate for the predecessor. ◁

Having defined the significant pairs, we propose the following two theorems, which together form our main result. The first bounds the number of such pairs, the second proposes an algorithm that exploits this bound:

▶ **Theorem 11.** *For any A, B with $|A|, |B| \leq n$, the number of significant pairs is at most $\mathcal{O}\left(\frac{n^2}{\log^{1/3} n}\right)$.*

▶ **Theorem 12.** *Suppose that $|A|, |B| \leq n$ and that there are at most $\mathcal{O}\left(\frac{n^2}{t}\right)$ significant pairs, with $t = t(n)$ satisfying $\log t = \Theta(\log \log n)$. There is an algorithm which finds LCIS in $\mathcal{O}\left(\frac{n^2 (\log \log n)^2}{\sqrt{t}}\right)$ time complexity.*

The obvious consequence of Theorems 11 and 12 is the following:

▶ **Corollary 13.** *There is an algorithm which finds LCIS of two sequences A, B with $|A|, |B| \leq n$ in $\mathcal{O}\left(\frac{n^2 (\log \log n)^2}{\log^{1/6} n}\right)$ time complexity.*

The rest of the paper is devoted to proving the two main theorems: in Section 3 we prove Theorem 11, whereas Section 4 describes the algorithm of Theorem 12.

3 Counting significant pairs

3.1 The idea

In this section we present the high-level idea behind the bound for the number of significant pairs. Please note that while the proof originally stems from this concept, its final version needs also some careful counting and balancing arguments, as well as a few non-intuitive tricks. Therefore, we start with an informal sketch to give some intuitions, while the full, formal proof will be presented in Sections 3.2-3.4.

Imagine two sequences A and B with $|A| = |B| = n$ and with the number of significant pairs „very close" to $\Theta\left(n^2\right)$. This requires at least one symbol σ generating a lot of significant σ-pairs itself (as opposed to, for example, $\sqrt{n}$ different symbols generating $\Theta\left(n^{3/2}\right)$ pairs each – this is impossible, as it would imply $|A| > n$ or $|B| > n$). We then focus on one particular such symbol σ and imagine a graph G_σ with all occurrences of σ in A and B as vertices of G_σ, and all significant σ-pairs as its edges. (An example is already provided with Figure 2 – for $\sigma = 5$, the red elements of A and B are the vertices of G_σ, and red solid lines are the edges).

Denote by $s = s(n)$ the largest integer such that G_σ has at least s vertices of degree at least s – the total number of edges cannot exceed $\mathcal{O}(n \cdot s)$, so simplifying a little bit, our goal is to show that $s = o(n)$. But every σ-pair must have its predecessors, which are τ-pairs for some other symbols $\tau < \sigma$. The proof is based on the observation that these predecessors need quite a lot of different symbols – we argue that the total number of required elements of A and B is asymptotically greater than s. This forces $s = o(n)$, and after careful calculations we obtain more specific bounds.

To take a closer (but still preliminary) look at our main tools, consider a vertex $A[x]$ in G_σ with edges $(x, y_1), \ldots, (x, y_s)$ with $y_1 > y_2 > \ldots > y_s$. Consider, for a fixed $k > 0$, all predecessors $\pi^k(x, y_i)$ for $1 \leq i \leq s$ (let us not worry, for the moment, whether the predecessors exist), denoting $\pi^k(x, y_i) = (u_i, v_i)$. We claim that for every i, $v_{i+k} < v_i$. This is because $v_{i+k} < y_{i+k}$, while $v_i < y_{i+k}$ would lead to $(u_i, v_i) \prec (x, y_{i+k})$, which would in turn yield $lcis^{\rightarrow}(x, y_{i+k}) \geq lcis^{\rightarrow}(u_i, v_i) + 1 = lcis^{\rightarrow}(x, y_i) - k + 1$. But the values $lcis^{\rightarrow}(x, y_i), \ldots, lcis^{\rightarrow}(x, y_{i+k})$ must all be different (as the pairs are significant), which

implies $lcis^{\rightarrow}(x, y_{i+k}) \leq lcis^{\rightarrow}(x, y_i) - k$ – a contradiction. Therefore $v_i > y_{i+k} > v_{i+k}$. This shows that there must be at least $\frac{s}{k}$ different elements in B to accommodate the k-th predecessors of the pairs (x, y_i) for $i = 1, 2, \ldots, s$. A notable edge case is that for every k, all those predecessors happen to use the same symbol τ_k, implying at least $\frac{s}{k}$ occurrences of τ_k in B and thus at least $s + s/2 + s/3 + \ldots = \Omega(s \log s)$ symbols in B, which would immediately imply $s = \mathcal{O}(n/\log n)$. Of course, we cannot hope to be that fortunate, but we can salvage some bounds from this argument: if, for some $\delta = \delta(n)$, which is $\omega(1)$ and $o(\log n)$, and for any particular $x \in A$, the k-th predecessors use at most δ different symbols for every k, we can prove that this yields $\Omega\left(\frac{s \log s}{\delta}\right)$ symbols in B, so $s = \mathcal{O}\left(\frac{n\delta}{\log n}\right) = o(n)$ and we are done – this is formally proven in more general form in Section 3.4. On the other hand, if there are more than δ different symbols among predecessors for every possible x, we expect δ different elements before x in A. Using similar arguments as before, we argue that those sets of elements must be (at least partly) disjoint for different picks of x. But there are $\Omega(s)$ possible choices of a high-degree vertex x, which also implies $s = o(n)$ – this sketch roughly corresponds to Section 3.3.

We now move on to the full proof. Before presenting the core observations, we start with padding the sequences – adding dummy elements to make computing predecessors easier.

3.2 Preliminaries – padding the sequences

Consider two sequences A, B with $|A|, |B| \leq n$. Suppose that $lcis(|A|, |B|) \geq 1$ – if there are no common elements in A and B, there is nothing to be proven. We can also assume $n \geq 2$. For the sake of analyzing significant pairs between A and B, we shall modify the sequences a little bit. First, we can assume, without loss of generality, that A and B contain only positive integers (adding any constant to all the elements does not change anything). Then we pad both the sequences, inserting a prefix of dummy elements $P_n = (-2n, -2n+1, -2n+2, \ldots, -1)$ in the front, obtaining new sequences $\hat{A}$ and $\hat{B}$. More precisely, we put $\hat{A} = P_n \circ A$ and $\hat{B} = P_n \circ B$, where $\circ$ is the operator of sequence concatenation. These new elements now contribute to all previous common increasing subsequences, increasing their lengths by exactly $2n$. This does not change the significance of any "old" matching pairs, i.e. any significant pair (x, y) present in A and B remains a significant pair $(2n + x, 2n + y)$ in $\hat{A}$ and $\hat{B}$. Therefore the number of significant pairs can only increase in this operation, so it is enough to prove the inequality of Theorem 11 for $\hat{A}$ and $\hat{B}$. The padding operation also ensures that for every matching pair (x, y) with $x, y > 2n$ we have $lcis^{\rightarrow}(x, y) > 2n$, allowing us to compute up to $2n$ predecessors of (x, y).

3.3 The σ-pair graph

First, let $\delta = \delta(n) = \frac{\log^{1/3} n}{4}$ – our goal is to bound the number of significant pairs by $\mathcal{O}\left(\frac{n^2}{\delta(n)}\right)$, and the order of magnitude of δ is chosen as „the highest one for which the proof still holds".

The crucial step of the proof starts with fixing a symbol σ and bounding only the number of significant σ-pairs, which we will then sum up over all possible σ. Without loss of generality we remove – for a while – all elements greater than σ from $\hat{A}$ and $\hat{B}$, as they do not affect the significance of any σ-pair.

Let A_σ (resp. B_σ) be the set of all positions x with $\hat{A}[x] = \sigma$ (resp. $\hat{B}[x] = \sigma$). Consider a bipartite graph G_σ with the set of vertices $V(G_\sigma) = A_\sigma \cup B_\sigma$, and the edge set $E(G_\sigma) = \{(x, y) \in A_\sigma \times B_\sigma : (x, y) \text{ is a significant pair}\}$. Our main goal is to bound the number of

edges in G_σ. To do that, we first define some family of „bad" configurations of edges and prove that if any of them is forbidden, the number of edges can be bounded as desired. Finally – which is the most technical part – we show that at least one of those configurations does not, indeed, appear in the graph.

For any $1 \leq x \leq |\hat{A}|$, let us denote by $A_x^{\rightarrow}$ the suffix of $\hat{A}$ starting at x (including x). For an integer k we say that $A_x^{\rightarrow}$ is a *k-dense suffix*, if:

- $|A_x^{\rightarrow}| \leq \lceil k \cdot \delta \rceil$,
- There are $\lfloor n/\delta \rfloor$ distinct edges $(x, c_1), \ldots, (x, c_{\lfloor n/\delta \rfloor}) \in G_\sigma$,
- Every c_i has, in turn, k distinct edges (y_{ij}, c_i) for $j = 1, 2, \ldots, k$ and some $y_{ij} \in A_x^{\rightarrow}$.

The following lemma is the core idea of the proof, as it forbids at least one dense suffix to appear in G_σ. As its proof needs careful analysis (boiling down to counting predecessors of σ-pairs), and is somewhat technical, we defer the proof until Section 3.4. Before that, we will use Lemma 14 to show our ultimate goal, Theorem 11.

▶ **Lemma 14.** *For every A and B with $|A|, |B| \leq n$, and for the corresponding graph G_σ, there exists some positive integer $k \leq n/\delta$ such that there is no k-dense suffix in $\hat{A}$.*

To prove that this lemma bounds the number of edges in G_σ, we first split vertices of $\hat{A}$ according to their degree. The *small* vertices are these with degree at most $\frac{2n}{\delta}$, the rest being *large* vertices. The following observation is straightforward:

▶ Remark 15. The total number of edges incident to small vertices of $\hat{A}$ is at most $|A_\sigma| \cdot \frac{2n}{\delta}$.

It remains to bound the number of edges incident to large vertices in $\hat{A}$. For every connected substring $S \subseteq \hat{A}$, let us denote by $L(S)$ the number of such edges between S and $\hat{B}$.

▶ **Lemma 16.** *For every A and B with $|A|, |B| \leq n$, $L(\hat{A}) \leq |B_\sigma| \cdot \frac{2|\hat{A}|}{\delta}$.*

Proof. We use induction on $|\hat{A}|$ (please note that n remains fixed throughout the proof, so $|\hat{A}|$ is always equal to $|A| + 2n$). The minimal length of a padded sequence is $|\hat{A}| = 2n + 1$ – i.e. with only one non-dummy element – and in this case we have $|E| \leq |B_\sigma|$. Suppose now that we have some $\hat{A}$ with $|\hat{A}| = a$, and have already proven the statement for all A' with $|A'| < a$.

Let k be the integer obtained from Lemma 14 and let Y be the suffix of of $\hat{A}$ of length $\lceil k\delta \rceil$ (as $\lceil k\delta \rceil \leq n$, Y is a proper suffix). We can now use Lemma 14 to bound the number of edges between Y and $\hat{B}$. Let us initially place k tokens on every σ-element of $\hat{B}$ and consider all elements of Y, starting from the last one. For every large vertex $x \in Y$ we look at all its neighbors and remove one token from each of them, whenever they still have one. We claim that every time, at least half of these neighbours (which is at least n/δ, as we are dealing with large vertices) must still have a token to spare. This is because if at least n/δ neighbors of x were already tokenless, than $A_x^{\rightarrow}$ would be a k-dense suffix: clearly $|A_x^{\rightarrow}| \leq \lceil k\delta \rceil$, and we have just found that x has $\lfloor n/\delta \rfloor$ neighbors which have already lost their k tokens, so each of them has k neighbours in $A_x^{\rightarrow}$.

Therefore, every x must be able to take a token from at least half of its neighbors. As there are only $k \cdot |B_\sigma|$ tokens to be removed, the total number of edges $L(Y)$ cannot exceed $2k|B_\sigma| \leq 2 \cdot \frac{\lceil k\delta \rceil}{\delta} \cdot |B_\sigma| = |Y| \cdot \frac{2|B_\sigma|}{\delta}$.

Let us denote by $\hat{A} - Y$ the prefix of $\hat{A}$ obtained by deleting Y from the end of $\hat{A}$ (i.e. the prefix of length $|\hat{A}| - |Y|$). Now if $|\hat{A} - Y| \leq 2n$ (i.e. Y uses up all non-padding symbols), then $L(\hat{A} - Y) = 0$. Otherwise, we can apply the induction hypothesis to $\hat{A} - Y$, obtaining $L(\hat{A} - Y) \leq |\hat{A} - Y| \cdot \frac{2|B_\sigma|}{\delta}$. In both cases we have $L(\hat{A}) = L(\hat{A} - Y) + L(Y) \leq |\hat{A}| \cdot \frac{2|B_\sigma|}{\delta}$, as desired. ◀

We can now prove Theorem 11 and bound the total number of significant pairs:

Proof of Theorem 11. We want to show that for any A, B with $|A|, |B| \le n$, the number of significant pairs is at most $\mathcal{O}\left(\frac{n^2}{\log^{1/3} n}\right)$. We already know that is enough to prove it for padded sequences $\hat{A}$ and $\hat{B}$. For a fixed σ, the number of significant σ-pairs is, from Remark 15 and Lemma 16, at most $|A_\sigma| \cdot \frac{2n}{\delta} + |B_\sigma| \cdot \frac{2|\hat{A}|}{\delta} \le (|A_\sigma| + |B_\sigma|) \cdot \frac{6n}{\delta}$. Summing this over all possible symbols σ (and using the fact that $\sum_\sigma (|A_\sigma| + |B_\sigma|) = |\hat{A}| + |\hat{B}| \le 6n$), we get that the total number of significant pairs does not exceed $\frac{36n^2}{\delta} \le \frac{144n^2}{\log^{1/3} n}$. ◀

To close this section, it may be worth asking if our bound of $\mathcal{O}\left(\frac{n^2}{\log^{1/3} n}\right)$ significant pairs is tight, or at least close to the optimal one. We partially answer that in the full version of the paper, providing an example of two sequences with $\Omega\left(\frac{n^2}{\log n}\right)$ significant pairs. Hence, we cannot go lower than this bound, but there is still a gap for possible future work.

3.4 Dense suffixes and the predecessor matrix

In this section we complete the missing part by proving Lemma 14. To do that, we need to introduce a new concept – the *predecessor matrix* M of significant pairs. To give some intuition what this matrix is, imagine that we first find the longest common increasing subsequence of $\hat{A}$ and $\hat{B}$ and put this sequence of significant pairs into the first column of M, one pair in every cell (starting from last element of LCIS, downwards). Then we delete some final elements of $\hat{B}$ such that the length of LCIS decreases by exactly 1, find the new (possibly very different) LCIS, and form M's second column the same way. We can repeat this process n times, and the padding of the sequences always allows us to compute n predecessors.

Each entry of M is a significant pair (x, y). We refer to $symbol(x, y)$ as *color* of this entry of M, as we feel this gives a better intuition. To analyze M, we look at the number of different colors in each row. We show that too few colors in every row would cause $\hat{B}$ to accumulate more than $3n$ elements – which is impossible – so there is a row (say, k-th) with somewhat more colors – we then prove that this row corresponds to the desired k fulfilling the statement of Lemma 14.

To formally define M, recall the previous assumptions: we have sequences $\hat{A}$ and $\hat{B}$ which are both padded with P_n and do not contain symbols greater than σ. Let $a = |\hat{A}|$, $b = |\hat{B}|$ and $\ell = lcis(a, b)$. Because of padding we know that $2n < a, b \le 3n$ and that $\ell > 2n$. It is easy to see that for every $y > 1$ we have $lcis(a, y-1) \ge lcis(a, y) - 1$. Therefore, if we iterate y downwards from b to 1, $lcis(a, y)$ takes all values between ℓ and 1. In particular, there must exist elements $b = b_1 > b_2 > \ldots > b_n$ such that:

$$lcis(a, b_j) = \ell - j + 1,$$

for every $j = 1, 2, \ldots, n$.

We define the *predecessor matrix* M as an $n \times n$ matrix of matching pairs. For $1 \le j \le n$ we consider the longest common increasing sequence realizing $lcis(a, b_j)$ and define $M[1, j]$ as its last element (x^*, y^*). If there are multiple possibilities, we pick the one with minimal y^* and then with minimal x^*. We then define $M[i, j] = \pi^{i-1}(M[1, j])$. In other words, below every pair in M we put its predecessor. Observe that the properties of predecessors immediately imply $lcis^{\rightarrow}(M[i, j]) = lcis^{\rightarrow}(M[1, j]) - i + 1 = \ell - i - j + 2$.

If we pick, instead of some b_j, another b'_j such that $lcis(a, b'_j) = lcis(a, b_j) = \ell - j + 1$, we will get exactly the same $M[1, j]$, and thus the same $M[i, j]$ for all $i = 1, 2, \ldots, n$ – this is because of the tie-breaker rule for the choice of $M[1, j]$. Thus, the matrix M does not depend on the choice of $b_1, \ldots, b_n$, but only on $\hat{A}$ and $\hat{B}$.

We begin with a technical lemma about M which will be useful later. This observation is a generalization of the idea introduced in Section 3.1 – if s different significant pairs are incident to a single vertex $x \in \hat{A}$, then among their i-th predecessors we expect at least about s/i distinct values.

▶ **Lemma 17.** *For some i, i', j, j' with $1 \leq j < j' \leq n$ and $1 \leq i, i' \leq n$, let $(x, y) = M[i, j]$ and $(x', y') = M[i', j']$. If $j' \geq j + i$, then $y' < y$.*

Proof. Suppose to the contrary that $y \leq y'$. As $y' \leq b_{j'}$ and a is the last element of $\hat{A}$, it would imply $(x, y) \leq (a, b_{j'})$, which would in turn yield $lcis(a, b_{j'}) \geq lcis^{\rightarrow}(x, y) = lcis^{\rightarrow}(M[i, j]) = lcis^{\rightarrow}(M[1, j]) - i + 1 = lcis(a, b_j) - i + 1$. But as $j' \geq j + i$, there must be $lcis(a, b_{j'}) \leq lcis(a, b_j) - i$. This contradiction shows $y' < y$. ◀

As stated before, we will refer to the symbols of $\hat{A}$ and $\hat{B}$ as *colors*, imagining that every entry (x, y) in M is painted with a color corresponding to $symbol(x, y)$, the (common) symbol of $\hat{A}[x]$ and $\hat{B}[y]$. In every column of M the colors are strictly decreasing, and thus different. Hence, no color can have more than n entries in M. It is also evident that two entries in M must correspond to different elements in $\hat{A}$ and $\hat{B}$ if they have different colors. The main lemma of this section states, roughly, that the rows of M do not contain too few colors:

▶ **Lemma 18.** *There is some $2 \leq k \leq n/\delta$ such that every submatrix of M consisting of some $\lceil \frac{n}{\delta} \rceil$ columns (not necessarily consecutive) and rows $1, \ldots, k-1$ uses at least $\lceil k\delta \rceil$ colors.*

Proof. Consider all k that are powers of 2: $k = 2^q$ for $1 \leq q \leq \lfloor \log n - \log \delta \rfloor$. Suppose, to the contrary, that for every such $k = 2^q$ we can find some $\lceil \frac{n}{\delta} \rceil$ columns $c_1, \ldots, c_{\lceil \frac{n}{\delta} \rceil}$ of M which have at most $\delta \cdot 2^q$ colors in total in rows $1, 2, \ldots, 2^q - 1$. From these columns c_i and the lower half of these rows $(2^{q-1}, \ldots, 2^q - 1)$ we form a submatrix M_q of M. These matrices are defined for $1 \leq q \leq \lfloor \log n - \log \delta \rfloor$ and have the following properties:

- they are disjoint submatrices of M (as every one takes different rows),
- for any q, the matrix M_q contains $2^{q-1} \cdot \lceil \frac{n}{\delta} \rceil$ pairs,
- for any q, the entries of M_q use at most $\delta \cdot 2^q$ colors between them.

Let a color be *q-strong*, if at least $\frac{n}{4\delta^2}$ entries in M_q are of that color. Observe that a particular color can be q-strong for at most $4\delta^2$ distinct values of q, otherwise – as all M_q's are disjoint – there would be more than n entries of that color in M, which is impossible.

For any q, the colors which are not q-strong can make up for at most half of entries in M_q (as there are $\delta \cdot 2^q$ colors in M_q, none of which can have more than $\frac{n}{4\delta^2}$ entries – a total of $\frac{n}{2\delta} \cdot 2^{q-1}$). Hence, there are at least $\frac{n2^{q-1}}{2\delta}$ pairs in M_q which have q-strong colors. Let us *mark* all these entries of M_q.

For any $t = 0, 1, \ldots, 2^q - 1$ let $\mathcal{M}_q(t)$ be the set of columns $M_q[\cdot, j]$ with $j \equiv t \mod 2^q$. For a fixed q, at least one of the sets $\mathcal{M}_q(t)$ must contain at least $\frac{n}{4\delta}$ marked entries, as there are $\frac{n2^{q-1}}{2\delta}$ marked entries in M_q split between 2^q sets. But we can show that if (x, y) and (x', y') are two different pairs in some $\mathcal{M}_q(t)$, then $y \neq y'$. Indeed, both pairs are either in the same column – which makes them have different colors and thus no common elements – or at least 2^q columns apart, in which case $y \neq y'$ because of Lemma 17. This in turn means that for every q there is a set B_q of at least $\frac{n}{4\delta}$ distinct elements of $\hat{B}$, each of a q-strong color. A color can be q-strong for at most $4\delta^2$ values of q, so in the sum $B_1 \cup \ldots \cup B_{\lfloor \log n - \log \delta \rfloor}$ every element can be repeated at most $4\delta^2$ times. This accounts for at least $\lfloor \log n - \log \delta \rfloor \cdot \frac{n}{4\delta} \cdot \frac{1}{4\delta^2} = n\frac{\lfloor \log n - \log \delta \rfloor}{16\delta^3}$ distinct elements of $\hat{B}$. For $\delta = \frac{\log^{1/3} n}{4}$ this is equal to $4n \cdot \frac{\lfloor \log n - 1/3 \cdot \log \log n + 2 \rfloor}{\log n} > 3n \geq |\hat{B}|$. The contradiction proves that at least one $k = 2^q$ for some q must satisfy the statement. ◀

Now let us return to the graph G_σ of significant pairs. Recall that σ is the largest symbol appearing in input sequences – for the rest of the section, we retain this assumption. As Figure 2 shows, any two incident edges must correspond to pairs with different values of LCIS, as otherwise the pairs could not be significant. This is formalized in a following simple observation:

▶ **Lemma 19.**

a) *If $(x, y_1), (x, y_2), \ldots, (x, y_s)$ are significant σ-pairs for $y_1 > y_2 > \ldots > y_s$, then for every $1 \le i \le j \le s$, we have $lcis(x, y_i) \ge lcis(x, y_j) + (j - i)$.*

b) *If $(x_1, y), (x_2, y), \ldots, (x_s, y)$ are significant σ-pairs for $x_1 > x_2 > \ldots > x_s$, then for every $1 \le i \le j \le s$, we have $lcis(x_i, y) \ge lcis(x_j, y) + (j - i)$.*

Proof. The first claim easily follows from the fact that $lcis(x, y_i) \ge lcis(x, y_{i+1}) + 1$, which is part of the definition of the significant pair. The second proof is symmetric. ◀

Finally, we can restate Lemma 14 and prove it using predecessor matrices:

▶ **Lemma 14.** *For every A and B with $|A|, |B| \le n$, and for the corresponding graph G_σ, there exists some positive integer $k \le n/\delta$ such that there is no k-dense suffix in $\hat{A}$.*

Proof. Assume, to the contrary, that for every k there is a k-dense suffix. Pick an arbitrary $k \le n/\delta$ and let $x \in A$ be such that $A_x^{\rightarrow}$ is k-dense. Let $|\hat{A}| = a$, let $r = \lceil n/\delta \rceil$ and let $(x, c_1), \ldots, (x, c_r)$ be the significant pairs from the definition of k-dense suffix (meaning that each c_i has k neighbours in $A_x^{\rightarrow}$). We can assume that $c_1 > c_2 > \ldots > c_r$. We will now show another sequence $c'_1, c'_2, \ldots, c'_r$ such that:

(1) $lcis(a, c'_1) > lcis(a, c'_2) > \ldots > lcis(a, c'_r)$,

(2) $lcis(a, c'_i) \ge lcis(x, c'_i) + k - 1$ for $i = 1, 2, \ldots, r$.

To do that, we set $c'_1 = c_1$, and for $i > 1$, we pick $c'_{i+1} = c_{i+1}$ if $lcis(a, c_{i+1}) < lcis(a, c'_i)$. If not, we define c'_{i+1} to be the largest element with $lcis(a, c'_{i+1}) = lcis(a, c'_i) - 1$. Observe that in the second case we know that $lcis(a, c_{i+1}) \ge lcis(a, c'_i) > lcis(a, c'_{i+1})$, so always $c'_{i+1} \le c_{i+1}$.

Inequality (1) follows immediately from the definition of c'_i. To see (2), first observe that if $c'_i = c_i$, then there are k significant σ-pairs between (x, c_i) and (a, c_i) – neighbors of c_i – which we will denote by $(y_1, c_i), \ldots, (y_k, c_i)$ and assume that $y_1 > \ldots > y_k$. As we assume σ to be the largest symbol, we can write $lcis(y_j, c_i) = lcis^{\rightarrow}(y_j, c_i)$. From this and Lemma 19 we derive:

$$lcis(a, c_i) \ge lcis(y_1, c_i) \ge lcis(y_k, c_i) + k - 1 \ge lcis(x, c_i) + k - 1.$$

Now consider the case $c'_i < c_i$. Let β be the smallest integer such that $\beta < i$ and $c_{i-\beta} = c'_{i-\beta}$ (it always exists, as we can take $\beta = i - 1$). From the definition of c'_i we have $lcis(a, c'_i) = lcis(a, c'_{i-1}) - 1 = \ldots = lcis(a, c'_{i-\beta}) - \beta = lcis(a, c_{i-\beta}) - \beta \ge lcis(x, c_{i-\beta}) + k - 1 - \beta$. Now, because all (x, c_j) are significant, we have $lcis(x, c_{i-\beta}) \ge lcis(x, c_i) + \beta$ from Lemma 19, so $lcis(a, c'_i) \ge lcis(x, c_i) + k - 1 \ge lcis(x, c'_i) + k - 1$.

The pairs (a, c'_i) are some choice of r different columns of the predecessor matrix M. Consider any c'_i, and its corresponding column j. Pick a positive integer $s \le k - 1$. Let $(a_i^*, c_i^*) = M[s, j]$. Recall that from the definition of M we have $lcis^{\rightarrow}(a_i^*, c_i^*) = lcis(a, c'_i) - s + 1$. We also know that $c_i^* \le c'_i \le c_i$. If $a_i^* < x$, then $(a_i^*, c_i^*) \le (x, c'_i)$, which implies $lcis(a, c'_i) - s + 1 = lcis(a_i^*, c_i^*) \le lcis(x, c'_i) \le lcis(a, c'_i) - k + 1$, which is impossible for $s \le k - 1$. Then $a_i^* \ge x$. So the only colors available for $M[s, j]$ for the chosen r columns

and $s \leq k-1$ are those appearing in $A_x^{\rightarrow}$, and there are at most $\lceil k\delta \rceil$ of them. Hence, for any k we can produce, from a k-dense suffix, an $\lceil \frac{n}{\delta} \rceil$-column submatrix of M having at most $\lceil k\delta \rceil$ colors in total in its first $k-1$ rows. This contradicts Lemma 18 and proves that for some k there are no k-dense suffixes. ◀

4 The algorithm

To implement our algorithm, we need a specific data structure – an associative array which can store a number of elements ordered by their *keys*. We assume the keys to be distinct integers between 1 and n. This data structure A must provide the following operations:

- INSERT(A, s) – adds the element s to A,
- DELETE(A, x) – removes the element having key x from the A (we assume that this is called only for $x \in A$),
- FIND(A, x) – returns the element whose key is x if there is one, or NULL otherwise,
- NEXT(A, x), PREV(A, x) – returns the first element whose key is larger (respectively, smaller) than x.

To achieve the desired running time, we need all these operations to work in $\mathcal{O}(\log \log n)$ complexity (possibly amortized), and *van Emde Boas queue* [26] does exactly that. While the standard implementation requires $\mathcal{O}(n)$ space (and thus $\mathcal{O}(n)$ initialization time, which would be too much for us, as we employ $\mathcal{O}(n)$ queues), there is also a randomized version ([22]) that needs only $\mathcal{O}(m)$ time and space, where m is the maximal number of elements on the queue. In the full version of the paper we show how to construct a deterministic van Emde Boas queue with $\mathcal{O}\left(m + \frac{n}{\log^c n}\right)$ time and space bounds, with c being any desired constant, while retaining the $\mathcal{O}(\log \log n)$ query complexity. These bounds also suit our needs.

The algorithm takes, as the input, two integer sequences A and B with $|A| = |B| = n$. Its main idea is to consider all symbols from A and B in increasing order (there are at most $2n$ of them, so we can sort them in $\mathcal{O}(n \log n)$). For every symbol σ, the algorithm finds and stores all significant σ-pairs. For that, we employ n van Emde Boas queues $Q_1, Q_2, \ldots, Q_n$, with every Q_k storing the significant pairs (x, y) with $lcis^{\rightarrow}(x, y) = k$, sorted by x. For convenience, we define $Q = Q_1 \cup \ldots \cup Q_n$. We also keep Q_0 as one-element queue $(0, 0)$.

Whenever some Q_k contains two pairs $(x, y) \leq (x', y')$, we drop (x', y') from Q_k. Informally, we can do it because (x, y) can replace (x', y') in every situation. We say that (x, y) *dominates* (x', y') and remove any dominated pairs from any Q_k. Observe that a pair is significant if and only if it is not dominated by any pair of the same symbol (a significant pair, however, may still be dominated by other pairs with larger symbols).

This leads to the following invariant:

▷ Claim 20 (Algorithm invariant). For every k, all pairs $(x, y) \in Q_k$ are in strict increasing order with respect to x and in strict decreasing order with respect to y.

To keep the invariant, we modify INSERT() into the following INSERT-INV() procedure. It only inserts a pair (x, y) if it is not dominated by another pair, and after inserting it removes all larger pairs.

The amortized complexity of INSERT-INV() is $\mathcal{O}(\log \log n)$: the loop in lines 14-19 deletes an element with every iteration (so it cannot do more iterations than the total number of elements in queue), and outside the loop there is only a constant number of standard queue operations.

Algorithm 1 New version of INSERT() keeping the invariant.

```
1: procedure INSERT-INV(Q_k, (x, y))
2:     (x', y') ← FIND(Q_k, x)                    ▷ check for other pairs with key x
3:     if (x', y') ≠ null and y' ≤ y then
4:         return
5:     end if
6:     if (x', y') ≠ null and y' > y then
7:         DELETE(x')
8:     end if
9:     (a, b) ← PREV(Q_k, x)
10:    if (a, b) ≠ null and b ≤ x then      ▷ (a, b) ≤ (x, y), so we should not insert (x, y)
11:        return
12:    end if
13:    INSERT(Q_k, (x, y))
14:    repeat                    ▷ now we remove all (a, b) ≥ (x, y), restoring the invariant
15:        (a, b) ← NEXT(Q_k, x)
16:        if (a, b) ≠ null and ≥ (x, y) then
17:            DELETE(Q_k, a)
18:        end if
19:    until (a, b) = null or not (a, b) ≥ (x, y)
20: end procedure
```

Apart from queues $Q_1, Q_2, \ldots, Q_n$ we will also need, for every symbol σ, two van Emde Boas queues X_σ and Y_σ which store positions of all σ-symbols in A and B, respectively: $X_\sigma = \{i : A[i] = \sigma\}$, $Y_\sigma = \{j : B[j] = \sigma\}$. These structures do not change during the algorithm, and their sole purpose is finding σ-symbols closest to a given position.

Now we are ready to introduce the main idea of the algorithm. Recall that we assume that the number of significant pairs between A and B is at most $\mathcal{O}\left(\frac{n^2}{t}\right)$ with $t = t(n) = \Theta\left(\log^p n\right)$ for some p. We iterate over all the symbols, dividing them into two categories:

- *frequent* – appearing more than $\frac{n}{\sqrt{t}}$ times in B,
- *infrequent* – with at most $\frac{n}{\sqrt{t}}$ occurrences in B.[‡]

Let us start with an informal sketch of the algorithm behavior for both cases. The frequent symbols are easier: for every such symbol σ we iterate through all previously found pairs, and for every $(x, y) \in Q$ we find the next occurrence of σ after $A[x]$ (say, $A[x^*]$) and the next occurrence y^* of σ in B after y. In other words, we find a σ-pair (x^*, y^*) for which (x, y) is a predecessor. As we will ensure that Q contains only significant pairs (and thus cannot get too big) and there are no more than $\sqrt{t}$ frequent symbols, the total complexity will fit into desired limits.

To handle infrequent symbols, observe that every such symbol in A can form a matching pair with at most $\frac{n}{\sqrt{t}}$ elements of B. Hence, there are at most $\frac{n^2}{\sqrt{t}}$ matching pairs on infrequent symbols, so we can iterate through all of them. The hardest part is to determine, for every infrequent pair (x, y), the value of $lcis^{\rightarrow}(x, y)$. For that, we will need a separate subroutine and a non-trivial analysis.

[‡]The technique of splitting symbols of a string according to their number of occurences is not new – it has been used, e.g. in [4] and [5], though it is more common to have split thresholds closer to $\sqrt{n}$.

The whole algorithm is presented below:

Algorithm 2 LCIS by significant pairs.

```
1: procedure LCIS(A, B)
2:     Q_0 ← {(0, 0)}
3:     for all σ – symbols in increasing order do
4:         T ← ∅                                              ▷ for storing new pairs
5:         if σ occurs less than n/√t times in B then         ▷ if σ is infrequent...
6:             for all x : A[x] = σ, in increasing order do   ▷ fix x...
7:                 k ← 0
8:                 for all y : B[y] = σ, in inc. order do     ▷ ...compute lcis→(x, y) for all y
9:                     k' ← ComputeNextPair(x, y, k)          ▷ using a special subroutine
10:                    T ← T ∪ (x, y, k')
11:                    k ← k'                                 ▷ k = lcis→(x, y), for last considered y
12:                end for
13:            end for
14:        else                                               ▷ If σ is frequent...
15:            for k = 1, 2, ..., n do
16:                for all (x, y) ∈ Q_k do                    ▷ every pair (x, y) ∈ Q may be a predecessor...
17:                    x' ← Next(X_σ, x)
18:                    y' ← Next(Y_σ, y)                      ▷ ...of some σ-pair (x', y')
19:                    T ← T ∪ (x', y', k + 1)
20:                end for
21:            end for
22:        end if
23:        for all (x, y, k) ∈ T do                           ▷ all new pairs are now added to Q
24:            Insert-Inv(Q_k, (x, y))
25:        end for
26:    end for
27:    return largest k with Q_k ≠ ∅
28: end procedure
```

Before analyzing the algorithm, we must explain the ComputeNextPair() subroutine. It takes three arguments: positions $x \in A$, $y \in B$, such that $A[x] = B[y] = \sigma$, and an integer k. It assumes that $lcis^{\rightarrow}(x, y) \geq k$ and its goal is to find the exact value of $lcis^{\rightarrow}(x, y)$. It also assumes that for every $j < lcis^{\rightarrow}(x, y)$, there is a pair $(x_j, y_j) \in Q_j$ with $(x_j, y_j) \prec (x, y)$ – informally, this means that all the predecessors of (x, y) have already been considered, and Lemma 21 will prove that this condition is indeed satisfied whenever ComputeNextPair() is invoked.

Therefore the subroutine must determine the largest $\ell \geq k - 1$ for which there is a pair $(x', y') \prec (x, y)$ with $lcis^{\rightarrow}(x', y') = \ell$. We can guess ℓ, and verify whether there exists a right pair $(x', y') \prec (x, y)$ in Q_ℓ: if there is one, then $\text{Prev}(Q_\ell, x) \leq (x, y)$. This allows us to do a binary search for ℓ:

Algorithm 3 Finding $lcis^{\rightarrow}(x, y)$, with assumption that it is at least k.

```
1: procedure ComputeNextPair(x, y, k)
2:     d ← 1                                  ▷ first, find d – a rough approximation for ℓ − k
3:     while Prev(Q_{k+d}, x) ≺ (x, y) do
4:         d ← 2d                                          ▷ if d is too small, try 2d
5:     end while
6:     p ← k                                   ▷ now we know that k + d/2 ≤ ℓ < k + d
7:     q ← k + d
8:     while p < q do                                  ▷ so we can do the real binary search
9:         s ← ⌈(p+q)/2⌉
10:        if Prev(Q_s, x) ≺ (x, y) then
11:            p ← s
12:        else
13:            q ← s − 1
14:        end if
15:    end while
16:    return p
17: end procedure
```

The first part of the algorithm finds d for which $d/2 \leq |\ell - k| < d$ (if $\ell = k$, then we assume $d = 1$). The second one is a binary search on the interval $[k, k + d)$. Therefore, the algorithm makes at most $2\log(d + 1) = 2 \cdot \log(lcis^{\rightarrow}(x, y) - k + 2)$ steps, each step invoking a queue operation once.

Let us now go back to the main algorithm, proving its correctness:

▶ **Lemma 21.** *After processing a symbol σ, the following two facts hold:*

(1) *If $(x, y) \in Q_k$, then (x, y) is a significant σ'-pair for some $\sigma' \leq \sigma$ with $lcis^{\rightarrow}(x, y) = k$;*

(2) *For any $k \geq 1$, $\sigma' \leq \sigma$ and for every σ'-pair (x, y) with $lcis^{\rightarrow}(x, y) \geq k$, there is some $(x^*, y^*) \in Q_k$ with $(x^*, y^*) \leq (x, y)$.*

Proof. Let us use induction on σ. Observe that assuming induction hypothesis, we only need to prove two weaker facts:

(1') If $(x, y) \in Q_k$, then $lcis^{\rightarrow}(x, y) = k$;

(2') For any $k \geq 1$ and for every σ-pair (x, y) with $lcis^{\rightarrow}(x, y) = k$, there is some $(x^*, y^*) \in Q_k$ with $(x^*, y^*) \leq (x, y)$.

Indeed, for any pair (x, y) with $lcis^{\rightarrow}(x, y) = k' > k$, (2) is true because of the induction hypothesis applied to the pair $(x', y') = \pi^{k'-k}(x, y)$. We know that (x', y') is a τ-pair for some $\tau < \sigma$, so induction hypothesis provides a pair $(x^*, y^*) \in Q_k$ with $(x^*, y^*) \leq (x', y') \leq (x, y)$. Next, note that (2) is also true for $\sigma' < \sigma$, again from induction hypothesis and the fact that once a pair is in Q, it can only be dislocated by another pair that dominates it.

Also, if we show (2) and (1'), this will automatically imply that every $(x, y) \in Q_k$ must be significant, and thus that (1) is true – let (x', y') be a pair with $(x', y') \leq (x, y)$ and $lcis^{\rightarrow}(x', y') = lcis^{\rightarrow}(x, y) = k$. There is, by (2), another pair $(x'', y'') \leq (x', y') \leq (x, y)$, which is also in Q_k. But with Claim 20, (x, y) and (x'', y'') cannot both be in Q_k unless $(x, y) = (x', y') = (x'', y'')$, so (x, y) is significant.

To prove the remaining statements (1') and (2'), consider two cases:

Case 1: infrequent σ. Pick an arbitrary σ-pair (x, y) and let $lcis^{\rightarrow}(x, y) = k$. We will show that at some point during processing symbol σ the instruction Insert-Inv$(Q_k, (x, y))$ is invoked.

For any integer i with $1 \leq i \leq k$ let $(x_i, y_i) = \pi^i(x, y)$. We know that $symbol(x_i, y_i) < \sigma$ and $lcis^{\rightarrow}(x_i, y_i) = k - i$, so by induction hypothesis (2) there was some $(x_i^*, y_i^*) \in Q_{k-i}$ with $(x_i^*, y_i^*) \leq (x_i, y_i)$, which implies $(x_i^*, y_i^*) \prec (x, y)$ (observe that for $i = k$, we need the artificial pair $(0, 0) \in Q_0$). But then when (x, y) is considered by ComputeNextPair(), for every $i \geq 1$ and for each of the predecessors $(x_i, y_i) = \pi^i(x, y)$ there is a pair $(x_i^*, y_i^*) \leq (x_i, y_i)$ in Q_{k-i}. This satisfies the conditions needed by ComputeNextPair(), so the binary search properly computes $lcis^{\rightarrow}(x, y)$ as k (note that it is not possible to find any larger candidate for predecessor – any $(u, v) \prec (x, y)$ found in $Q_{k'}$ for $k' \geq k$ would mean either that $lcis^{\rightarrow}(u, v)$ had been computed wrong, or that $lcis^{\rightarrow}(x, y) > k$). This shows that the algorithm tries to insert (x, y) into the proper queue Q_k, which immediately proves (1'). Also, (x, y) may remain in Q_k, be dislocated later, or even fail to be inserted because of some other pair dominating it. Either way, some pair $(x^*, y^*) \leq (x, y)$ will be present in Q_k to the very end of the algorithm. This completes the proof of (2').

Case 2: frequent σ. Let us first prove (2') for any k and for any σ-pair (x, y) with $lcis^{\rightarrow}(x, y) = k$. We can assume that (x, y) is significant (if not, we replace it with a significant σ-pair $(x', y') \leq (x, y)$). As in Case 1, from the induction hypothesis we know that some pair $(x', y') \prec (x, y)$ is present in Q_{k-1}. The algorithm must at some point consider (x', y'). If the next σ symbol in A after x' is not x but some z, then $lcis^{\rightarrow}(z, y) \geq k$, so (x, y) could not be significant. By the same argument, the next σ symbol in B after y' must be y. Therefore (x, y) is a candidate to be inserted into Q_k, so either it remains there itself, or is dominated by other pair $(x^*, y^*) \in Q_k$. Either way, (2') is shown.

For (1) we need to rule out a possibility that a σ-pair (x, y) with $lcis(x, y) = k$ will be inserted, besides Q_k, into some other $Q_{k'}$ with $k' \neq k$, as a frequent pair could be theoretically considered multiple times by the algorithm. But for $k' > k$ this would have been caused by another pair $(x^*, y^*) \in Q_{k'-1}$ with $(x^*, y^*) \prec (x, y)$. This contradicts $lcis^{\rightarrow}(x, y) = k$, as $k' - 1 \geq k$. For $k' < k$ observe that by induction hypothesis (2) applied to $\pi^{k-k'}(x, y)$ we already have a pair in $Q_{k'}$ which dominates (x, y), so the insertion must fail. ◀

▶ **Corollary 22.** *The algorithm correctly returns the length of longest common increasing subsequence of A and B.*

Proof. Let k be the value of LCIS and let (x, y) be a significant pair with $lcis^{\rightarrow}(x, y) = k$. From statement (2) of Lemma 21 we know that some pair $(x', y') \leq (x, y)$ must be in Q_k at the end of the algorithm. Therefore the algorithm returns at least k. On the other hand, for every $k' > k$, $Q_{k'} = \varnothing$, as any pair in it would contradict statement (1) from Lemma 21. ◀

▶ **Lemma 23.** *Let $x \leq |A|$ and $(x, y_1), (x, y_2), \ldots, (x, y_m)$ be all σ-pairs formed by x, for an infrequent σ. Then, all calls to ComputeNextPair$(x, \cdot, \cdot)$ work in $\mathcal{O}\left(\frac{n(\log\log n)^2}{\sqrt{t}}\right)$ total time complexity.*

Proof. We know that $m \leq \frac{n}{\sqrt{t}}$, as σ is infrequent. Recall also that $\log t = \Theta(\log\log n)$. Let $\ell_i = lcis^{\rightarrow}(x, y_i) - lcis^{\rightarrow}(x, y_{i-1}) + 2$ for $i = 1, 2, \ldots, m$, assuming $y_0 = 0$. The i-th call to ComputeNextPair() requires $\mathcal{O}(\log \ell_i)$ steps of binary search, with every step having

$\mathcal{O}(\log\log n)$ complexity from queue operations. Therefore, the whole procedure works in $\mathcal{O}(\log\log n \cdot \sum_{i=1}^{m} \log \ell_i)$. Consider two cases:

- If $m \le \frac{n}{\sqrt{t}\log n}$, then $\sum_{i=1}^{m} \log \ell_i \le m \log n = \mathcal{O}\left(n/\sqrt{t}\right)$.
- If $m > \frac{n}{\sqrt{t}\log n}$, then the Jensen equality yields:
  $\sum_{i=1}^{m} \log \ell_i = m \frac{\sum_{i=1}^{m} \log \ell_i}{m} \le m \log\left(\frac{\sum_{i=1}^{m} \ell_i}{m}\right)$,
  and as $\sum \ell_i = 2m + lcis^{\rightarrow}(x, y_m) - lcis^{\rightarrow}(x, y_0) \le 3n$ we have:
  $\sum_{i=1}^{m} \log \ell_i \le m \log \frac{3n}{m} < m \log(3\sqrt{t} \cdot \log n) = \mathcal{O}(m \log\log n)$.

With $m \le n/\sqrt{t}$, the total complexity in both cases is $\mathcal{O}\left(\frac{n(\log\log n)^2}{\sqrt{t}}\right)$. ◀

Proof of Theorem 12. We already know that the algorithm correctly computes $lcis(A, B)$. It remains to determine its complexity.

There are $\mathcal{O}(n)$ van Emde Boas queues. Each of them is initialized in constant time if we use randomized version [22], or in $\mathcal{O}(n/\log^c n)$ with some c large enough if we choose the version described in the appendix of the full version of the paper . Either way, total complexity is $\mathcal{O}\left(n^2/\log^c n\right)$, which is fast enough.

We first analyze the cost for infrequent symbols – it is dominated by the calls of ComputeNextPair(). By Lemma 23 the cost is $\mathcal{O}\left(\frac{n(\log\log n)^2}{\sqrt{t}}\right)$ for a fixed x, which yields $\mathcal{O}\left(\frac{n^2(\log\log n)^2}{\sqrt{t}}\right)$ total complexity.

Now let us move on to frequent symbols. For every such symbol, we iterate over all Q. But from Lemma 21 we know Q only contains significant pairs, therefore $|Q| = \mathcal{O}\left(\frac{n^2}{t}\right)$. As every iteration needs only a constant number of queue operations, the total cost for a single symbol is $\mathcal{O}\left(\frac{n^2 \log\log n}{t}\right)$.

Finally, observe that there are at most $\sqrt{t}$ frequent symbols (otherwise there would be $|B| > n$), so the final complexity in this case is $\mathcal{O}\left(\frac{n^2 \log\log n}{\sqrt{t}}\right)$.

It is also worth noting that we can replace the threshold between frequent and infrequent symbols $\frac{n}{\sqrt{t}}$ with $\frac{n}{\sqrt{t \log\log n}}$, and all the proofs would essentially work in the same way, with only minor changes needed. This way we could show the complexity of the algorithm to be in fact $\mathcal{O}\left(\frac{n^2(\log\log n)^{3/2}}{\sqrt{t}}\right)$. The current analysis seems, however, a bit easier to read. ◀

5 Final remarks and open problems related to LCIS

We have shown an algorithm for LCIS that breaks the $\mathcal{O}(n^2)$ barrier, but there still appears to be plenty of room for improvement and further work on this matter. First, the bound in Theorem 11 for the number of significant pairs may not be tight. In the full version of the paper we give an example of two sequences A and B having $\Omega\left(\frac{n^2}{\log n}\right)$ significant pairs, but this still leaves a gap between $\Omega\left(\frac{n^2}{\log n}\right)$ and $\mathcal{O}\left(\frac{n^2}{(\log n)^{1/3}}\right)$. Also, the algorithm itself might be improved to work in $\mathcal{O}\left(s \cdot (\log\log n)^k\right)$, where s is the number of significant pairs and k is a constant. Taking all this into account, we conjecture that there is an $\mathcal{O}\left(\frac{n^2(\log\log n)^k}{\log n}\right)$ algorithm which uses the significant pairs technique.

The second question related to LCIS stems from the papers [3] and [2]: we know that there is a constant $c \le 7$ such that an $\mathcal{O}\left(\frac{n^2}{\log^c n}\right)$ algorithm for LCS would lead to unexpected breakthroughs in circuit complexity. Can a similar statement be made for LCIS?

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Quadratic-time hardness of LCS and other sequence similarity measures. In *Proc. 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS'15)*, pages 59–78, 2015.

2 Amir Abboud and Karl Bringmann. Tighter connections between formula-sat and shaving logs. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 8:1–8:18, 2018. doi:10.4230/LIPIcs.ICALP.2018.8.

3 Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: Or: a polylog shaved is a lower bound made. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 375–388, New York, NY, USA, 2016. ACM. doi:10.1145/2897518.2897653.

4 Karl Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, December 1987. doi:10.1137/0216067.

5 Moshe Lewenstein Amihood Amir and Ely Porat. Faster algorithms for string matching with k mismatches. In *J. of Algorithms*, pages 794–803, 2000.

6 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proc. 47th Annual ACM Symposium on Theory of Computing (STOC'15)*, pages 51–58, 2015.

7 Ilya Baran, Erik D. Demaine, and Mihai Pătraşcu. Subquadratic algorithms for 3sum. In *Proceedings of the 9th International Conference on Algorithms and Data Structures*, WADS'05, pages 409–421, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/11534273_36.

8 Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*, pages 1216–1235, 2018.

9 Timothy M. Chan. The art of shaving logs. In Frank Dehne, Roberto Solis-Oba, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures*, pages 231–231, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

10 Timothy M. Chan. More logarithmic-factor speedups for 3sum, (median,+)-convolution, and some geometric 3sum-hard problems. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 881–897, Philadelphia, PA, USA, 2018. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=3174304.3175327.

11 Wun-Tat Chan, Yong Zhang, Stanley P. Y. Fung, Deshi Ye, and Hong Zhu. Efficient algorithms for finding a longest common increasing subsequence. *Journal of Combinatorial Optimization*, 13(3):277–288, 2007.

12 Lech Duraj. A linear algorithm for 3-letter longest common weakly increasing subsequence. *Information Processing Letters*, 113(3):94–99, 2013.

13 Lech Duraj, Marvin Künnemann, and Adam Polak. Tight conditional lower bounds for longest common increasing subsequence. *Algorithmica*, 81(10):3968–3992, October 2019. doi:10.1007/s00453-018-0485-7.

14 Szymon Grabowski. New tabulation and sparse dynamic programming based techniques for sequence similarity problems. *Discrete Applied Mathematics*, 212:96–103, 2016. doi:10.1016/j.dam.2015.10.040.

15 Yijie Han and Tadao Takaoka. An $o(n^3 \log\log n/\log^2 n)$ time algorithm for all pairs shortest paths. In Fedor V. Fomin and Petteri Kaski, editors, *Algorithm Theory – SWAT 2012*, pages 131–141, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

16 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

17 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

18 Guy Jacobson and Kiem-Phong Vo. Heaviest increasing/common subsequence problems. In *Combinatorial Pattern Matching, Third Annual Symposium, CPM 92, Tucson, Arizona, USA, April 29 - May 1, 1992, Proceedings*, pages 52–66, 1992.

19 Martin Kutz, Gerth Stølting Brodal, Kanela Kaligosi, and Irit Katriel. Faster algorithms for computing longest common increasing subsequences. *J. Discrete Algorithms*, 9(4):314–325, 2011. doi:10.1016/j.jda.2011.03.013.

20 Kasper Green Larsen and Ryan Williams. Faster online matrix-vector multiplication. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 2182–2189, Philadelphia, PA, USA, 2017. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=3039686.3039828.

21 William J. Masek and Mike Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.

22 Kurt Mehlhorn and Stefan Näher. Bounded ordered dictionaries in o(log log N) time and o(n) space. *Inf. Process. Lett.*, 35(4):183–189, 1990. doi:10.1016/0020-0190(90)90022-P.

23 Adam Polak. Why is it hard to beat $O(n^2)$ for longest common weakly increasing subsequence? *Information Processing Letters*, 132:1–5, 2018.

24 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proc. 45th Annual ACM Symposium on Symposium on Theory of Computing (STOC'13)*, pages 515–524, 2013.

25 Yoshifumi Sakai. A linear space algorithm for computing a longest common increasing subsequence. *Inf. Process. Lett.*, 99(5):203–207, 2006. doi:10.1016/j.ipl.2006.05.005.

26 P. van Emde Boas. Preserving order in a forest in less than logarithmic time. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, SFCS '75, pages 75–84, Washington, DC, USA, 1975. IEEE Computer Society. doi:10.1109/SFCS.1975.26.

27 Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

28 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005.

29 Ryan Williams. Matrix-vector multiplication in sub-quadratic time: (some preprocessing required). In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 995–1001, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1283383.1283490.

30 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 664–673, New York, NY, USA, 2014. ACM. doi:10.1145/2591796.2591811.

31 I-Hsuan Yang, Chien-Pin Huang, and Kun-Mao Chao. A fast algorithm for computing a longest common increasing subsequence. *Information Processing Letters*, 93(5):249–253, 2005.

32 Daxin Zhu and Xiaodong Wang. A space efficient algorithm for lcis problem. In Guojun Wang, Mohammed Atiquzzaman, Zheng Yan, and Kim-Kwang Raymond Choo, editors, *Security, Privacy, and Anonymity in Computation, Communication, and Storage*, pages 70–77, Cham, 2017. Springer International Publishing.

Fixed-Parameter Algorithms for Unsplittable Flow Cover

Andrés Cristi
Universidad de Chile, Santiago, Chile
andres.cristi@ing.uchile.cl

Mathieu Mari
École Normale Supérieure, Université PSL, Paris, France
mathieu.mari@ens.fr

Andreas Wiese
Universidad de Chile, Santiago, Chile
awiese@dii.uchile.cl

Abstract

The Unsplittable Flow Cover problem (UFP-cover) models the well-studied general caching problem and various natural resource allocation settings. We are given a path with a demand on each edge and a set of tasks, each task being defined by a subpath and a size. The goal is to select a subset of the tasks of minimum cardinality such that on each edge e the total size of the selected tasks using e is at least the demand of e. There is a polynomial time 4-approximation for the problem [Bar-Noy et al., STOC 2000] and also a QPTAS [Höhn et al., ICALP 2014]. In this paper we study fixed-parameter algorithms for the problem. We show that it is W[1]-hard but it becomes FPT if we can slightly violate the edge demands (resource augmentation) and also if there are at most k different task sizes. Then we present a parameterized approximation scheme (PAS), i.e., an algorithm with a running time of $f(k) \cdot n^{O_\epsilon(1)}$ that outputs a solution with at most $(1+\epsilon)k$ tasks or assert that there is no solution with at most k tasks. In this algorithm we use a new trick that intuitively allows us to pretend that we can select tasks from OPT multiple times.

2012 ACM Subject Classification Theory of computation → Packing and covering problems; Theory of computation → Fixed parameter tractability

Keywords and phrases Unsplittable Flow Cover, fixed parameter algorithms, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.42

Funding *Andrés Cristi*: Supported by CONICYT-PFCHA/Doctorado Nacional/2018-21180347.
Mathieu Mari: This work was partially funded by the grant ANR-19-CE48-0016 from the French National Research Agency (ANR).
Andreas Wiese: Partially supported by FONDECYT Regular grant 1170223.

1 Introduction

In the Unsplittable Flow Cover problem (UFP-cover) we are given a path $G = (V, E)$ where each edge e has a demand $u_e \in \mathbb{N}$, and a set of tasks T where each task $i \in T$ has a start vertex $s_i \in V$ and an end vertex $t_i \in V$, defining a path $P(i)$, and a size $p_i \in \mathbb{N}$. The goal is to select a subset of the tasks $T' \subseteq T$ of minimum cardinality $|T'|$ that covers the demand of each edge, i.e., such that $\sum_{i \in T' \cap T_e} p_i \geq u_e$ for each edge e where T_e denotes the set of tasks $i \in T$ for which e lies on $P(i)$. It is the natural covering version of the well-studied Unsplittable Flow on a Path problem (UFP), see e.g., [22, 21, 9] and references therein. Also, it is a generalization of the knapsack cover problem [11] and it can model general caching in the fault model where we have a cache of fixed size and receive requests for non-uniform size pages, the goal being to minimize the total number of cache misses (see [1, 5, 16] and Appendix A). Caching and generalizations of it have been studied for several decades in computer science, see e.g., [1, 8, 20, 24]. Also, UFP-cover is motivated by

many resource allocation settings in which for instance the path specifies a time interval and the edge demands represent minimum requirements for some resource like energy, bandwidth, or number of available machines at each point in time.

UFP-cover is strongly NP-hard, since it generalizes general caching in the fault model [16], and the best known polynomial time approximation algorithm for it is a 4-approximation [5] with no improvement in almost 20 years. However, the problem admits a QPTAS for the case of quasi-polynomial input data [23] which suggests that better polynomial time approximation ratios are possible.

In this paper, we study the problem for the first time under the angle of fixed parameter tractability (FPT). We define our parameter k to be the number of tasks in the desired solution and seek algorithms with a running time of $f(k)n^{O(1)}$ for some function f. We show that by allowing such a running time we can compute solutions that are almost optimal.

1.1 Our contribution

First, we prove that UFP-cover is W[1]-hard which makes it unlikely that it admits an FPT-algorithm. In particular, this motivates studying FPT-approximation algorithms or other relaxations of the problem. We first show that under slight resource augmentation the problem becomes FPT. We define an additional parameter $\delta > 0$ controlling the amount of resource augmentation and we compute either a solution that is feasible if we decrease the demand of each edge e to $u_e/(1+\delta)$, or we assert that there is no solution of size k for the original edge demands. Key to our result is to prove that due to the resource augmentation we can assume that each edge e is completely covered by tasks whose size is comparable to u_e or it is covered by at least one task whose size is much larger than u_e. Based on this we design an algorithm that intuitively sweeps the path from left to right and on each uncovered edge e we guess which of the two cases applies. In the former case, we show that due to the resource augmentation we can restrict ourselves to only $f(k,\delta)$ many guesses for the missing tasks using e. In the latter case e belongs to a subpath in which each edge is covered by a task that is much larger than the demand of e. We guess the number of tasks in this subpath and select tasks to maximize the length of the latter. This yields a subproblem that we solve recursively and we embed the recursion into a dynamic program.

▶ **Theorem 1.** *There is an algorithm for UFP-cover with running time $k^{O(\frac{k}{\delta}\log k)} \cdot n^{O(1)}$ that either outputs a solution of size at most k that is feasible if the edge capacities are decreased by a factor $1+\delta$ or asserts that there is no solution of size k for the original edge capacities.*

We use the above algorithm to obtain a simple FPT-2-approximation algorithm *without* resource augmentation. Also, with similar ideas we derive an algorithm computing the *optimal* solution, assuming that additionally the number of different task sizes in the input is bounded by a parameter.

▶ **Theorem 2.** *There is an algorithm that solves UFP-cover in time $k^{O(k'k)} \cdot n^{O(1)}$, assuming that $|\{p_i : i \in T\}| \leq k'$.*

Then we present a parameterized approximation scheme (PAS) for UFP-cover, i.e., an algorithm with a running time of $f(k) \cdot n^{O_\epsilon(1)}$ that outputs a solution with at most $(1+\epsilon)k$ tasks or assert that there is no solution with at most k tasks. This algorithm is based on a lemma developed for UFP in which we have the same input as in UFP-cover but we want to *maximize* the weight of the selected tasks T' and require that their total size is *upper*-bounded by u_e on each edge e, i.e., $\sum_{i \in T' \cap T_e} p_i \leq u_e$. Informally, the mentioned lemma states that we can remove a set of tasks from $OPT_{SL} \subseteq OPT$ of negligible cardinality

such that on each edge e we remove one of the largest tasks of OPT using e. This yields some slack that we can use in order to afford inaccuracies in the computation. Translated to UFP-cover, the natural correspondence would be a solution in which the tasks in OPT_{SL} are not removed but selected twice. This is not allowed in UFP-cover. However, we guess a set of tasks T' that intuitively yields as much slack as OPT_{SL} and whose size is also negligible. If $OPT \cap T' \neq \emptyset$ then we cannot add the tasks in T' to OPT to gain slack since some of them are already included in OPT. Therefore, we use the following simple but useful trick: we guess $T' \cap OPT$ for which there are $2^{|T'|} \leq 2^{O(\epsilon k)}$ options, select the tasks in $T' \cap OPT$, and recurse on the remaining instance. Since $OPT \leq k$ the whole recursion tree has a complexity of $O(k^k)$ which depends only on our parameter k.

If $OPT \cap T' = \emptyset$ then $T' \cup OPT$ is a $(1+\epsilon)$-approximate solution with some slack and we can use the slack in our computation. We compute a partition of E into $O(k)$ intervals. Some of these intervals are *dense*, meaning that there are many tasks from OPT that start or end in them. We ensure that for each dense interval there is a task in T' that covers the whole interval and whose size is at least a $\Omega(1/k)$-fraction of the demand of each edge in the interval. Intuitively this is equivalent to decreasing the demand on each edge by a factor $1 + \Omega(1/k)$. If we had only dense intervals we could apply the FPT-algorithm for resource augmentation from above for the remaining problem. On the other hand, if only few tasks start or end in an interval we say that it is *sparse*. If all intervals are sparse, we devise a dynamic program that processes them in the order of their amount of slack and guesses their tasks step by step. We use the slack in order to be able to "forget" some of the previously guessed tasks which yields a DP with only polynomially many cells.

Unfortunately, in an instance there can be dense *and* sparse intervals and our algorithms above for the two special cases are completely incompatible. Therefore, we identify a type of tasks in OPT such that we can guess tasks that cover as much as those tasks, while losing only a factor of $1 + \epsilon$. Using some charging arguments, we show that then we can split the remaining problem into two independent subinstances, one with only dense intervals and one with only sparse intervals which we then solve with the algorithms mentioned above.

▶ **Theorem 3.** *There is a parameterized approximation scheme for UFP-cover.*

Our algorithms for resource augmentation, a bounded number of task sizes, and the FPT-2-approximation even work in the weighted case, at the expense of a factor $1 + \epsilon$ in the approximation ratio. Due to space constraints the details of this and many proofs are deferred to the full version of the paper.

1.2 Other related work

The study of parameterized approximation algorithms was initiated independently by Cai and Huang [10], Chen, Grohe, and Grüber [14], and Downey, Fellows, and McCartin [18]. A good survey on the topic was given by Marx [26]. Recently, the notion of approximate kernels was introduced [25]. Independently, Bazgan [7] and Cesati and Trevisan [12] established an interesting connection between approximation algorithms and parameterized complexity by showing that EPTASs, i.e., $(1+\epsilon)$-approximation algorithms with running time $f(\epsilon)n^{O(1)}$, imply FPT algorithms for the decision version. Hence a W[1]-hardness result for a problem makes the existence of an EPTAS for it unlikely.

For the unweighted case of UFP (packing) a PAS is known [27]. Note that in the FPT setting UFP is easier than UFP-cover since we can easily make the following simplifying assumptions that we cannot make in UFP-cover. First, we can assume that the input tasks are not too small: if there are k input tasks whose size is smaller than $1/k$ times the capacity

of any the edges they use, then we can simple output those tasks and we are done; otherwise we can enumerate over them and only large tasks remain. Second, the tasks are not too big since the size of a task can be assumed to be at most the minimum capacity of an edge in its path. Third, we can easily find a set of at most k edges that together intersect the path of each input task (i.e., a hitting set for the input task's paths) unless a simple greedy algorithm finds a solution of size k [27]. The best known polynomial time approximation algorithm for UFP has a ratio of $5/3 + \epsilon$ [22] and the problem admits a QPTAS [3, 6].

Recently, polynomial time approximation algorithms for special cases of UFP-cover under resource augmentation were found: an algorithm computing a solution of optimal cost if $p_i = c_i$ for each task i and a $(1 + \epsilon)$-approximation if the cost of each task equals its "area", i.e., the product of p_i and the length of $P(i)$ [17]. UFP-cover is a special case of the general scheduling problem (GSP) on one machine in the absence of release dates. The best known polynomial time result for GSP is a $(4 + \epsilon)$-approximation [15] and a QPTAS for quasi-polynomial bounded input data [2]. Also, UFP-cover is a special case of the capacitated set cover problem, e.g., [13, 4].

2 Few different task sizes

In this section, we show that UFP-cover is FPT when it is parameterized by $k + k'$ where k' is the number of different task sizes. We are given two parameters k and k' and assume $|\{p_i : i \in T\}| = k'$. We seek to compute a solution $T' \subseteq T$ with $|T'| \leq k$ such that for each edge e it holds that $p(T' \cap T_e) := \sum_{i \in T' \cap T_e} p_i \geq u_e$ or assert that there is no such solution. Denote by $T^{(\ell)}$ for $\ell = 1, \ldots, k'$ the partition of the set T into sets of tasks with equal size.

Our algorithm sweeps the path from left to right and guesses the tasks in OPT step by step (in contrast to similar such algorithms it is *not* a dynamic program). We maintain a set T' of previously selected tasks and a pointer indicating an edge e. We initialize the algorithm with e being the leftmost edge of E and $T' := \emptyset$. Suppose that the pointer is at some edge e. If the tasks in T' already cover the demand of e, i.e., $p(T_e \cap T') \geq u_e$, then we move the pointer to the edge on the right of e. Otherwise, in OPT the edge e must be covered by a task that is not in T'. For each group $T^{(\ell)}$ we guess the number of tasks using e that are missing in T' compared to OPT, i.e., we guess $k_\ell := |T_e \cap OPT \cap T^{(\ell)}| - |T_e \cap T' \cap T^{(\ell)}|$. Since there are at most k' groups $T^{(\ell)}$, the number of possible guesses is bounded by $(k+1)^{k'}$. For each group $T^{(\ell)}$ we add to T' the k_ℓ tasks in $(T_e \cap T^{(\ell)}) \setminus T'$ with rightmost endvertex. Then we move the pointer to the edge on the right of e. Overall, we want to select at most k tasks. Therefore, at each guessing step, we enumerate only guesses that ensure that we do not select more than k tasks altogether. Hence, the total number of possible guesses overall is bounded by $((k+1)^{k'})^k = k^{O(k'k)}$. Each of them yields a set T'. In case that the resulting set T' is not a feasible solution we reject the guesses that lead to T'.

Assume from now on that all guesses were correct. In the next lemma we show that then we obtain a feasible solution. The intuition for the proof is as follows: suppose that the pointer is at some edge e and we select additional tasks from a group $T^{(\ell)}$. These additional tasks were not necessary in order to cover the demands of the edges on the left of e. All tasks in $T^{(\ell)}$ have exactly the same size. Therefore, the best choice is to select the tasks in $T_e \cap T^{(\ell)}$ with rightmost endvertices.

▶ **Lemma 4.** *Assume that the given instance has a solution of size at most k. Then the resulting set T' satisfies $\sum_{i \in T' \cap T_e} p_i \geq u_e$ for each edge e.*

The total number of guesses is bounded by $k^{O(k'k)}$ and for each set of guesses we can compute the corresponding set T' in time $n^{O(1)}$. Hence, we obtain Theorem 2.

3 Resource augmentation

In this section, we turn to the case where we have resource augmentation but the number of different task sizes is arbitrary. As a consequence of Theorem 2, we first show that UFP-cover with $(1+\delta)$ resource augmentation, can be solved in time $f(k,\delta)\cdot n^{O(1)}$ if the edge demands come in a polynomial range. In Section 3.1 we generalize this algorithm to arbitrary edge demands.

For now, we assume that the edge demands come in a polynomial range. Then, for two parameters $k \in \mathbb{N}$ and $\delta > 0$ we seek to compute a solution $T' \subseteq T$ with $|T'| \leq k$ such that for each edge e it holds that $p(T' \cap T_e) := \sum_{i \in T' \cap T_e} p_i \geq \tilde{u}_e := u_e/(1+\delta)$ or assert that there is no solution $T' \subseteq T$ with $|T'| \leq k$ such that for each edge e it holds that $p(T' \cap T_e) \geq u_e$.

The idea is to round the task sizes and then use our algorithm for bounded number of task sizes. Let OPT denote a solution with at most k tasks. We group the tasks into groups such that the sizes of the tasks in the same group differ by at most a factor of $1+\delta$. For each $\ell \in \mathbb{N}$ we define the group $T^{(\ell)} := \{i \in T | p_i \in [(1+\delta)^\ell, (1+\delta)^{\ell+1})\}$. For each ℓ we round the sizes of the tasks in $T^{(\ell)}$ to $(1+\delta)^\ell$, i.e., for each $i \in T^{(\ell)}$ we define its rounded size to be $\tilde{p}_i := (1+\delta)^\ell$ (for convenience, we allow rounded the task sizes and edge demands to be fractional). As we show in the next lemma, this rounding step is justified due to our resource augmentation.

▶ **Lemma 5.** *By decreasing the demand of each edge e to $\tilde{u}_e := u_e/(1+\delta)$ we can assume for each $\ell \in \mathbb{N}$ that each task $i \in T^{(\ell)}$ has a size of $\tilde{p}_i = (1+\delta)^\ell$, i.e., for each edge e it holds that $\sum_{i \in OPT \cap T_e} \tilde{p}_i \geq \tilde{u}_e$.*

Note that w.l.o.g. we can assume that $p_i \leq \max_e u_e$ for each task i. Since we assumed that the edge demands are bounded by a polynomial in n, there are only $O(\log_{1+\delta} n)$ groups $T^{(\ell)}$ with $T^{(\ell)} \neq \emptyset$. The optimal solution contains tasks from at most k of these groups. We guess the groups $T^{(\ell)}$ that satisfy that $OPT \cap T^{(\ell)} \neq \emptyset$ in time $\binom{O(\log_{1+\delta} n)}{k} = (\frac{1}{\delta}\log n)^{O(k)}$. Note that the latter quantity is of the form $f(k,\delta)\cdot n^{O(1)}$, since $(\log n)^{O(k)} \leq n + k^{O(k)}$. We delete the tasks from all other groups. This yields an instance with at most k different (rounded) task sizes, and then we can apply Theorem 2 with $k' = k$. Hence, there is an algorithm with running time $(\frac{1}{\delta}\log n)^{O(k)} \cdot k^{O(k^2)} \cdot n^{O(1)} = f(k,\delta)\cdot n^{O(1)}$ if the edge demands are in a polynomial range.

3.1 Arbitrary demands

We extend the above algorithm now to the case of arbitrary demands. To this end, we start with a shifting step that intuitively partitions the groups above into supergroups such that the sizes of two tasks in different supergroups differ by at least a factor of $2k/\delta$. In particular, one task from one supergroup will be larger than any k tasks from supergroups with smaller tasks together. We define K to be the smallest integer such that $k(1+\delta)^{-K-1} < \delta/2$, i.e., $K = O(\frac{1}{\delta}\log k)$. Let $\alpha \in \{0, ..., k\}$ be an offset to be defined later. Intuitively, we remove an $\frac{1}{k+1}$-fraction of all groups $T^{(\ell)}$ and combine the remaining groups into supergroups. With a shifting argument we ensure that no task from OPT is contained in a deleted group. Formally, we define a supergroup $\mathcal{T}^{(s)} := \bigcup_{\ell=K(\alpha+s(k+1))}^{K(\alpha+(s+1)(k+1)-1)-1} T^{(\ell)}$ for each integer s. In particular, each supergroup contains $K \cdot k$ groups.

▶ **Lemma 6.** *There exists an offset $\alpha \in \{0, ..., k\}$ such that for each task $i \in OPT$ there is a supergroup $\mathcal{T}^{(s)}$ such that $i \in \mathcal{T}^{(s)}$.*

We guess the value α due to Lemma 6. If the edge demands are no necessarily polynomially bounded we can no longer guess the groups that contains tasks from OPT since there can be up to $\Omega(n)$ groups. Instead, for each edge e we define a level s_e to be the largest value s such that $(1+\delta)^{K(\alpha+s(k+1))} \leq \hat{u}_e := \tilde{u}_e/(1+\delta)$. Note that $(1+\delta)^{K(\alpha+s(k+1))}$ is a lower bound on the size of each task in $\mathcal{T}^{(s)}$. In the next lemma, using resource augmentation we prove that for each edge e it holds that the tasks in $\bigcup_{\ell'} \mathcal{T}^{(s_e)} \cap OPT$ are sufficient to cover the demand of e or that in OPT the edge e is completely covered by one task in a supergroup $\mathcal{T}^{(s')}$ with $s' > s_e$.

▶ **Lemma 7.** *For each edge e it holds that $p(OPT \cap \mathcal{T}^{(s_e)} \cap T_e) \geq \hat{u}_e$ or that there is a task $i \in OPT \cap \bigcup_{s=s_e+1}^{\infty} \mathcal{T}^{(s)} \cap T_e$. In the latter case it holds that $p_i \geq \tilde{p}_i \geq \hat{u}_e$.*

In order to solve our problem, we define a set of subproblems that we solve via dynamic programming. Let us denote $\mathcal{T}^{(\geq s)} := \bigcup_{s' \geq s} \mathcal{T}^{(s')}$. Each subproblem is characterized by a subpath $\tilde{E} \subseteq E$, and integers $\tilde{k}, \tilde{s}$ with $0 \leq \tilde{k} \leq k$ and $\tilde{s} \in \{-1, ..., O(\log \max_e u_e)\}$. A tuple $(\tilde{E}, \tilde{k}, \tilde{s})$ represents the following subproblem: select a set of tasks $T' \subseteq \mathcal{T}^{(\geq \tilde{s})}$ with $|T'| \leq \tilde{k}$ such that for each edge $e \in \tilde{E}$ it holds that $\sum_{i \in T' \cap T_e} \tilde{p}_i \geq \hat{u}_e$. Note that the subproblem $(E, k, -1)$ corresponds to the original problem that we want to solve. Moreover, the number of DP-cells is polynomial in the input length. Notice that there are only a polynomial number of values $\tilde{s}$ for which $\mathcal{T}^{(\tilde{s})}$ is non-empty.

Suppose we are given a subproblem $(\tilde{E}, \tilde{k}, \tilde{s})$ and assume that we already solved each subproblem of the form $(\tilde{E}', \tilde{k}', \tilde{s}')$ where $\tilde{E}' \subseteq \tilde{E}$, $\tilde{k}' \leq \tilde{k}$, and $\tilde{s}' > \tilde{s}$. Assume that in OPT each edge $e \in \tilde{E}$ is covered by at least one task in $\mathcal{T}^{(\geq \tilde{s})}$ (it will turn out that we need to compute a feasible solution to $(\tilde{E}, \tilde{k}, \tilde{s})$ only in this case). Hence, for covering the reduced edge demands $\hat{u}$ we do not need the tasks in supergroups $\mathcal{T}^{(s')}$ with $s' < \tilde{s}$. Our algorithm sweeps the path from left to right and guesses the tasks in OPT step by step (where OPT denotes the optimal solution to the original input instance). We maintain a pointer at some edge e and a set T' of previously selected tasks. We initialize the algorithm with e being the leftmost edge of $\tilde{E}$ and $T' := \emptyset$. Suppose that the pointer is at some edge e. If the tasks in T' already cover the reduced demand of e, i.e., $p(T_e \cap T') \geq \hat{u}_e$, then we move the pointer to the edge on the right of e. Otherwise, in OPT the edge e must be covered by a task that is not in T'. We guess whether $p(OPT \cap \mathcal{T}^{(\geq \tilde{s}+1)} \cap T_e) \geq \hat{u}_e$ or $p(OPT \cap \mathcal{T}^{(\tilde{s})} \cap T_e) \geq \hat{u}_e$. Since we assumed that e is covered by a task in $\mathcal{T}^{(\geq \tilde{s})}$, Lemma 7 yields that one of these two cases applies.

Suppose we guessed that $p(OPT \cap \mathcal{T}^{(\geq \tilde{s}+1)} \cap T_e) \geq \hat{u}_e$. For any two edges e_1, e_2 denote by P_{e_1,e_2} the subpath of E starting with e_1 and ending with e_2 (including e_1 and e_2). Let e' be the rightmost edge on the right of e such that each edge $e'' \in P_{e,e'}$ the set $OPT \cap T_{e''}$ contains at least one task in $\mathcal{T}^{(\geq \tilde{s}+1)}$. Let $\tilde{k}'$ denote the number of tasks in $OPT \cap \mathcal{T}^{(\geq \tilde{s}+1)}$ whose path intersects $P_{e,e'}$. We guess $\tilde{k}'$. Then we determine the rightmost edge e'' such that $(P_{e,e''}, \tilde{k}', \tilde{s}')$ is a yes-instance, where $\tilde{s}' \geq \tilde{s}+1$ is the smallest integer such that $\mathcal{T}^{(\tilde{s}')} \neq \emptyset$. We add to T' the tasks in the solution of $(P_{e,e''}, \tilde{k}', \tilde{s}')$ and move the pointer to the edge on the right of e''.

Assume now that we guessed that $p(OPT \cap \mathcal{T}^{(\tilde{s})} \cap T_e) \geq \hat{u}_e$. Observe that $\mathcal{T}^{(\tilde{s})}$ consists of only Kk non-empty groups $T^{(\ell)}$. For each of these groups $T^{(\ell)}$ we guess $k_\ell := |OPT \cap T_e \cap T^{(\ell)}| - |T' \cap T_e \cap T^{(\ell)}|$. Note that there are only $(k+1)^{Kk}$ possible guesses. For each group $T^{(\ell)}$ we add to T' the k_ℓ tasks in $(T_e \cap T^{(\ell)}) \setminus T'$ with rightmost endvertex. Then we move the pointer to the edge on the right of e.

Like before, at each guessing step, we enumerate only guesses that ensure that we do not select more than $\tilde{k}$ tasks altogether. Hence, the total number of possible guesses overall is bounded by $2^{\tilde{k}}(\tilde{k}+1)^{O(K\tilde{k})} = k^{O(\frac{k}{\delta} \log k)}$. We store in the cell $(\tilde{E}, \tilde{k}, \tilde{s})$ the set T' of minimum

size if a set of size $\tilde{k}$ was found. Finally, we output the solution in the cell $(E, k, -1)$ if it contains a feasible solution. If it does not contain a feasible solution we output that there is no solution of size k for the original edge capacities u. Theorem 1 follows.

4 FPT-2-approximation algorithm

We present an FPT-2-approximation algorithm *without* resource augmentation (for arbitrary edge demands), i.e., an algorithm that runs in time $f(k)n^{O(1)}$ and finds a solution of size at most $2k$ or asserts that there is no solution of size at most k. Suppose we are given an instance (I, k). First, we call the algorithm for resource augmentation from Section 3 with $\delta = 1$. If this algorithm asserts that there is no solution of size at most k then we stop. Otherwise, let ALG denote the found solution. We guess $ALG \cap OPT$. Note that there are only 2^k possibilities for $ALG \cap OPT$. If $ALG \cap OPT = \emptyset$ then the solution $OPT \cup ALG$ covers each edge e to an extent of at least $3/2 \cdot u_e$, i.e., $p((OPT \cup ALG) \cap T_e) \geq 3/2 \cdot u_e$. Therefore, we create a new UFP-cover instance I' whose input tasks are identical with the tasks in I and in which the demand of each edge e is changed to $u'_e := 3/2 \cdot u_e$. We invoke our algorithm for the resource augmentation setting from Section 3 to I' where we look for a solution of size at most $|ALG| + k \leq 2k$ and we set $\delta := 1/2$. Let ALG' be the returned solution. It holds that $|ALG'| \leq |ALG| + k \leq 2k$ and ALG' covers each edge e to an extent of at least $3/2 \cdot u_e/(1 + 1/2) = u_e$. We output ALG'.

If $ALG \cap OPT \neq \emptyset$ then we generate a new instance I'' in which the tasks in $ALG \cap OPT$ are already taken, i.e., the demand of each edge e is reduced to $u''_e := u_e - p(ALG \cap OPT \cap T_e)$ and the set of input tasks consists of $T \setminus (ALG \cap OPT)$. We recurse on I'' where the parameter k is set to $k - |ALG \cap OPT|$. Observe that $OPT'' := OPT \setminus ALG$ is a solution to I'' and if $|OPT| \leq k$ then $|OPT''| \leq k - |ALG \cap OPT|$. The resulting recursion tree has depth at most k with at most 2^k children per node and hence it has at most $2^{O(k^2)}$ nodes in total. This yields the following theorem.

▶ **Theorem 8.** *There is an algorithm for UFP-cover with a running time of $2^{O(k^2)} \cdot n^{O(1)}$ that either finds a solution of size at most $2k$ or asserts that there is no solution of size k.*

5 Parameterized approximation scheme

In this section, we present a PAS for UFP-cover. Given a parameter k, we seek to compute a solution of size at most $(1 + \epsilon)k$ or assert that there is no solution of size at most k. The running time of our algorithm is $k^{O(k)} n^{(1/\epsilon)^{O(1/\epsilon)}}$. Let OPT denote a solution with at most k tasks and let $\epsilon > 0$ such that $1/\epsilon \in \mathbb{N}$.

We describe first how we guess a partition of $E = I_0 \dot\cup I_1 \dot\cup \cdots \dot\cup I_r$ into $O(k)$ many subpaths that we denote as *intervals*. Also, we will guess a set of tasks $T_S \subseteq T$ such that if we add T_S to OPT then we obtain a certain amount of slack that we will use in the computation later. If $T_S \cap OPT \neq \emptyset$ then we will simply guess $T_S \cap OPT$ and recurse, *without* losing anything in the approximation ratio.

We group the tasks into groups such that the tasks in the same group have the same size, up to a factor $1+\epsilon$. Formally, for each integer ℓ we define $T^\ell := \{i \in T, p_i \in [(1+\epsilon)^\ell, (1+\epsilon)^{\ell+1})\}$ and we say that a task i is *of level* ℓ if $i \in T^\ell$. Then we run the 4-approximation algorithm from [5] to obtain a solution S. If $|S| > 4k$ then $OPT > k$ and we stop. For each edge e let $OPT_e^{1/\epsilon}$ denote the $1/\epsilon$ largest tasks in $OPT \cap T_e$ (breaking ties in an arbitrary fixed way). Intuitively, we would like to select the tasks OPT_{SL} due to the following lemma from [6, Lemma 3.1].

▶ **Lemma 9** ([6]). *There is a set $OPT_{SL} \subseteq OPT$ with $|OPT_{SL}| \leq \gamma\epsilon|OPT|$ such that for each edge e with $|OPT \cap T_e| \geq 1/\epsilon$ it holds that $OPT_{SL} \cap OPT_e^{1/\epsilon} \neq \emptyset$, where γ is a universal constant that is independent of the given instance.*

We cannot guess OPT_{SL} directly. Instead, we run the following algorithm that computes a set $T_S = T_S^{(1)} \cup T_S^{(2)} \cup T_S^{(3)}$ with at most $O(|OPT_{SL}|)$ tasks that gives us similar slack as OPT_{SL}. The reader may imagine that $T_S^{(1)} \cup T_S^{(2)} = OPT_{SL}$ and that $T_S^{(3)}$ are additional tasks that we select. We initialize $T_S^{(1)} = \emptyset$. Let $\tilde{V}$ be the set of start and end vertices of the tasks in S. We partition E according to the vertices in $\tilde{V}$. Formally, we consider the partition $E = \tilde{I}_1 \dot\cup \cdots \dot\cup \tilde{I}_{\tilde{r}}$ of E such that for each $\tilde{I}_j = \{v_1, ..., v_s\}$ we have that $v_1 \in \tilde{V}$ and $v_s \in \tilde{V}$ and for each $s' \in \{2, ..., s-1\}$ we have that $v_{s'} \notin \tilde{V}$. We say that a task i starts in an interval $\tilde{I}_j$ if $\tilde{I}_j$ is the leftmost interval that contains an edge of $P(i)$ and a task i ends in an interval $\tilde{I}_j$ if $\tilde{I}_j$ is the rightmost interval that contains an edge of $P(i)$. For each pair of intervals $\tilde{I}_j, \tilde{I}_{j'}$ we guess whether there is a task from OPT_{SL} that starts in $\tilde{I}_j$ and ends in $\tilde{I}_{j'}$. If yes, we add to $T_S^{(1)}$ the largest task with this property. Additionally, for each interval $\tilde{I}_j$ in which at least one task from OPT_{SL} starts or ends we add to $T_S^{(1)}$ the largest task $i^* \in T$ such that $\tilde{I}_j \subseteq P(i^*)$ and define $\bar{s}_j := p_{i^*}$. One can show that the maximum demand of an edge $e \in \tilde{I}_j$ is upper-bounded by $4k\bar{s}_j$ since i^* is at least as large as the largest task $i \in S$ with $\tilde{I}_j \subseteq P(i)$ and each task $i \in S$ starts or ends at a vertex in $\tilde{V}$. On the other hand, $\bar{s}_j$ is as large as k tasks of size at most $\frac{1}{k}\bar{s}_j$ together and hence we can ignore tasks of the latter kind for $\tilde{I}_j$ if we have $\bar{s}_j$ units of slack in $\tilde{I}_j$. Let L denote the set of values ℓ such that there is an interval $\tilde{I}_j$ and a task $i \in T^\ell$ with $p_i \in [\frac{1}{2k}\bar{s}_j, 4k\bar{s}_j]$. One can show that $|L| \leq O_\epsilon(k \log k)$ and $|T_S^{(1)}| \leq O(\epsilon k)$.

Next, we define a set $T_S^{(2)}$ of additional slack tasks. We maintain a queue $Q \subseteq V$ of vertices that we call *interesting* and a set of tasks $T_S^{(2)}$. At the beginning, we initialize $T_S^{(2)} := \emptyset$ and $Q := \tilde{V}$. In each iteration we extract an arbitrary vertex v from Q. Let Q' be the set of vertices that were removed from the queue Q in an earlier iteration. For each vertex v let T_v denote the input tasks i whose path $P(i)$ uses v, i.e., such that $P(i)$ contains an edge e incident to v. For each group T^ℓ with $\ell \in L$ we guess whether there is a task in OPT_{SL} that uses v but that does not use any vertex in Q', i.e., we guess whether there is a task in $OPT_{SL} \cap T^\ell \cap T_v \setminus \bigcup_{v' \in Q'} T_{v'}$. We add to $T_S^{(2)}$ the task with leftmost startvertex and the task with rightmost endvertex from $T^\ell \cap T_v \setminus \bigcup_{v' \in Q'} T_{v'}$. For each added task, we add its start- and its endvertex to Q if it has not been in Q before. The algorithm terminates once Q is empty.

Let $T_S^{(2)}$ be the resulting set. One can show that $|T_S^{(2)}| \leq O(\epsilon k)$. Let now V' be the set of start- and endvertices of tasks in $S \cup T_S^{(1)} \cup T_S^{(2)}$ and let $I_0 \cup I_1 \cup \cdots \cup I_r = E$ be the partition into subpaths defined by the vertices in V'. In the following, we partition intervals into three groups according to the number of tasks from OPT that start or end in them. Given an interval I let d be the number of tasks that start or end in I. Let α be some constant in $\{5, \cdots, 5/\epsilon\}$. We say that I is *sparse* if $d \leq 1/\epsilon^\alpha$, *medium* if $1/\epsilon^\alpha < d \leq 1/\epsilon^{\alpha+5}$ and *dense* if $d > 1/\epsilon^{\alpha+5}$.

▶ **Lemma 10.** *There exists an integer $\alpha \in \{5, \ldots, 5/\epsilon\}$ such that the number of tasks in OPT that start or end in a medium interval is at most $2\epsilon k$.*

We guess α and for each interval I_j we guess whether it is sparse, medium, or dense. Note that there are in total $\frac{5}{\epsilon}3^{O(k)}$ many guesses. We select now some more tasks that will provide us with additional slack. For each medium or dense interval I_j we select the largest task $i \in T$ such that $I_j \subseteq P(i)$. Also, for each maximal set of contiguous sparse intervals

$I_j \cup I_{j+1} \cup \cdots \cup I_{j'} =: \mathcal{I}$ we select the largest task $i \in T$ such that $\mathcal{I} \subseteq P(i)$. Let $T_S^{(3)}$ denote the resulting set. We have that $|T_S^{(3)}|$ is at most twice the total number of intervals that are medium or dense. In each of the latter intervals there are at least $1/\epsilon^5$ tasks from OPT that start or end. Therefore, $|T_S^{(3)}| \leq \epsilon k$. We call $T_S := T_S^{(1)} \cup T_S^{(2)} \cup T_S^{(3)}$ the *slack tasks*. For each interval I_j we denote by $\hat{s}_j$ the slack in the interval given by T_S, i.e., $\hat{s}_j = \min_{e \in I_j} p\left(T_e \cap T_S\right)$. To summarize, we obtained the following properties of our intervals and T_S.

▶ **Lemma 11.** *We have that $I_0 \dot\cup I_1 \dot\cup \cdots \dot\cup I_r$ is a partition of E into $O(k)$ intervals and $|T_S| \leq O(\epsilon k)$. For each edge e that is the leftmost or the rightmost edge of an interval I_j we have that there are at most $1/\epsilon$ tasks $i' \in OPT \cap T_e$ such that $\hat{s}_j(1+\epsilon) < p_{i'} < 4k\hat{s}_j$. For each dense interval I_j we have that $\hat{s}_j \geq \frac{1}{4k} \max_{e \in I_j} u_e$. Also, for each maximal set of contiguous sparse intervals $I_j \cup I_{j+1} \cup \cdots \cup I_{j'} =: \mathcal{I}$ we have that $\min_{j'' : j \leq j'' \leq j'} \hat{s}_{j''}$ is at least size of the largest task $i \in OPT$ with $\mathcal{I} \subseteq P(i)$ (if OPT contains such a task i).*

If the tasks in T_S are not contained in OPT then $OPT \cup T_S$ is a solution of size $(1+O(\epsilon))k$ in which each edge has some slack and hence we can use this slack algorithmically. Otherwise, we recurse: We guess $OPT \cap T_S$ and if $OPT \cap T_S \neq \emptyset$ then we recurse on a new instance where we assume that $OPT \cap T_S$ is already selected. Formally, this instance has input task $\bar{T} := T \setminus (OPT \cap T_S)$, each edge $e \in E$ has demand $\bar{u}_e := u_e - \sum_{i \in T_e \cap (OPT \cap T_S)} p_i$, and the parameter is $\bar{k} := k - |OPT \cap T_S|$. Together with the guesses above, this yields $k^{O(k^2)}$ many guesses. Hence, the recursion tree has depth at most k and each internal node has at most $k^{O(k^2)}$ children which yields $k^{O(k^3)}$ vertices in total. In the sequel, we will assume that $OPT \cap T_S = \emptyset$ and solve the remaining problem without any further recursion in time $f(k)n^{O(1)}$ for some function f.

5.1 Medium intervals

We describe a routine that essentially allows us to reduce the problem to the case where there are no medium intervals. From Lemma 10 we know that there are no more than $2\epsilon k$ tasks in OPT that start or end in a medium interval. Therefore, for those tasks we can afford to make mistakes that cost us a constant factor, i.e., we can select $O(\epsilon k)$ instead of $2\epsilon k$ of those tasks.

Let $T_{\text{med}} \subseteq T$ be the set of tasks that start or end in a medium interval. Let I_j be a medium interval. In OPT, the demand of the edges in I_j is partially covered by tasks $i \in OPT \setminus T_{\text{med}}$ that completely cross I_j, i.e., such that $I_j \subseteq P(i)$. We guess an estimate for the total size of such tasks, i.e., an estimate for $\hat{p}_j = p\left(\{i \in OPT \setminus T_{\text{med}} : I_j \subseteq P(i)\}\right)$. Formally, we guess $\hat{u}_j := \lfloor \hat{p}_j / (\hat{s}_j/3) \rfloor$.

▶ **Lemma 12.** *We have that $\hat{u}_j \in \{0, ..., 3k\}$ and for each edge $e \in I_j$ it holds that $p(T_e \cap OPT \cap T_{\text{med}}) + \hat{u}_j \hat{s}_j/3 + p(T_e \cap T_S) \geq u_e$.*

Since there are only $O(k)$ intervals, there are only $k^{O(k)}$ many guesses in total. We construct an auxiliary instance on the same graph $G = (V, E)$ with input tasks T_{med} and demand $u_e^{\text{med}} = \max\{u_e - p(T_e \cap (T_S)) - \hat{u}_\ell \hat{s}_\ell/3, 0\}$ for each $e \in E$ in a medium interval, and $u_e^{\text{med}} = 0$ for each edge $e \in E$ in a sparse or dense interval. We run the 4-approximation algorithm [5] on this instance, obtaining a set of tasks $T'_{\text{med}} \subseteq T_{\text{med}}$ with $|T'_{\text{med}}| \leq 4|OPT \cap T_{\text{med}}| \leq O(\epsilon k)$.

For our remaining computation for each medium interval I_j we define the demand u_e of each edge $e \in I_j$ to be $u_e := \hat{u}_j \hat{s}_j/3$. Lemma 12 implies that any solution T' for this changed instance yields a solution $T' \cup T'_{\text{med}} \cup T_S$ with at most $|T'| + O(\epsilon k)$ tasks for the original instance. In the sequel, denote by OPT' the optimal solution to the new instance.

5.2 Heavy vertices

Our strategy is to decouple the sparse and dense intervals. A key problem is that there are tasks $i \in OPT'$ such that $P(i)$ contains edges in sparse and in dense intervals. Intuitively, our first step is therefore to guess some of them in an approximate way.

There are vertices $v \in V'$ that are used by many tasks in $OPT' \cap T^\ell$ for some level ℓ. Formally, we say that for a set of tasks $T' \subseteq T$ a vertex $v \in V'$ is (ℓ, T')-*heavy* if there are more than $1/\epsilon^{\alpha+1}$ tasks $i \in T' \cap T^\ell \cap T_v$ such that i starts or ends in a sparse or a medium interval. We are interested in vertices $v \in V'$ that are (ℓ, OPT')-heavy for some ℓ. It turns out that we can compute a small number of levels ℓ for which this can happen based on the slacks $\hat{s}_j$ of the intervals I_j.

▶ **Lemma 13.** *Let $v \in V'$ and assume that v is (ℓ, OPT')-heavy for some $\ell \in \mathbb{N}_0$. Then $(1+\varepsilon)^\ell \in [\frac{1}{2k}\hat{s}_j, 4k \cdot \hat{s}_j]$ for some interval I_j.*

Therefore, let L denote the set of levels ℓ such that a vertex $v \in V'$ can be (ℓ, OPT')-heavy according to Lemma 13, i.e., $L := \{\ell | \exists j : (1+\varepsilon)^\ell \in [\frac{1}{2k}\hat{s}_j, 4k \cdot \hat{s}_j]\}$. Intuitively, for each level $\ell \in L$ and each (ℓ, OPT')-heavy vertex $v \in V'$ we want to select a set of tasks $\bar{T}_{\ell,v} \subseteq T^\ell \cap T_v$ that together cover as much as the tasks in OPT' due to which v is (ℓ, OPT')-heavy, i.e., the tasks in $OPT' \cap T^\ell \cap T_v$.

To this end, we do the following operation for each level $\ell \in L$. We perform several iterations. We describe now one iteration and assume that $v' \in V'$ is the vertex that we processed in the previous iteration (at the first iteration v' is undefined and let $T_{v'} := \emptyset$ in this case). We guess the leftmost vertex $v \in V'$ on the right of v' that is $(\ell, OPT' \setminus T_{v'})$-heavy. Let $OPT'_{\ell,v} := OPT' \cap T^\ell \cap T_v \setminus T_{v'}$. We want to compute a set $\bar{T}_{\ell,v}$ that is not much bigger than $OPT'_{\ell,v}$, i.e., $|\bar{T}_{\ell,v}| \leq (1 + O(\epsilon))|OPT'_{\ell,v}|$ and that covers at least as much on each edge e as $OPT'_{\ell,v}$, i.e., $p(T_e \cap \bar{T}_{\ell,v}) \geq p(T_e \cap OPT'_{\ell,v})$. We initialize $\bar{T}_{\ell,v} := \emptyset$. We consider each pair of intervals I_j and $I_{j'}$ such that all edges of I_j are on the left of v (but might have v as an endpoint) and all edges of $I_{j'}$ are on the right of v (but might have v as an endpoint) and such that I_j or $I_{j'}$ is sparse or medium. We guess the number $k^{v,\ell}_{j,j'}$ of tasks from $OPT' \cap T^\ell \cap T_v \setminus T_{v'}$ that start in I_j and end in $I_{j'}$ (and hence are contained in T_v). If I_j is sparse or medium (and hence then $I_{j'}$ can be anything), we add to $\bar{T}_{\ell,v}$ the $k^{v,\ell}_{j,j'}$ tasks from $T^\ell \setminus T_{v'}$ with rightmost endvertex that start in I_j and end in $I_{j'}$. If I_j is dense (and hence then $I_{j'}$ is sparse or medium) we add to $\bar{T}_{\ell,v}$ the $k^{v,\ell}_{j,j'}$ tasks from $T^\ell \setminus T_{v'}$ with leftmost startvertex that start in I_j and end in $I_{j'}$. Note that $\sum_{j,j'} k^{v,\ell}_{j,j'} = |OPT'_{\ell,v}|$. Intuitively, the tasks in $\bar{T}_{\ell,v}$ cover each edge of E to a similar extent as the tasks in $OPT'_{\ell,v}$. We will show that the difference is compensated by additionally adding the following tasks to $\bar{T}_{\ell,v}$: we add to $\bar{T}_{\ell,v}$ the $\lfloor 2/\epsilon^\alpha + 2\epsilon|OPT'_{\ell,v}| \rfloor$ tasks from $T^\ell \cap T_v \setminus (\bar{T}_{\ell,v} \cup T_{v'})$ with leftmost start vertex. After this, we add to $\bar{T}^\ell_v$ the $\lfloor 2/\epsilon^\alpha + 2\epsilon|OPT'_{\ell,v}| \rfloor$ tasks from $T^\ell \cap T_v \setminus (\bar{T}_{\ell,v} \cup T_{v'})$ with rightmost end vertex. Let $\bar{T}_{\ell,v}$ denote the resulting set. We prove that it covers as much as $OPT'_{\ell,v}$ and that it is not much bigger than $|OPT'_{\ell,v}|$.

▶ **Lemma 14.** *For each edge $e \in E$ in a dense or a sparse interval, we have that $p(T_e \cap \bar{T}_{\ell,v}) \geq p(T_e \cap OPT'_{\ell,v})$. For each edge e in a medium interval, we have that $p(T_e \cap \bar{T}_{\ell,v} \setminus T_{\text{med}}) \geq p(T_e \cap OPT'_{\ell,v} \setminus T_{\text{med}})$. Also, it holds that $|\bar{T}_{\ell,v}| \leq (1 + O(\epsilon))|OPT'_{\ell,v}|$.*

We continue with the next iteration where now v' is defined to be the vertex v from above. We continue until in some iteration there is no vertex $v \in V'$ on the right of v' that is $(\ell, OPT' \setminus T_{v'})$-heavy. Let $V'_\ell \subseteq V'$ denote all vertices that at some point were guessed as being the $(\ell, OPT' \setminus T_{v'})$-heavy vertex v above. Let $T_H := \bigcup_{\ell \in L} \bigcup_{v \in V'_\ell} \bar{T}_{\ell,v}$ denote the set of computed tasks and define $OPT'_H := \bigcup_{\ell \in L} \bigcup_{v \in V'_\ell} OPT'_{\ell,v}$.

▶ **Lemma 15.** *We have that $p(T_e \cap T_H) \geq p(T_e \cap OPT'_H)$ for each edge e in a dense and a sparse interval, and $p(T_e \cap T_H \setminus T_{\text{med}}) \geq p(T_e \cap OPT'_H \setminus T_{\text{med}})$ for each edge e in a medium interval. Also, it holds that $|T_H| \leq (1 + O(\epsilon))|OPT'_H|$.*

It remains to compute a set of tasks T' such that $T' \cup T_H \cup T_S$ is feasible. Intuitively, T' should cover as much as $OPT' \setminus OPT'_H$ on each edge. To this end, we decouple the problem into one for the dense intervals and one for the sparse intervals. One problem for this is that even after selecting the tasks in T_H we might need to select additional tasks that have one endpoint in a dense interval and the other one in a sparse interval. However, we will show that since we selected the tasks in T_S, for each dense interval I there are only constantly many such tasks that still matter. We show that we can afford to select such tasks twice (once in the subproblem for the dense intervals and once in the subproblem for the sparse intervals) since we can charge them to the many tasks that start or end in I. For this charging to work we use that in a dense interval there can be many more tasks that start or end than in a sparse interval.

5.3 Dense intervals

Recall that for each dense interval I_j we have that $\hat{s}_j \geq \frac{1}{4k} \max_{e \in I_j} u_e$ (see Lemma 11). Hence, intuitively it suffices to compute a solution for I_j that is feasible under $(1 + \frac{1}{4k})$-resource augmentation. So in order to compute a set of tasks T' that cover the remaining demand in all dense intervals I_j (after selecting T_H) we could apply the algorithm for resource augmentation from Section 3 directly as a black box. However, there are also the sparse intervals and it might be that there are tasks $i \in OPT' \setminus OPT'_H$ that are needed for a dense interval *and* for a sparse interval. There are two types of such tasks. The first type are tasks that have at least one endpoint in a sparse interval. For each dense interval I_j there can be at most $2/\epsilon^\alpha$ such tasks in $T^\ell \cap OPT' \setminus OPT'_H$ for each group ℓ (since each of them needs to overlap one of the endpoints of I_j). We will show later that the slack due to T_S is as large as almost all of them together, all apart from $1/\epsilon^{\alpha+4}$ many. Hence, if we select the latter tasks twice (once in the subproblem for the dense intervals and once in the subproblem for the sparse intervals) we can charge them to the tasks that start or end in I_j and hence we increase our cost by at most a factor $1 + \epsilon$. The second type are tasks that start and end in a dense interval. Let $T_D \subseteq T$ denote the set of all such tasks. Note that a vertex can be crossed by more than constantly many tasks in $T_D \cap OPT' \setminus OPT'_H$. To handle those tasks, we guess an estimate for the demand that such tasks cover in the sparse intervals. Therefore, for each sparse interval $I_{j'}$ we guess a value $\hat{u}_{j'}$ such that $\hat{u}_{j'} = \left\lfloor \frac{p(T_e \cap T_D \cap OPT' \setminus OPT'_H)}{\hat{s}_j/4} \right\rfloor \cdot \hat{s}_j/4$ for each edge $e \in I_{j'}$ (note that $p(T_e \cap T_D \cap OPT' \setminus OPT'_H)$ is identical for each edge $e \in I_{j'}$). Then $p(T_e \cap T_D \cap OPT' \setminus OPT'_H)$ essentially equals $\hat{u}_{j'}$ and we show that the difference is compensated by our slack, even if we cover a bit less than $\hat{u}_{j'}$ units on each edge $e \in I_{j'}$.

▶ **Lemma 16.** *Let $I_{j'}$ be a sparse interval. Then $\hat{u}_{j'} \in \{0, \frac{\hat{s}_j}{4}, 2 \cdot \frac{\hat{s}_j}{4}, \ldots, 4k \cdot \frac{\hat{s}_j}{4}\}$ and $\hat{u}_{j'} \leq p(T_e \cap T_D \cap OPT' \setminus OPT'_H) \leq \frac{1}{(1+\frac{1}{4k})} \hat{u}_{j'} + p(T_e \cap T_S)/2$ for each $e \in I_{j'}$.*

We generate now an auxiliary instance where in each sparse interval $I_{j'}$ we reduce the demand u_e of each edge $e \in I_{j'}$ to $\hat{u}_{j'}$ (but do not change the demand on any edge in a dense interval) and remove all input tasks i such that $P(i)$ does not contain an edge of a dense interval. Also, for each remaining task i we shorten its path $P(i)$ to a path $P'(i)$ such that $P'(i)$ is the longest path contained in $P(i)$ that starts and ends on a vertex in a dense interval. We apply the algorithm from Section 3 with $(1 + \delta)$-resource augmentation to this instance with $\delta := 1/4k$. We obtain a solution $T^{(1)}$ such that for each edge e in an interval I_j, the solution

$T^{(1)}$ covers at least $u_e - \hat{s}_j/2$ when I_j is dense and $\hat{u}_j - \hat{s}_j/2$ when I_j is sparse. Notice that according to Lemma 11, we have $\hat{s}_j \geq u_e/(4k)$ for each edge e in a dense interval I_j and for each edge e lying in a maximal set of contiguous sparse intervals that is completely crossed by at least one input task.

▶ **Lemma 17.** *For each edge e in a dense interval we have that $p(T_e \cap (T^{(1)} \cup T_H \cup T_S)) \geq u_e$. For each edge e in a sparse interval $I_{j'}$ we have that $p(T_e \cap T^{(1)} \cap T_D) + p(T_e \cap T_S)/2 \geq p(T_e \cap T_D \cap OPT' \setminus OPT'_H)$.*

Due to Lemma 17 we cover the complete demand in each dense interval and some portion of the demand in each sparse interval. Therefore, for the remaining problem for each edge e in a sparse interval $I_{j'}$ we change its demand to $\bar{u}_e := u_e - \hat{u}_{j'}$. Also, we remove all tasks in T_D from the input, i.e., we work with the input tasks $\bar{T} := T \setminus T_D$. We claim that $\overline{OPT} := OPT' \setminus (OPT'_H \cup T_D)$ is a solution to the residual instance.

▶ **Lemma 18.** *For each edge e in a sparse interval we have that $p(T_e \cap \overline{OPT}) \geq \bar{u}_e$.*

5.4 Sparse intervals

Recall that in each sparse interval $I_{j'}$ there are at most $1/\epsilon^\alpha$ tasks from OPT that start or end in $I_{j'}$ (and hence the same is true for $\overline{OPT} \subseteq OPT$). Therefore, for each sparse interval we can guess these tasks in time $n^{O(1/\epsilon^\alpha)}$. If it was even true that for each sparse interval $I_{j'}$ there are at most $1/\epsilon^\alpha$ tasks $i \in OPT$ with $P(i) \cap I_{j'} \neq \emptyset$ then we could devise a simple dynamic program (DP) that sweeps the intervals from left to right and computes the optimal solution. Unfortunately, this is not true, but note that each vertex $v \in V'$ is used by at most $1/\epsilon^{\alpha+1}$ tasks in $\overline{OPT} \cap T^\ell$ for each group T^ℓ. Using this, we devise a more complicated DP that processes the intervals in the order of their slacks $\hat{s}_j$ and guesses step by step the at most $1/\epsilon^\alpha$ tasks that start or end in each of them. In order to restrict the running time to a polynomial we use the tasks in T_S in order to be able to "forget" some previously guessed tasks, i.e., we argue that the forgotten tasks have a total size that is at most the size of the slack due to T_S. Let us define a constant $\beta := 1 + \log_{1+\epsilon}\left(\frac{6}{\epsilon^{\alpha+2}}\right)$ and a constant $\Gamma := 1/\epsilon^\alpha + 1/\epsilon + (\beta+2)/\epsilon^{\alpha+1}$. Formally, each DP-cell is described by

- two intervals $I_j, I_{j'}$ such that for each interval $I_{j''}$ between I_j and $I_{j'}$ it holds that $\hat{s}_{j''} \geq \max\{\hat{s}_j, \hat{s}_{j'}\}$,
- two sets of tasks T'_j and $T'_{j'}$ of size at most Γ such that for each $i \in T'_j$ (*resp.* $i \in T'_{j'}$) it holds that $P(i) \cap I_j \neq \emptyset$ (*resp.* $P(i) \cap I_{j'} \neq \emptyset$) and $p(T_e \cap T'_j) + p(T_e \cap T_S)/2 \geq \bar{u}_e$ (*resp.* $p(T_e \cap T'_{j'}) + p(T_e \cap T_S)/2 \geq \bar{u}_e$) for each edge $e \in I_j$ (*resp.* $e \in I_{j'}$), i.e., the tasks in T'_j (*resp.* $T'_{j'}$) essentially cover the demand of I_j (*resp.* $I_{j'}$).

Such a cell $(I_j, I_{j'}, T'_j, T'_{j'})$ represents the subproblem of selecting a set of tasks $\hat{T}$ such that the path of each task $i \in \hat{T}$ lies between I_j and $I_{j'}$ and does not use any edge of $I_j \cup I_{j'}$ and such that $T'_j \cup T'_{j'} \cup \hat{T}$ cover the demand $\bar{u}_e$ for each edge between I_j and $I_{j'}$ together with half of the slack, i.e., $p(T_e \cap (T'_j \cup T'_{j'} \cup \hat{T})) + p(T_e \cap T_S)/2 \geq \bar{u}_e$.

Suppose we are given a cell $(I_j, I_{j'}, T'_j, T'_{j'})$ and we want to compute a solution $DP(I_j, I_{j'}, T'_j, T'_{j'})$ for it. Let $I_{j''}$ denote the interval between I_j and $I_{j'}$ with smallest slack $\hat{s}_{j''}$ (breaking ties arbitrarily). Let ℓ'' be the greatest integer such that $(1+\epsilon)^{\ell''} \leq \hat{s}_{j''}$. Let $T^{\geq \ell''-\beta} := \bigcup_{\ell \geq \ell''-\beta} T^\ell$. The intuition is that we guess the tasks in $\overline{OPT}$ that use $I_{j''}$ and when we recurse we forget all tasks that are not in $T^{\geq \ell''-\beta}$. We will show that our slack compensates the forgotten tasks. Therefore, we can ensure that if we always guess all tasks from $\overline{OPT}$ correctly then we will have only subproblems $(I_j, I_{j'}, T'_j, T'_{j'})$ where $|T'_j| \leq \Gamma$ and $|T'_{j'}| \leq \Gamma$. Formally, we enumerate all sets of tasks $T'_{j''} \subseteq T^{\geq \ell''-\beta}$ such that there are at most Γ tasks in $T'_{j''}$,

$P(i) \cap I_{j''} \neq \emptyset$ for each $i \in T'_{j''}$, and the tasks in $T'_{j''}$ cover the demand of $I_{j''}$ together with half of the slack in T_S, i.e., $p(T_e \cap T'_{j''}) + p(T_e \cap T_S)/2 \geq \bar{u}_e$ for each edge $e \in I_{j''}$. For a fixed guess of $T'_{j''}$ we associate the solution $T'_j \cup T'_{j'} \cup T'_{j''} \cup DP(I_j, I_{j''}, T'_j, T'_{j''}) \cup DP(I_{j''}, I_{j'}, T'_{j''}, T'_{j'})$. We define $DP(I_j, I_{j'}, T'_j, T'_{j'})$ to be the solution of minimum size associated to one of the enumerated sets $T'_{j''}$. For DP-cells $(I_j, I_{j'}, T'_j, T'_{j'})$ such that there is no interval between I_j and $I_{j'}$ we define $DP(I_j, I_{j'}, T'_j, T'_{j'}) := \emptyset$. For convenience, assume that we append two dummy intervals I_{-1} and I_{r+1} on the left and on the right of E that are not used by any task and that have zero demand on each of their edges. Also, we define that they have zero slack, i.e., $\hat{s}_{-1} = \hat{s}_{r+1} = 0$. We output the solution $DP(I_{-1}, I_{r+1}, \emptyset, \emptyset)$.

In order to show that the above DP is correct, one key step is to argue that it is unproblematic to neglect the tasks that are not in $T^{\geq \ell'' - \beta}$ in each respective step. This is shown in the following lemma.

▶ **Lemma 19.** *Let $I_j, I_{j'}$ be two intervals such that for each interval $I_{j''}$ between I_j and $I_{j'}$ it holds that $\hat{s}_{j''} \geq \max\{\hat{s}_j, \hat{s}_{j'}\}$. Let ℓ'' be the greatest integer such that $(1+\epsilon)^{\ell''} \leq \hat{s}_{j''}$ for all intervals $I_{j''}$ between I_j and $I_{j'}$. Then for each edge e between I_j and $I_{j'}$ it holds that $d\left(T_e \cap \overline{OPT} \cap T^{\geq \ell'' - \beta}\right) + p(T_e \cap T_S)/2 \geq \bar{u}_e$.*

Also, we need to show that when we enumerate the sets $T'_{j''}$ above one candidate set consists of the tasks in $\overline{OPT}$ that use $I_{j''}$ but neither I_j nor $I_{j'}$ and that in particular the latter set contains at most Γ tasks.

▶ **Lemma 20.** *Let $I_{j''}$ be a sparse interval and let ℓ'' be the greatest integer such that $(1+\epsilon)^{\ell''} \leq \hat{s}_{j''}$. Then there are at most Γ tasks in $\overline{OPT} \cap T^{\geq \ell'' - \beta}$ that use an edge of $I_{j''}$.*

Equipped with Lemmas 19 and 20 we can prove that the above DP is correct by arguing that it will produce $\overline{OPT}$ if it makes the corresponding guesses for each DP-cell. Also, by construction the returned solution is feasible. This yields the following lemma.

▶ **Lemma 21.** *There is an algorithm with a running time of $n^{O(1/\epsilon^{\alpha+4})}$ that computes a set $T^{(2)} \subseteq \bar{T}$ with $|T^{(2)}| \leq |\overline{OPT}|$ and $p(T_e \cap T^{(2)}) + p(T_e \cap T_S)/2 \geq \bar{u}_e$ for each edge e.*

It remains to argue that our computed sets $T'_{\text{med}}, T^{(1)}, T^{(2)}, T_H, T_S$ together form a feasible solution and do not contain too many tasks. With the following lemma we complete the proof of Theorem 3.

▶ **Lemma 22.** *We have that $T'_{\text{med}} \cup T^{(1)} \cup T^{(2)} \cup T_H \cup T_S$ is a feasible solution to the original input instance (T, E) and $|T'_{\text{med}} \cup T^{(1)} \cup T^{(2)} \cup T_H \cup T_S| \leq (1 + O(\epsilon))k$.*

6 $W[1]$-hardness

In this section we prove that UFP-cover is $W[1]$-hard if the parameter k represents the number of tasks in the optimal solution. Our proof goes along the lines of the proof that UFP (packing) is $W[1]$-hard for the same parameter as in [27].

▶ **Theorem 23.** *UFP-cover problem is $W[1]$-hard when parameterized by the number of tasks in the optimal solution.*

We give a reduction from the k-subset sum problem which is $W[1]$-hard [19]. Given a set of n values $A = \{a_1, ..., a_n\}$, a target value B and an integer k, the goal is to choose exactly k values from A that sum up to exactly B.

Suppose we are given an instance of k-subset sum. First, we claim that we can assume w.l.o.g. some properties of it.

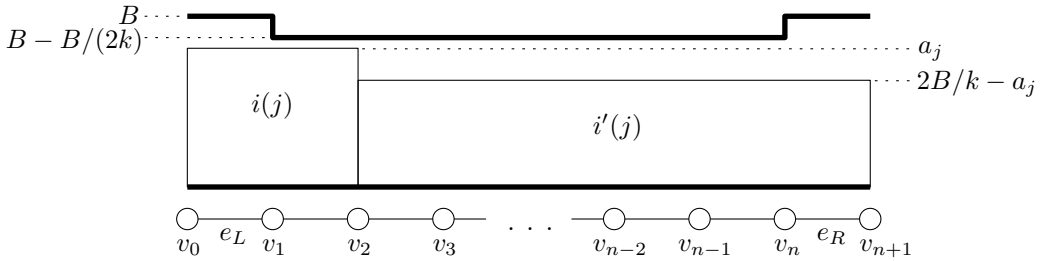


Figure 1 Sketch of the reduction used in order to prove Theorem 23. The sketch shows the tasks $i(j)$ and $i'(j)$ for only one index j. The figure is essentially identical to a figure in [27], taken with consent of the author.

▶ **Lemma 24.** *W.l.o.g. we can assume that there are values $\epsilon_1, ..., \epsilon_n$, not necessarily positive, such that $a_i = B/k + \epsilon_i$ for each $i \in [n]$ and that $\sum_{i=1}^{n} |\epsilon_i| < B/(2k)$.*

We construct an instance of UFP-cover that admits a solution with $2k$ tasks if and only if the given k-subset sum is a yes-instance. Our UFP-cover instance has a path with $n+2$ vertices $v_0, v_1, ..., v_{n+1}$. Denote the leftmost and the rightmost edge by e_L and e_R, respectively. We define $u(e_L) = u(e_R) = B$. For all other edges e we define $u(e) := B - B/(2k)$. Assume that the values in S are ordered such that $a_1 \geq a_2 \geq ... \geq a_n$. Let $j \in [n]$. We introduce two tasks $i(j), i'(j)$ with $s(i(j)) := v_0$, $t(i(j)) := v_j$, $p(i(j)) := a_j$ $s(i'(j)) := v_j$, $t(i'(j)) = v_{n+1}$, and $p(i'(j)) := 2B/k - a_j$. See Figure 1 for a sketch.

In order to get some intuition about the constructed instance, we prove the following lemma.

▶ **Lemma 25.** *Any feasible solution contains at least $2k$ tasks. Among them are k tasks covering e_L and k tasks covering e_R. If a task covers e_L then it does not cover e_R and vice versa.*

In the next lemma we show how to construct a solution with $2k$ tasks if the given k-subset sum instance is a yes-instance.

▶ **Lemma 26.** *If the given k-subset sum instance is a yes-instance, then the constructed UFP-cover instance has a solution with $2k$ tasks.*

Conversely, we show that if the UFP-cover instance has a solution with at most $2k$ tasks then the k-subset sum instance is a yes-instance. Suppose we are given such a solution for the UFP-cover instance. First, we establish that for each $j \in [n]$ the solution selects either both $i(j)$ and $i'(j)$ or none of these two tasks.

▶ **Lemma 27.** *Given a solution T' to the UFP-cover instance with $2k$ tasks. For each $j \in [n]$ we have that either $\{i(j), i'(j)\} \subseteq T'$ or $\{i(j), i'(j)\} \cap T' = \emptyset$.*

Suppose we are given a solution T' to the UFP instance with $2k$ tasks (for which hence Lemma 27 applies). Let J' be the set of indices j such that $i(j) \in T'$. Note that Lemma 27 implies that $|J'| = k$.

▶ **Lemma 28.** *We have that $\sum_{j \in J'} a_j = B$.*

Hence, we proved that the constructed UFP-cover instance has a solution with $2k$ tasks if and only if the k-subset sum instance is a yes-instance. This implies that UFP-cover is $W[1]$-hard when parameterized by the number of tasks in the optimal solution. This completes the proof of Theorem 23.

7 Conclusion and open questions

In this paper we presented a PAS for UFP-cover and showed that the problem is FPT under resource augmentation or if additionally the number of different task sizes are bounded by a parameter. It remains open whether the problem is FPT if *only* the number task sizes is bounded by a parameter, but not the number of tasks in the optimal solution. Also, we showed that UFP-cover is $W[1]$-hard. Our $W[1]$-hardness proof is based on a reduction from the k-subset sum problem, which can be solved in pseudopolynomial time $O(nB)$. Hence, it is open whether UFP-cover is FPT if the input data are polynomially bounded. Our PAS can be simplified in this setting, however, it crucially relies on the slack obtained by selecting ϵk additional tasks and thus does not solve the problem optimally in this case.

References

1 Susanne Albers, Sanjeev Arora, and Sanjeev Khanna. Page replacement for general caching problems. In *SODA*, volume 99, pages 31–40. Citeseer, 1999.

2 Antonios Antoniadis, Ruben Hoeksma, Julie Meißner, José Verschae, and Andreas Wiese. A QPTAS for the General Scheduling Problem with Identical Release Dates. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2017.31.

3 Nikhil Bansal, Amit Chakrabarti, Amir Epstein, and Baruch Schieber. A quasi-PTAS for unsplittable flow on line graphs. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC 2006)*, pages 721–729. ACM, 2006.

4 Nikhil Bansal, Ravishankar Krishnaswamy, and Barna Saha. On capacitated set cover problems. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 38–49. Springer, 2011.

5 Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph (Seffi) Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, 2001. doi:10.1145/502102.502107.

6 Jatin Batra, Naveen Garg, Amit Kumar, Tobias Mömke, and Andreas Wiese. New approximation schemes for unsplittable flow on a path. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*, pages 47–58, 2015. doi:10.1137/1.9781611973730.5.

7 Cristina Bazgan. Schémas d'approximation et Complexité Paramétrée, Rapport du stage (DEA). Technical report, Universitée Paris Sud, 1995.

8 Laszlo A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal*, 5(2):78–101, 1966.

9 Paul Bonsma, Jens Schulz, and Andreas Wiese. A constant-factor approximation algorithm for unsplittable flow on paths. *SIAM Journal on Computing*, 43:767–799, 2014.

10 Liming Cai and Xiuzhen Huang. Fixed-parameter approximation: Conceptual framework and approximability results. In *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, pages 96–108, 2006. doi:10.1007/11847250_9.

11 Robert D Carr, Lisa K Fleischer, Vitus J Leung, and Cynthia A Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *Proceedings of the 11th annual ACM-SIAM symposium on Discrete algorithms (SODA 2000)*, pages 106–115. Society for Industrial and Applied Mathematics, 2000.

12 Marco Cesati and Luca Trevisan. On the efficiency of polynomial time approximation schemes. *Inf. Process. Lett.*, 64(4):165–171, 1997. doi:10.1016/S0020-0190(97)00164-6.

13 Deeparnab Chakrabarty, Elyot Grant, and Jochen Könemann. On column-restricted and priority covering integer programs. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 355–368. Springer, 2010.

14 Yijia Chen, Martin Grohe, and Magdalena Grüber. On parameterized approximability. In *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, pages 109–120, 2006. doi:10.1007/11847250_10.

15 Maurice Cheung, Julián Mestre, David B Shmoys, and José Verschae. A primal-dual approximation algorithm for min-sum single-machine scheduling problems. *SIAM Journal on Discrete Mathematics*, 31(2):825–838, 2017.

16 M. Chrobak, G. Woeginger, K. Makino, and H. Xu. Caching is hard, even in the fault model. In *ESA*, pages 195–206, 2010.

17 Andrés Cristi and Andreas Wiese. Better approximations for general caching and UFP-cover under resource augmentation. Unpublished, 2019.

18 Rodney G. Downey, Michael R. Fellows, and Catherine McCartin. Parameterized approximation problems. In *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, pages 121–129, 2006. doi:10.1007/11847250_11.

19 Michael R Fellows and Neal Koblitz. Fixed-parameter complexity and cryptography. In *International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pages 121–131. Springer, 1993.

20 P. A. Franaszek and T. J. Wagner. Some distribution-free aspects of paging algorithm performance. *J. ACM*, 21(1):31–39, January 1974. doi:10.1145/321796.321800.

21 Fabrizio Grandoni, Tobias Mömke, Andreas Wiese, and Hang Zhou. To augment or not to augment: Solving unsplittable flow on a path by creating slack. In *Proc. of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, 2017. To appear.

22 Fabrizio Grandoni, Tobias Mömke, Andreas Wiese, and Hang Zhou. A (5/3+ε)-approximation for unsplittable flow on a path: placing small tasks into boxes. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC 2018)*, pages 607–619. ACM, 2018.

23 Wiebke Höhn, Julián Mestre, and Andreas Wiese. How unsplittable-flow-covering helps scheduling with job-dependent cost functions. *Algorithmica*, 80(4):1191–1213, 2018.

24 Sandy Irani. Page replacement with multi-size pages and applications to web caching. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 701–710. ACM, 1997.

25 Daniel Lokshtanov, Fahad Panolan, MS Ramanujan, and Saket Saurabh. Lossy kernelization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2017)*, pages 224–237. ACM, 2017.

26 Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008. doi:10.1093/comjnl/bxm048.

27 Andreas Wiese. A (1+epsilon)-approximation for unsplittable flow on a path in fixed-parameter running time. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, pages 67:1–67:13, 2017.

A Reduction from Generalized Caching in the fault model

A reduction from generalized caching in the fault model to UFP-cover was given in [1, 5]. For completeness we present the reduction here using our notation. In the fault model of general caching we are given a value $M \in \mathbb{N}$ that denotes the size of the cache and we are given a set of pages $\mathcal{P}$. Each page $q \in \mathcal{P}$ has a (not necessarily unit) size $s(q) \in \mathbb{N}$. Also we are given a set of requests $\mathcal{R}$ where each request $j \in \mathcal{R}$ is characterized by a time $t_j \geq 0$ and a page $q_j \in \mathcal{P}$ meaning that at time t_j the page q_j has to be present in the cache. The goal

is to decide at what times we bring each page into the cache in order to minimize the total number of these transfers, assuming that initially the cache is empty. We show here how to reduce this problem to UFP-cover with unit weights.

▶ **Lemma 29.** *Given an instance $(\mathcal{P}, \mathcal{R}, M)$ of general caching in the fault model, in polynomial time we can compute an instance (V, E, T, u) of UFP-cover such that for any solution to $(\mathcal{P}, \mathcal{R}, M)$ with cost C, there is a solution $T' \subseteq T$ to (V, E, T, u) with $|T'| = C$, and vice versa.*

Proof. W.l.o.g. we can restrict ourselves to solutions of $(\mathcal{P}, \mathcal{R}, M)$ where each page enters the cache only when it is requested and leaves the cache only right after it is requested, and to instances where each page is requested at least once and $M < \sum_{q\in\mathcal{P}} s(q)$. Thus, a solution is completely defined by deciding at each point in time whether we evict the page that was just requested or whether we keep it in the cache until it is requested again. We construct the path (V, E) by defining one edge $e(t)$ for each time t such that there is a request $j \in \mathcal{R}$ with $t_j = t$, and ordering the edges on the path by increasing values of t. For defining the tasks T, we initialize $T := \emptyset$. Then, for every page $q \in \mathcal{P}$ and every pair $j_1, j_2 \in R$ of consecutive requests of page q, we add a task $i(j_1, j_2)$ to T with size $p_{i(j_1,j_2)} = s(q)$. The subpath $P_{i(j_1,j_2)}$ is the one that starts in the right vertex of $e(t_{j_1})$ and ends in the left vertex of $e(t_{j_2})$. We define the demand of the edge $e(t)$ to be $u_{e(t)} = p(T_{e(t)}) - M + \sum_{j\in R:t_j=t} s(q_j)$ for every time t. We also add an extra edge e_0 at the left of E with capacity $u_{e_0} = \sum_{q\in\mathcal{P}} s(q)$ and we add a task i^*_q with $P_{i^*_q} = \{e_0\}$ and $p_{i^*_q} = s(q)$ for each page $q \in \mathcal{P}$. The cost of these tasks is exactly the total cost of loading each page into the cache once, i.e., the first time that the respective page is requested.

Given a solution to $(\mathcal{P}, \mathcal{R}, M)$, we construct a solution T' for (V, E, T, u) in the following way. For every page q and every pair of consecutive requests j_1, j_2 of page q, we add $i(j_1, j_2)$ to T' if and only if page q is evicted from (and therefore re-loaded into) the cache between t_{j_1} and t_{j_2}. We also add i^*_q to T' for all $q \in \mathcal{P}$. It is clear that $|T'|$ is exactly the number of times a page is brought into the cache in the original solution. We now check that T' is a feasible solution. Consider an edge $e(t) \in E$. Then $p(T_{e(t)} \setminus T')$ is the sum of sizes of the pages that are in the cache at time t that are *not* requested exactly at time t. The total size of all pages in the cache is at most M, so $p(T_{e(t)} \setminus T') + \sum_{j\in R:t_j=t} s(q_j) \leq M$. Then, as $p(T_{e(t)}) = p(T_{e(t)} \cap T') + p(T_{e(t)} \setminus T')$ and $u_{e(t)} = p(T_{e(t)}) - M + \sum_{j\in R:t_j=t} s(q_j)$ we conclude that $p(T' \cap T_{e(t)}) \geq u_{e(t)}$. Also it holds by construction that $p(T' \cap T_{e_0}) = u_{e_0}$.

Let now T' be a feasible solution to (V, E, T, u). Of course $i^*_q \in T'$ for all $q \in \mathcal{P}$. This accounts for the first time each page is brought into the cache. We construct a solution S' to $(\mathcal{P}, \mathcal{R}, M)$ as follows. For every page q and every pair of consecutive requests j_1, j_2 of page q we keep page q in the cache between t_{j_1} and t_{j_2} if and only if $i(j_1, j_2) \notin T'$. Thus, for each element in T' we have to bring a page into the cache once, and then the cost of the solution is exactly $|T'|$. We have to check that the size of the pages in the cache never exceeds M in S'. In fact, note that the total size of the pages in the cache at time t is $p(T_{e(t)} \setminus T') + \sum_{j\in R:t_j=t} s(q_j)$. But $p(T_{e(t)} \cap T') \geq u_{e(t)}$, and therefore,

$$
\begin{aligned}
& p(T_{e(t)} \cap T') \geq p(T_{e(t)}) - M + \sum_{j\in R:t_j=t} s(q_j) \\
\Leftrightarrow \quad & M \geq p(T_{e(t)}) + p(T_{e(t)} \cap T') + \sum_{j\in R:t_j=t} s(q_j) \\
\Leftrightarrow \quad & M \geq p(T_{e(t)} \setminus T') + \sum_{j\in R:t_j=t} s(q_j)\,.
\end{aligned}
$$

◀

Identifiability of Graphs with Small Color Classes by the Weisfeiler-Leman Algorithm

Frank Fuhlbrück
Institut für Informatik, Humboldt-Universität zu Berlin, Germany
fuhlbfra@informatik.hu-berlin.de

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

Oleg Verbitsky
Institut für Informatik, Humboldt-Universität zu Berlin, Germany
verbitsky@informatik.hu-berlin.de

Abstract

It is well known that the isomorphism problem for vertex-colored graphs with color multiplicity at most 3 is solvable by the classical 2-dimensional Weisfeiler-Leman algorithm (2-WL). On the other hand, the prominent Cai-Fürer-Immerman construction shows that even the multidimensional version of the algorithm does not suffice for graphs with color multiplicity 4. We give an efficient decision procedure that, given a graph G of color multiplicity 4, recognizes whether or not G is identifiable by 2-WL, that is, whether or not 2-WL distinguishes G from any non-isomorphic graph. In fact, we solve the more general problem of recognizing whether or not a given coherent configuration of maximum fiber size 4 is separable. This extends our recognition algorithm to directed graphs of color multiplicity 4 with colored edges.

Our decision procedure is based on an explicit description of the class of graphs with color multiplicity 4 that are *not* identifiable by 2-WL. The Cai-Fürer-Immerman graphs of color multiplicity 4 distinctly appear here as a natural subclass, which demonstrates that the Cai-Fürer-Immerman construction is not ad hoc. Our classification reveals also other types of graphs that are hard for 2-WL. One of them arises from patterns known as (n_3)-*configurations* in incidence geometry.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography; Theory of computation → Design and analysis of algorithms

Keywords and phrases Graph Isomorphism, Weisfeiler-Leman Algorithm, Cai-Fürer-Immerman Graphs, coherent Configurations

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.43

Related Version A full version of the paper is available at [17], `https://arxiv.org/abs/1907.02892`.

Funding *Oleg Verbitsky*: Supported by DFG grant KO 1053/8–1. On leave from the IAPMM, Lviv, Ukraine.

Acknowledgements We thank Ilia Ponomarenko and the anonymous referees for their numerous detailed comments and Daniel Neuen and Pascal Schweitzer for a useful discussion of multipede graphs. The third author is especially grateful to Ilia Ponomarenko for his patient and insightful guidance through the theory of coherent configurations.

1 Introduction

Over 50 years ago, Weisfeiler and Leman [34] described a natural combinatorial procedure that since then constantly plays a significant role in the research on the graph isomorphism problem. The procedure is now most often referred to as the 2-dimensional Weisfeiler-Leman algorithm (2-WL). It generalizes and improves the classical color refinement method (1-WL)

and has an even more powerful k-dimensional version (k-WL) for any $k > 2$. The original 2-dimensional version and the logarithmic-dimensional enhancement are important components in Babai's quasipolynomial-time isomorphism algorithm [4].

Even on its own, 2-WL is a quite powerful tool in isomorphism testing. For instance, it solves the isomorphism problem for several important graph classes, in particular, for interval graphs [16]. Also, it is successful for almost all regular graphs of a fixed degree [5]. On the other hand, not every pair of non-isomorphic graphs is distinguishable by 2-WL. For example, it cannot detect any difference between two non-isomorphic strongly regular graphs with the same parameters.

We call a graph G *amenable to* k-WL if the algorithm distinguishes G from any non-isomorphic graph. An efficient characterization of the class of graphs amenable to 1-WL is obtained by Arvind et al. in [2], where it is given also for vertex-colored graphs. Independently, Kiefer et al. [26] give an efficient criterion of amenability to 1-WL in a more general framework including also directed graphs with colored edges. Similar results for 2-WL are currently out of reach. A stumbling block here is the lack of understanding which strongly regular graphs are uniquely determined by their parameters. Note that a strongly regular graph is determined by its parameters up to isomorphism if and only if it is amenable to 2-WL.

A general strategy to approach a hard problem is to examine its complexity in the parameterized setting. We consider vertex-colored graphs with the *color multiplicity*, that is, the maximum number of equally colored vertices, as parameter. If this parameter is bounded, the graph isomorphism problem is known to be efficiently solvable. More specifically, it is solvable in time polynomial in the number of vertices and quasipolynomial in the color multiplicity [4, Corollary 4], and it is solvable in polylogarithmic parallel time [27]. Graph Isomorphism is known to be in the $\mathrm{Mod}_k\mathrm{L}$ hierarchy for any fixed color multiplicity [3], and even in $\oplus\mathrm{L} = \mathrm{Mod}_2\mathrm{L}$ for color multiplicity at most 5 [1].

Every graph of color multiplicity at most 3 is amenable to 2-WL (Immerman and Lander [24]). Starting from color multiplicity 4, the amenability concept is non-trivial: The prominent Cai-Fürer-Immerman construction [9] shows that for any k, there exist graphs with color multiplicity 4 that are not amenable to k-WL.

We design an efficient decision procedure that, given a graph G with color multiplicity 4, recognizes whether or not G is amenable to 2-WL. Note that an a priori upper complexity bound for this decision problem is coNP, as a consequence of the aforementioned fact that Graph Isomorphism for graphs of bounded color multiplicity is in P. From now on, amenability is meant with respect to 2-WL, unless stated otherwise.

We actually solve a much more general problem. 2-WL transforms an input graph G, possibly with colored vertices and directed and colored edges, into a *coherent configuration* $\mathcal{C}(G)$, which is called the *coherent closure* of G. The concept of a coherent configuration has been discovered independently in statistics [6] and algebra [22] and, playing an important role in diverse areas, has been developed to the subject of a rich theory; see a recent monograph [10], that we will use in this paper as a reference book. A coherent configuration $\mathcal{C}$ is called *separable* if the isomorphism type of $\mathcal{C}$ is determined by its regularity parameters in a certain strong sense; see the definition in Section 2. The separability of the coherent closure $\mathcal{C}(G)$ implies the amenability of the graph G. This was the approach undertaken in [16], where it was shown that the coherent closure of any interval graph is separable. Somewhat less obviously, the converse relation between amenability of G and separability of $\mathcal{C}(G)$ is also true: For every graph G,

$$G \text{ is amenable if and only if } \mathcal{C}(G) \text{ is separable;} \tag{1}$$

see Theorem 2.1 in Section 2. Equivalence (1) reduces the amenability problem for graphs to the separability problem for coherent configurations. This reduction works as well for

directed graphs with colored vertices and colored edges, that is, essentially for arbitrary binary relational structures. If G has color multiplicity b, then the maximum *fiber* size of $\mathcal{C}(G)$ is also bounded by b (see Section 2 for the definitions). While all coherent configurations with fibers of size at most 3 are known to be separable [10], the separability property for coherent configurations with fibers of size 4 is non-trivial, and our first result is this.

▶ **Theorem 1.1.** *The problem of deciding whether a given coherent configuration with maximum fiber size 4 is separable is solvable in* $\oplus$L.

Since $\oplus\mathrm{L} \subseteq \mathrm{NC}^2$ (which follows from the inclusion $\#L \subseteq \mathrm{NC}^2$ in [35]), Theorem 1.1 implies that the separability problem is solvable in parallel polylogarithmic time. Using the reduction (1), we obtain our result for graphs.

▶ **Theorem 1.2.** *The problem of deciding whether a given vertex-colored graph with maximum color multiplicity 4 is amenable to* 2-WL *is solvable in* P. *This holds true also for vertex- and edge-colored directed graphs.*

More precisely, the proof of Theorem 1.2 yields an algorithm deciding amenability of graphs of color multiplicity at most 4 with running time $O(n^{2+\omega})$, where $\omega < 2.373$ is the exponent of fast matrix multiplication [19]. Using randomization, the running time can be improved to $O(n^4 \log^2 n)$.

Theorems 1.1 and 1.2 are proved in Section 5. The proof is based on a combinatorial Cut-Down Lemma in Section 3, which reduces deciding separability of a coherent configuration $\mathcal{C}$ with maximum fiber size 4 to deciding separability of a subconfiguration $\mathcal{C}'$ of $\mathcal{C}$ belonging to a class of well-structured coherent configurations which we call *irredundant*. Separability of irredundant configurations is studied in Section 4, where it is recast as a question about a certain permutation group.

Our results have the following consequences, which we discuss in Section 6.

Highlighting the inherent structure of the Cai-Fürer-Immerman graphs. The essence of our proof of Theorem 1.2 is an explicit description of the class of graphs with color multiplicity 4 that are *not* amenable to 2-WL. The Cai-Fürer-Immerman graphs of color multiplicity 4 distinctly appear here as a natural subclass, which demonstrates that the Cai-Fürer-Immerman construction is not ad hoc. In a sense, the famous CFI gadget [9, Fig. 3] (or [25, Fig. 13.24]) appears in our analysis inevitably "by itself".[1]

While the CFI graphs have many automorphisms, Gurevich and Shelah [21] came up with a construction of (non-binary) *multipede* structures that are rigid and yet not identifiable by k-WL. Neuen and Schweitzer [30, 31] combined both approaches to construct *multipede graphs* and to give sufficient conditions ensuring that these graphs are not amenable to k-WL (see also a recent related paper [12]). The multipede graphs are vertex-colored and the results of [30, 31] make perfect sense if the color multiplicity is bounded by 4. An irredundant coherent configuration typically admits a natural representation by a multipede graph and vice versa; see Remark 6.6. Though non-amenability to k-WL for higher dimensions implies non-amenability to 2-WL, the results obtained in [12, 30, 31] and in our paper are incomparable as we provide both sufficient and *necessary* conditions for 2-WL-non-amenability.

[1] More precisely, this concerns a simplified version of the CFI gadget, where each vertex in a cubic pattern graph is replaced with a quadruple of new vertices and two quadruples are connected by edges directly, and not via two extra pairs of auxiliary vertices as in the original version; cf. Fig. 4. The simplified gadget appears in an algebraic analog of the CFI result by Evdokimov and Ponomarenko [15]; see also Fürer's survey paper [18]. This gadget comes out also in the *shrunken* multipede graphs [30].

More graphs hard for 2-WL. Our analysis reveals new types of non-amenable graphs. A particularly elegant construction is based on the well-studied (n_3)-configurations of lines and points [20, 32]. For example, the 7-point Fano plane and the 9-point Pappus configuration give rise to non-amenable graphs of color multiplicity 4 with, respectively, 28 and 36 vertices.

Classification of small graphs. Our amenability criteria are easy to apply in many cases. In particular, they imply that all graphs of color multiplicity 4 with no more than 15 vertices are amenable. Among graphs of color multiplicity 4 with 16 vertices there are 434 non-amenable graphs, which are split into 217 pairs of 2-WL indistinguishable non-isomorphic graphs.

All proofs omitted in this version of the paper can be found in [17].

2 Basic definitions and facts

Let V be a set, whose elements are called *points*. Let $\mathcal{C} = \{R_1, \dots, R_s\}$ be a partition of the Cartesian square V^2, that is, $\bigcup_{i=1}^{s} R_i = V^2$ and any two R_i and R_j are disjoint. An element R of $\mathcal{C}$ will be referred to as a *basis relation*. $\mathcal{C}$ is called a *coherent configuration* on $V = V(\mathcal{C})$ if it has the following properties:

(A) If a basis relation $R \in \mathcal{C}$ contains a loop vv, then all pairs in R are loops;
(B) For every $R \in \mathcal{C}$, the *transpose relation* $R^* = \{\, uv : vu \in R\}$ is also in $\mathcal{C}$.
(C) For every triple $R, S, T \in \mathcal{C}$, the number $p(uv) = |\{\, w : uw \in R,\, wv \in S\}|$ is the same for all $uv \in T$.

For a coherent configuration $\mathcal{C}$, the number $p(uv)$ in (C) does not depend on the choice of uv in T and is denoted by p^T_{RS}. The entries of this 3-dimensional matrix are called *intersection numbers* of $\mathcal{C}$.

Two coherent configurations $\mathcal{C}$ and $\mathcal{D}$ are *combinatorially isomorphic* if there is a bijection $\phi : V(\mathcal{C}) \to V(\mathcal{D})$, called a *combinatorial isomorphism* from $\mathcal{C}$ to $\mathcal{D}$, such that $\phi(R) \in \mathcal{D}$ for every $R \in \mathcal{C}$. We write $\mathcal{C} \cong_{comb} \mathcal{D}$ for this relationship. Here $\phi(R) = \{\, \phi(u)\phi(v) : uv \in R\}$.

Coherent configurations $\mathcal{C}$ and $\mathcal{D}$ are *algebraically isomorphic* if their 3-dimensional matrices of intersection numbers, p^T_{RS} and $p^{T'}_{R'S'}$, are isomorphic, that is, there is a bijection $f : \mathcal{C} \to \mathcal{D}$ such that

$$p^T_{RS} = p^{f(T)}_{f(R)f(S)}.$$

In this case we write $\mathcal{C} \cong_{alg} \mathcal{D}$. Such a bijection f is called an *algebraic isomorphism* from $\mathcal{C}$ to $\mathcal{D}$. Note that combinatorially isomorphic coherent configurations are also algebraically isomorphic. Indeed, any combinatorial isomorphism ϕ from $\mathcal{C}$ to $\mathcal{D}$ gives rise to the algebraic isomorphism f defined by $f(R) = \phi(R)$.

To allow a uniform treatment of ordinary graphs, vertex-colored graphs, and even edge-colored directed graphs, we formally define a *colored graph* G on a vertex set $V = V(G)$ as a function $c_G : V^2 \to C$ such that $c_G(vv) \neq c_G(uw)$ whenever $u \neq w$. For each color $c \in C$, the set $\{\, uv : c_G(uv) = c\}$ is called a *color class* of G. Two colored graphs G and H are isomorphic if there is a bijection $\phi : V(G) \to V(H)$ such that $c_H(\phi(u)\phi(v)) = c_G(uv)$ for all $u, v \in V(G)$. In the context of the isomorphism problem, we can always assume that

$$c_G(uv) = c_G(u'v') \text{ if and only if } c_G(vu) = c_G(v'u'), \tag{2}$$

that is, if arrows have the same color, then the inverse arrows must also be equally colored. This condition can be ensured by modifying the coloring as follows. Suppose that an arrow uv is colored *red* in G, and the inverse arrow vu is colored *blue*. Then uv is recolored a new color *redblue*, and vu is recolored a new color *bluered*. The new colored graph $\hat{G}$ satisfies the condition (2). Note that $\hat{G} \cong \hat{H}$ exactly when $G \cong H$.

We remark that ordinary graphs are covered by this setting as adjacency and non-adjacency can be seen as two distinct colors of vertex pairs. The vertex-colored graphs are covered as well as a vertex v can be seen as having color $c_G(vv)$. A set of vertices of the same color is referred to as *vertex color class*.

Given a colored graph G as input, the *2-dimensional Weisfeiler-Leman algorithm* (2-WL for short) iteratively computes colorings c^i_G of the Cartesian square V^2 for $V = V(G)$. Initially, $c^0_G = c_G$ and then,

$$c^{i+1}_G(uv) = c^i_G(uv) \mid \{\!\{ c^i_G(uw) \mid c^i_G(wv) \}\!\}_{w\in V},$$

where $\{\!\{ \}\!\}$ denotes the multiset and $\mid$ denotes the string concatenation (an appropriate encoding is assumed). Denote the partition of V^2 into the color classes of c^i_G by $\mathcal{R}^i_G$. Note that $\mathcal{R}^{i+1}_G$ refines $\mathcal{R}^i_G$. Let $t = t_G$ be the minimum number such that $\mathcal{R}^t_G = \mathcal{R}^{t-1}_G$. The algorithm terminates after the t-th color refinement round. It is easy to verify that $\mathcal{R}^t_G$ is a coherent configuration. Moreover, let $\mathcal{R}_G$ denote the partition of $V(G)^2$ into the color classes of G. It turns out that $\mathcal{R}^t_G$ coincides with the *coherent closure* of $\mathcal{R}_G$, which is the coarsest coherent configuration refining $\mathcal{R}_G$; see [10, Section 2.6.1]. We call this configuration the *coherent closure of the graph* G and denote it by $\mathcal{C}(G)$.

We say that colored graphs G and H are 2-WL *equivalent* and write $G \equiv_{\text{2-WL}} H$ if

$$\{\!\{ c^t_G(uv) \}\!\}_{uv\in V(G)^2} = \{\!\{ c^t_H(uv) \}\!\}_{uv\in V(H)^2} \tag{3}$$

for $t = t_G$ (equivalently, for $t = t_H$, or for all t).

Suppose that $G \equiv_{\text{2-WL}} H$. Equality (3) implies that there is a one-to-one map $f : \mathcal{C}(G) \to \mathcal{C}(H)$ preserving the 2-WL colors. Note that f is an algebraic isomorphism from $\mathcal{C}(G)$ to $\mathcal{C}(H)$. We, therefore, have the following diagram:

$$\begin{array}{ccc} G \cong H & \Longrightarrow & G \equiv_{\text{2-WL}} H \\ \Downarrow & & \Downarrow \\ \mathcal{C}(G) \cong_{comb} \mathcal{C}(H) & \Longrightarrow & \mathcal{C}(G) \cong_{alg} \mathcal{C}(H) \end{array}$$

We call a colored graph G *amenable (to* 2-WL*)* if 2-WL distinguishes G from any non-isomorphic graph H, that is, $G \equiv_{\text{2-WL}} H$ implies $G \cong H$.

A coherent configuration $\mathcal{C}$ is *separable* if every algebraic isomorphism from $\mathcal{C}$ to any coherent configuration $\mathcal{D}$ is induced by a combinatorial isomorphism from $\mathcal{C}$ to $\mathcal{D}$.

▶ **Theorem 2.1.** *A colored graph G is amenable if and only if its coherent closure $\mathcal{C}(G)$ is separable.*

Let $\mathcal{C}$ be a coherent configuration on the point set $V = V(\mathcal{C})$. A set of points $X \subseteq V$ is called a *fiber* of $\mathcal{C}$ if the set of loops $\{ xx : x \in X\}$ is a basis relation of $\mathcal{C}$. Denote the set of all fibers of $\mathcal{C}$ by $F(\mathcal{C})$. By Property (A) of a coherent configuration, $F(\mathcal{C})$ is a partition of V. Property (C) implies that, for every basis relation R of $\mathcal{C}$ there are, not necessarily distinct, fibers X and Y such that $R \subseteq X \times Y$. Thus, if $X, Y \in F(\mathcal{C})$, then the Cartesian product $X \times Y$ is split into basis relations of $\mathcal{C}$. We denote this partition by $\mathcal{C}[X, Y]$. If $X = Y$, we simplify notation to $\mathcal{C}[X] = \mathcal{C}[X, X]$. Note that $\mathcal{C}[X]$ is a coherent configuration on X, with X being its single fiber. Such coherent configurations are called *association schemes*. We will call $\mathcal{C}[X]$ a *cell* of $\mathcal{C}$.

All possible association schemes on at most 4 points are depicted in Figure 1. Basis relations are represented by undirected edges if they are equal to their transposes, and by arrows otherwise. Loops are omitted. The 4-point cells are named K_4, C_4, $\vec{C}_4$, and F_4 according to the graphs underlying their shapes. Here, $\vec{C}_4$ stands for the directed 4-cycle, and F_4 stands for the factorization of K_4 into three matchings $2K_2$.

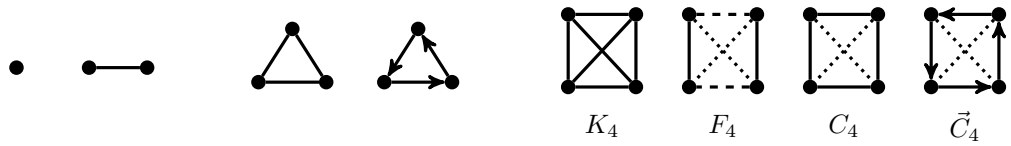


Figure 1 Cells $\mathcal{C}[X]$ on at most 4 points.

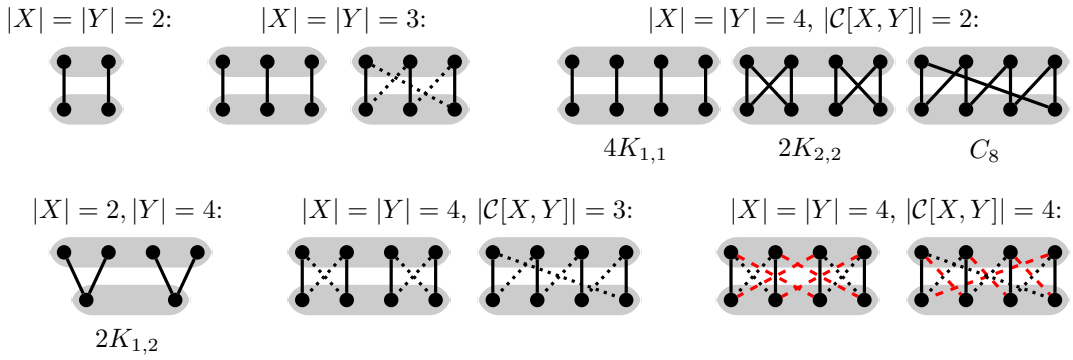


Figure 2 Non-uniform interspaces $\mathcal{C}[X,Y]$ for fibers X, Y with at most 4 points.

If $X \neq Y$, we call the partition $\mathcal{C}[X,Y]$ an *interspace* of $\mathcal{C}$. If $|\mathcal{C}[X,Y]| = 1$, that is, $X \times Y$ is a basis relation of $\mathcal{C}$, then the interspace $\mathcal{C}[X,Y]$ will be called *uniform*. If $R \in \mathcal{C}[X,Y]$, then the number of arrows in R from a point $x \in X$ is the same for each x in X. We call this number the *valency* of R and denote it by $d(R)$.

▶ **Lemma 2.2.** *Let $X, Y \in F(\mathcal{C})$. If $|X|$ and $|Y|$ are coprime, then $\mathcal{C}[X,Y]$ is uniform.*

Proof. Let R be a basis relation such that $R \subseteq \mathcal{C}[X,Y]$. Recall that the valency $d(R)$ is equal to the number of arrows in R from each point $x \in X$. Note also that the valency $d(R^*)$ of the transpose relation R^* is equal to the number of arrows in R to a point $y \in Y$; it does not depend on the choice of y. It follows that $d(R)|X| = |R| = d(R^*)|Y|$. Since $|X|$ and $|Y|$ are coprime, $d(R)$ is divisible by $|Y|$. Taking into account that $d(R) \leq |Y|$, we obtain the equality $d(R) = |Y|$. As a consequence, $R = X \times Y$. ◀

Thus, all interspaces $\mathcal{C}[X,Y]$ with $|X| = 1$ are uniform, and so are also interspaces with $|X| = 2$ and $|Y| = 3$. Figure 2 shows all non-uniform interspaces $\mathcal{C}[X,Y]$ with $|X|, |Y| \leq 4$. Here we depict pairs xy by undirected edges as they are implicitly ordered by the fibers. To facilitate visualization, one of the basis relations in each picture is missing as it is reconstructable from the others. We use the notation like $\mathcal{C}[X] \simeq C_4$, $\mathcal{C}[X,Y] \simeq 2K_{2,2}$ etc. to indicate the type of a cell or an interspace.

3 Cutting it down

Given a family of sets $\mathcal{P}$, we use $\mathcal{P}^\cup$ to denote the closure of $\mathcal{P}$ under unions. For any $U \in F(\mathcal{C})^\cup$ we let $\mathcal{C}[U]$ denote the set of all basis relations of a coherent configuration $\mathcal{C}$ contained in U^2. Note that $\mathcal{C}[U]$ is a coherent configuration on the point set U. Let $W = V \setminus U$. We say that $\mathcal{C}$ is the *direct sum* of coherent configurations $\mathcal{C}[U]$ and $\mathcal{C}[W]$ and write $\mathcal{C} = \mathcal{C}[U] \boxplus \mathcal{C}[W]$ if the interspace $\mathcal{C}[X,Y]$ is uniform for any two fibers $X, Y \in F(\mathcal{C})$ with $X \subseteq U$ and $Y \subseteq W$.

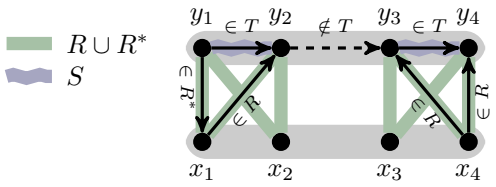


Figure 3 Proof of Lemma 3.4.

▶ **Lemma 3.1** (see [10, Corollary 3.2.8]). *Suppose that $\mathcal{C} = \mathcal{C}_1 \boxplus \mathcal{C}_2$. The coherent configuration $\mathcal{C}$ is separable if and only if both $\mathcal{C}_1$ and $\mathcal{C}_2$ are separable.*

Lemma 3.1 reduces the general separability problem to its restriction for *indecomposable* coherent configurations, that is, those configurations which cannot be split into a direct sum. Lemma 2.2 implies that an indecomposable coherent configuration of maximum fiber size at most 4 either has maximum fiber size at most 3 or has only fibers of size 4 or 2. We use the following known fact.

▶ **Lemma 3.2** (cf. [10, Exercise 3.7.20]). *Every coherent configuration $\mathcal{D}$ with maximum fiber size at most 3 is separable.*

Lemma 3.2, along with Lemmas 3.1 and 2.2, reduces the decision problem of whether a coherent configuration $\mathcal{C}$ with maximum fiber size 4 is separable to the case that $\mathcal{C}$ has fibers only of size 4 or 2. Next, we reduce the separability problem to instances having only fibers of size 4 and non-uniform interspaces of type $2K_{2,2}$.

Let M be a basis relation of a coherent configuration $\mathcal{C}$. Suppose that $M \in \mathcal{C}[X, Y]$ for distinct fibers X and Y, We call M a *matching* if M is irreflexive and both M and its transpose have valency 1, i.e., $d(M) = 1$ and $d(M^*) = 1$. This means that M determines a one-to-one correspondence between X and Y. If $M \in \mathcal{C}[X]$ for a fiber X, we additionally require that M is symmetric. In this case, M determines a partition of X into pairs of points.

The backward implication in Part 1 of the following lemma follows from [14, Lemma 9.4]. Part 2 applies, in particular, to the multipede graphs of color multiplicity at most 4. In this setting, Neuen and Schweitzer [30, Section 4.2] use exactly this shrinking operation in order to reduce the number of vertices in their construction of benchmark graphs challenging for practical isomorphism solvers.

▶ **Lemma 3.3** (Cut-Down Lemma). *Let $\mathcal{C}$ be a coherent configuration on $V = V(\mathcal{C})$.*

1. *Suppose that an interspace $\mathcal{C}[X, Y]$ contains a matching M. Then $\mathcal{C}$ is separable if and only if $\mathcal{C}[V \setminus X]$ is separable.*
2. *Suppose that $|V| > 2$, all fibers of $\mathcal{C}$ have size 4 or 2, and no interspace of $\mathcal{C}$ contains a matching. Let $X \in F(\mathcal{C})$ with $|X| = 2$. Under these conditions, $\mathcal{C}$ is separable if and only if $\mathcal{C}[V \setminus X]$ is separable.*
3. *Suppose that $|F(\mathcal{C})| > 3$, all fibers of $\mathcal{C}$ have size 4, and no interspace of $\mathcal{C}$ contains a matching. Let $\mathcal{C}[X, Y]$ be a C_8-interspace (see Figure 2). Under these conditions, $\mathcal{C}$ is separable if and only if $\mathcal{C}[V \setminus (X \cup Y)]$ is separable.*

We, therefore, focus on coherent configurations with non-uniform fibers only of type $2K_{2,2}$.

▶ **Lemma 3.4.** *Let $\mathcal{C}[X, Y] \simeq 2K_{2,2}$ and suppose that $\mathcal{C}[X, Y]$ contains a relation $R = \{x_1, x_2\} \times \{y_1, y_2\} \cup \{x_3, x_4\} \times \{y_3, y_4\}$. Then $\mathcal{C}[Y]$ contains the basis relation $S = \{y_1y_2, y_2y_1, y_3y_4, y_4y_3\}$.*

Proof. Let T be the basis relation of $\mathcal{C}[Y]$ containing the arrow y_1y_2. We have $T \subseteq S$ because $p^T_{R^*R} > 0$. For example, $y_2y_3 \notin T$ because y_1y_2 extends to $y_1x_1y_2$ and y_2y_3 cannot be extended to a triangle of this kind. On the other hand, $S \subseteq T$ because $p^R_{RT} > 0$. For example, $y_3y_4 \in T$ because otherwise, while x_1y_2 extends to $x_1y_1y_2$, the pair x_4y_4 could not be extended to a triangle of this kind; see Figure 3. ◀

In the context of Lemma 3.4, we say that $\mathcal{C}[X,Y]$ *determines* a matching in Y (namely $\{y_1y_2, y_2y_1, y_3y_4, y_4y_3\}$). Suppose that $\mathcal{C}[X,Y]$ determines a matching M in Y, and $\mathcal{C}[Z,Y]$ determines a matching M' in Y. We say that $\mathcal{C}[X,Y]$ and $\mathcal{C}[Z,Y]$ have a *direct connection at* Y if $M = M'$ (or are *directly connected at* Y). If $M \neq M'$, we say that $\mathcal{C}[X,Y]$ and $\mathcal{C}[Z,Y]$ have a *skewed connection at* Y (or are *askew connected at* Y).

▶ **Lemma 3.5** (Transitivity of direct $2K_{2,2}$-connections). *If $\mathcal{C}[X,Y] \simeq 2K_{2,2}$ and $\mathcal{C}[Z,Y] \simeq 2K_{2,2}$ are directly connected at Y, then either $\mathcal{C}[X,Z]$ contains a matching or $\mathcal{C}[X,Z] \simeq 2K_{2,2}$ and the connections between $\mathcal{C}[Z,X]$ and $\mathcal{C}[Y,X]$ at X and between $\mathcal{C}[X,Z]$ and $\mathcal{C}[Y,Z]$ at Z are direct.*

4 Irredundant configurations

The Cut-Down Lemma and the preceding analysis in Section 3 reduce our task to deciding separability of a coherent configuration $\mathcal{C}$ under the following three conditions:
(1) $\mathcal{C}$ is indecomposable,
(2) all fibers of $\mathcal{C}$ have size 4,
(3) every non-uniform interspace of $\mathcal{C}$ is of type $2K_{2,2}$.
A coherent configuration satisfying Conditions (1)–(3) will be called *irredundant*. Irredundant configurations are closely related to the *reduced Klein configurations* studied in [10, Section 4.1.2], but the two classes of coherent configurations are not identical. In particular, a reduced Klein configuration cannot contain C_4-cells.

We begin with noticing that, for irredundant configurations, every algebraic isomorphism f gives rise to a combinatorial isomorphism ϕ, even though ϕ does not need to induce f on the whole coherent configuration.

▶ **Lemma 4.1.** *Suppose that a coherent configuration $\mathcal{C}$ is irredundant. If f is an algebraic isomorphism from $\mathcal{C}$ to a coherent configuration $\mathcal{C}'$, then there exists a combinatorial isomorphism ϕ from $\mathcal{C}$ to $\mathcal{C}'$ such that ϕ induces f on each cell $\mathcal{C}[X]$ of $\mathcal{C}$.*

By Lemma 4.1, if a coherent configuration $\mathcal{C}$ is irredundant, then $\mathcal{C} \cong_{alg} \mathcal{C}'$ implies $\mathcal{C} \cong_{comb} \mathcal{C}'$. This has the following practical consequence: An irredundant configuration $\mathcal{C}$ is separable if and only if every algebraic *automorphism* of $\mathcal{C}$, i.e., an algebraic isomorphism from $\mathcal{C}$ to itself, is induced by a combinatorial automorphism of $\mathcal{C}$. Moreover, we call an algebraic automorphism f of $\mathcal{C}$ *strict* if f is the identity on each cell $\mathcal{C}[X]$ of $\mathcal{C}$.

▶ **Lemma 4.2.** *An irredundant coherent configuration $\mathcal{C}$ is separable if and only if every strict algebraic automorphism of $\mathcal{C}$ is induced by a combinatorial automorphism of $\mathcal{C}$.*

Let $\mathbb{A}(\mathcal{C})$ denote the set of strict algebraic automorphisms of $\mathcal{C}$. Our next task, which will be accomplished by Lemma 4.5 below, is to describe $\mathbb{A}(\mathcal{C})$ for a given irredundant coherent configuration $\mathcal{C}$. Call a permutation f on $\mathcal{C}$ *bound* if f is the identity on each cell, maps each interspace onto itself, and satisfies the condition $f(R^*) = f(R)^*$ for every basis relation R of $\mathcal{C}$. Since the last condition is obeyed by any algebraic isomorphism, every strict algebraic automorphism is bound. If $\mathcal{C}[X,Y] \simeq 2K_{2,2}$, then for a bound permutation f there are two

possibilities. Specifically, suppose that $\mathcal{C}[X,Y]$ partitions $X \times Y$ into two parts R_1 and R_2. We say that f *fixes* $\mathcal{C}[X,Y]$ if $f(R_i) = R_i$ and that f *switches* $\mathcal{C}[X,Y]$ if $f(R_i) = R_{3-i}$ for $i = 1, 2$. Note that, if f switches $\mathcal{C}[X,Y]$, then it switches also $\mathcal{C}[Y,X]$. Given a set S of pairs $\{X,Y\}$ such that $\mathcal{C}[X,Y]$ is non-uniform, let f_S denote the bijection from $\mathcal{C}$ onto itself which switches the interspace $\mathcal{C}[X,Y]$ as well as the interspace $\mathcal{C}[Y,X]$ for each $\{X,Y\} \in S$ and leaves the rest of $\mathcal{C}$ fixed. Thus, every bound permutation of $\mathcal{C}$ coincides with f_S for some S. Conversely, every f_S is a bound permutation, but not all f_S must be algebraic automorphisms.

Thus, we have to describe the class of those S for which f_S is an algebraic automorphism. Note that deciding whether a bound permutation f is a strict algebraic automorphism of $\mathcal{C}$ reduces to locally verifying this on all 3-fiber subconfigurations $\mathcal{C}[X \cup Y \cup Z]$. Therefore, we first consider coherent configurations with three fibers.

We call an irredundant configuration $\mathcal{C}$ *skew-connected* if $\mathcal{C}$ contains no directly connected interspaces.

▶ **Lemma 4.3.** *Let $\mathcal{C}$ be an irredundant coherent configuration with $F(\mathcal{C}) = \{X,Y,Z\}$ and f be a bound permutation of the set of basis relations of $\mathcal{C}$.*

1. *If $\mathcal{C}$ is skew-connected, then f is an algebraic automorphism of $\mathcal{C}$.*
2. *Suppose that $\mathcal{C}$ is not skew-connected. Then f is an algebraic automorphism of $\mathcal{C}$ if and only if f makes exactly two switches of interspaces (switching an interspace and its transpose is counted as a single switch).*

Let $\mathcal{C}$ be an irredundant coherent configuration. Like in the case of reduced Klein configurations [15], we define the *fiber graph* of $\mathcal{C}$, denoted by $F_{\mathcal{C}}$, as follows:

- The vertices of $F_{\mathcal{C}}$ are the fibers of $\mathcal{C}$, i.e., $V(F_{\mathcal{C}}) = F(\mathcal{C})$;
- Two fibers X and Y are adjacent in $F_{\mathcal{C}}$ if the interspace $\mathcal{C}[X,Y]$ is non-uniform.

Suppose that $\mathcal{C}[X,Y]$ is a non-uniform interspace. We define $D(X,Y)$ to be the set of fibers consisting of X, Y, and all Z such that $\mathcal{C}[Z,X]$ is non-uniform and directly connected with $\mathcal{C}[Y,X]$. Let $D_{\mathcal{C}}$ denote the family of all sets $D(X,Y)$ over non-uniform interspaces $\mathcal{C}[X,Y]$. We regard $D_{\mathcal{C}}$ as a hypergraph on $F(\mathcal{C})$ and call it the *hypergraph of direct connections* of $\mathcal{C}$.

The following properties of irredundant configurations are known for reduced Klein configurations [29]; see also [10, Lemma 4.1.18].

▶ **Lemma 4.4.**

1. *Every hyperedge of $D_{\mathcal{C}}$ is a clique in $F_{\mathcal{C}}$, and all interspace connections within this clique are direct.*
2. *Any two hyperedges of $D_{\mathcal{C}}$ have at most one common vertex.*

Proof. Lemma 3.5 implies that, if A and B are two fibers in $D(X,Y)$, then the interspace $\mathcal{C}[A,B]$ is non-uniform and $D(A,B) = D(X,Y)$. This implies both Parts 1 and 2. ◀

Given $C \in D_{\mathcal{C}}$ and a non-empty $U \subsetneq C$, let $S(U,C)$ be the set of all edges $\{X,Y\}$ in $F_{\mathcal{C}}$ such that $X \in U$ and $Y \in C \setminus U$. Using the notation f_S introduced above, we now define $f_{X,C} = f_{S(\{X\},C)}$ for $X \in C$.

▶ **Lemma 4.5.** *Suppose that a coherent configuration $\mathcal{C}$ is irredundant.*

1. *$f_S \in \mathbb{A}(\mathcal{C})$ if and only if, for every $C \in D_{\mathcal{C}}$, either the intersection $S \cap \binom{C}{2}$ is empty or it forms a spanning bipartite subgraph of $\binom{C}{2}$, where $\binom{C}{2}$ is considered the complete graph on the vertex set C.*
2. *$\mathbb{A}(\mathcal{C})$ is generated by the set of $f_{X,C}$ for all $C \in D_{\mathcal{C}}$ and all $X \in C$.*

Proof. *1.* For $C \in D_{\mathcal{C}}$, denote $S[C] = S \cap \binom{C}{2}$. By Lemma 4.4, $\{ S[C] \}_{C \in D_{\mathcal{C}}}$ is a partition of S. Therefore,

$$f_S = \prod_{C \in D_{\mathcal{C}}} f_{S[C]},$$

where the product is in the group of permutations of $\mathcal{C}$.

($\Longleftarrow$) It suffices to prove that each $f_{S[C]}$ is an algebraic automorphism of $\mathcal{C}$. It is enough to check that, for every triple of fibers X, Y, Z, the restriction of $f_{S[C]}$ to $\mathcal{C}[X \cup Y \cup Z]$ is an algebraic automorphism of $\mathcal{C}[X \cup Y \cup Z]$. If $|\{X, Y, Z\} \cap C| \leq 1$, then $f_{S[C]}$ is the identity on $\mathcal{C}[X \cup Y \cup Z]$. If $|\{X, Y, Z\} \cap C| = 2$, then Lemma 3.5 implies that $\mathcal{C}[X \cup Y \cup Z]$ is either decomposable or skew-connected. The former case is obvious, and in the latter case we are done by Part 1 of Lemma 4.3. If $\{X, Y, Z\} \subseteq C$, then $\mathcal{C}[X \cup Y \cup Z]$ cannot be skew-connected by the definition of $D_{\mathcal{C}}$ and the bipartiteness of $S[C]$ implies that $f_{S[C]}$ switches either two (up to transposing) or no interspaces between X, Y, Z. In this case we are done by Part 2 of Lemma 4.3.

($\Longrightarrow$) Let $C \in D_{\mathcal{C}}$ and suppose that $S[C]$ is non-empty. The claim is trivially true if $|C| = 2$, so we assume that $|C| \geq 3$. Let X, Y, and Z be three fibers in C. By assumption, the restriction of f_S to $\mathcal{C}[X \cup Y \cup Z]$ is an algebraic automorphism of $\mathcal{C}[X \cup Y \cup Z]$. By Part 2 of Lemma 4.3, f_S makes either none or exactly two switches in $\mathcal{C}[X \cup Y \cup Z]$. For $S[C]$, seen as a graph on the vertex set C, this implies that $S[C]$ does not contain any induced subgraph isomorphic to K_3 or to $K_2 + K_1$, where the latter is the graph with 3 vertices and 1 edge. A graph is $(K_2 + K_1)$-free if and only if it is complete multipartite. To see this, look at the complement and note that a graph is a vertex-disjoint union of cliques if and only if it does not contain an induced copy of a path on 3 vertices, the complement of $K_2 + K_1$. Thus, $S[C]$ is a complete multipartite graph. Since $S[C]$ is also triangle-free, it is bipartite.

2. Part 1 implies that $\mathbb{A}(\mathcal{C})$ is generated by the set of $f_{S(U,C)}$ for all $C \in D_{\mathcal{C}}$ and $\emptyset \neq U \subsetneq C$. Note that, if U is split into two non-empty parts U_1 and U_2, then $f_{S(U,C)} = f_{S(U_1,C)} \circ f_{S(U_2,C)}$ (as each interspace between U_1 and U_2 is switched twice). It follows that

$$f_{S(U,C)} = \prod_{X \in U} f_{X,C},$$

which implies the lemma. ◀

Separability test for irredundant coherent configurations

By Lemma 4.2, it suffices to check whether every strict algebraic automorphism $f \in \mathbb{A}(\mathcal{C})$ is induced by a combinatorial automorphism of $\mathcal{C}$. Note that $\mathbb{A}(\mathcal{C})$ forms a group of permutations of the set of basis relations of $\mathcal{C}$. Therefore, it is enough to choose an arbitrary generating set of $\mathbb{A}(\mathcal{C})$ and to check whether every f in this set is induced by a combinatorial automorphism. We use the generating set provided by Part 2 of Lemma 4.5. For deciding whether $f = f_{X,C}$ is induced by a combinatorial automorphism of $\mathcal{C}$, we construct a vertex-colored graph $G = G(\mathcal{C})$ whose automorphism group $\mathrm{Aut}(G)$ consists of all those combinatorial automorphisms of $\mathcal{C}$ which map every basis relation of $\mathcal{C}$ onto itself:

- $V(G) = V(\mathcal{C})$.
- The vertex color classes of G are exactly the fibers of $\mathcal{C}$.
- For two disjoint sets X and Y of vertices of G, let $G[X, Y]$ denote the subgraph of G on the vertex set $X \cup Y$ formed by the edges between a vertex in X and a vertex in Y. For each non-uniform interspace $\mathcal{C}[X, Y]$, we set $G[X, Y]$ to be one of the two $2K_{2,2}$ graphs underlying the basis relations of $\mathcal{C}[X, Y]$.

For each $X \in F(\mathcal{C})$, the subgraph $G[X]$ induced by G on X is defined as follows:

- If there are interspaces $\mathcal{C}[Y,X]$ and $\mathcal{C}[Z,X]$ with askew connection at X, then $G[X]$ is empty (in this case $\mathcal{C}[X] \simeq F_4$ by Lemma 3.4, and each matching relation on X will be anyway preserved by any automorphism of $G(\mathcal{C})$);
- Otherwise, $G[X]$ depends on $\mathcal{C}[X]$. We define $G[X]$ so that $\mathrm{Aut}(G[X])$ consists exactly of the combinatorial automorphisms of $\mathcal{C}[X]$ mapping each basis relation onto itself. Specifically,
  - if $\mathcal{C}[X] \simeq F_4$, then we put a matching $2K_2$ in $G[X]$ different from the one determined by some interspace $\mathcal{C}[Y,X]$ (note that at least one such interspace must exist);
  - if $\mathcal{C}[X] \simeq C_4$, we leave $G[X]$ empty (a matching on X is implicitly determined anyway);
  - if $\mathcal{C}[X] \simeq \vec{C}_4$, we have to put a directed 4-cycle in $G[X]$ coherently with the matching implicitly determined on X. To avoid making $G(\mathcal{C})$ a directed graph, we subdivide each edge of this cycle with two differently colored vertices in the direction given by $\vec{C}_4$. This costs us two new colors and four new vertices of each of these colors (which we put in $V(G)$ in addition to the vertices of $\mathcal{C}$).

For each pair (X,C) where $X \in C \in D_{\mathcal{C}}$, we now have to check whether the algebraic automorphism $f_{X,C}$ is induced by a combinatorial automorphism. A crucial fact is that the number of such pairs is polynomially bounded. Fix $G = G(\mathcal{C})$ as above and obtain a graph $G_{X,C}$ from G by complementing each bipartite subgraph $G[X,Y]$ spanned by the fiber X and a fiber Y in $C \setminus \{X\}$ (i.e., by connecting a vertex in X and a vertex in Y by an edge if and only if they are not adjacent in $G[X,Y]$). By construction, a combinatorial automorphism ϕ of $\mathcal{C}$ induces $f_{X,C}$ exactly when ϕ is an isomorphism of the graphs G and $G_{X,C}$. Thus, $f_{X,C}$ is induced by a combinatorial automorphism if and only if $G \cong G_{X,C}$. The last condition is efficiently verifiable [1] as the graphs G and $G_{X,C}$ are of color multiplicity 4.

5 Putting it together

5.1 Proof of Theorem 1.1

The Cut-Down Lemma (Lemma 3.3) and the preceding analysis of the irredundant case in Section 4 yield the following algorithm for recognizing whether or not a given coherent configuration $\mathcal{C}$ with fibers of size at most 4 is separable.

- Decompose $\mathcal{C}$ in the direct sum of indecomposable subconfigurations $\mathcal{C}_1, \ldots, \mathcal{C}_m$ and treat each $\mathcal{C}_i$ separately. By Lemma 3.1, $\mathcal{C}$ is separable if and only if every $\mathcal{C}_i$ is separable.
- Assume, therefore, that the input configuration $\mathcal{C}$ is indecomposable. If all fibers of $\mathcal{C}$ are of size at most 3, immediately decide that $\mathcal{C}$ is separable (see Lemma 3.2). Otherwise:
  - Remove all fibers of size 2 from $\mathcal{C}$.
  - Remove all pairs of fibers X and Y with $\mathcal{C}[X,Y] \simeq C_8$.
  - As long as $\mathcal{C}$ contains an interspace $\mathcal{C}[X,Y]$ with a matching, remove the fiber X from $\mathcal{C}$.
- If $\mathcal{C}$ becomes decomposable, split it into indecomposable components and handle each of them separately once again in the same way.
- If $\mathcal{C}$ becomes empty, decide that $\mathcal{C}$ is separable.
- Otherwise, we arrive at the case that $\mathcal{C}$ is irredundant and proceed as in Section 4.
- If all computational paths terminate with a positive decision, output "$\mathcal{C}$ *is separable*"; otherwise, output "$\mathcal{C}$ *is non-separable*".

Due to [1], each isomorphism test performed by the algorithm in Section 4 for an irredundant coherent configuration is implementable in $\oplus$L. A list of all subconfigurations to which this step is applied can be generated in logarithmic space [33]. Since $\mathrm{L}^{\oplus\mathrm{L}} = \oplus\mathrm{L}$ (see [8]), the whole algorithm can be implemented in $\oplus$L. Theorem 1.1 is proved.

5.2 Proof of Theorem 1.2

Suppose that the color multiplicity of G is bounded by 4. By Theorem 2.1, G is amenable to 2-WL if and only if its coherent closure $\mathcal{C}(G)$ is separable. Given G with n vertices, the coherent closure $\mathcal{C}(G)$ is computable in time $O(n^3 \log n)$ using the algorithm in [24]. Since G has color multiplicity at most 4, the coherent configuration $\mathcal{C}(G)$ has only fibers with at most 4 points. Therefore, we can decide separability of $\mathcal{C}(G)$ using the algorithm presented in Section 5.1. This algorithm reduces deciding separability for $\mathcal{C}(G)$ to deciding separability for a number of irredundant subconfigurations $\mathcal{C}_1, \dots, \mathcal{C}_t$ such that

$$\bigcup_{i=1}^{t} F(\mathcal{C}_i) \subseteq F(\mathcal{C}(G)). \tag{4}$$

Producing the list of coherent configurations $\mathcal{C}_1, \dots, \mathcal{C}_t$ has low time complexity. For each $i \le t$, we decide separability of $\mathcal{C}_i$ as described in Section 4. Specifically, $\mathcal{C}_i$ is separable if and only if the associated vertex-colored graph G^i is isomorphic to its modified version $H^i = G^i_{X,C}$ for every $X \in F(\mathcal{C}_i)$, where C is the hyperedge of $D_{\mathcal{C}_i}$ containing X. Denote the number of vertices in G^i by n_i. The isomorphism algorithm for graphs of color multiplicity 4 in [1] performs a low-cost conversion of the pair (G^i, H^i) into a system of $M_i < (n_i)^2$ linear equations with $N_i < n_i$ unknowns over the field $\mathbb{Z}_2$ such that $G^i \cong H^i$ if and only if the system is consistent.

Specifically, we here describe a simplified version of this general reduction suitable for any pair (G^i, H^i) arising from $\mathcal{C}_i$. Recall that $V(G^i) = V(H^i) = V(\mathcal{C}_i)$, and the vertex color classes of both G^i and H^i are exactly the fibers $X_1, \dots, X_s$ of $\mathcal{C}_i$, where each X_j has the same color both in G^i and H^i. For every X_j, we have $G^i[X_j] = H^i[X_j]$. Every non-empty bipartite subgraph $G^i[X_j, X_k]$ is isomorphic to $2K_{2,2}$. Moreover, $H^i[X_j, X_k]$ is equal either to $G^i[X_j, X_k]$ or to its bipartite complement.

Any isomorphism from G^i and H^i maps each vertex color class X_j onto itself. Moreover, if G^i and H^i are isomorphic, then there is an isomorphism ϕ preserving each of the three matchings on X_j for every j (recall that any isomorphism ϕ induces a strict algebraic automorphism of $\mathcal{C}_i$ and, hence, preserves the matchings in each X_j such that $\mathcal{C}_i[X_j] \simeq F_4$ and can be modified to obey this condition for each X_j such that $\mathcal{C}_i[X_j] \simeq C_4$). Denote the restriction of ϕ to X_j by ϕ_j. Thus, ϕ_j is one of the four elements of the *Klein group* $K(X_j)$, where

$$K(X) = \{\mathrm{id}_X, (x_1x_2)(x_3x_4), (x_1x_3)(x_2x_4), (x_1x_4)(x_2x_3)\}$$

for a 4-element set $X = \{x_1, x_2, x_3, x_4\}$. We say that a matching on X is *fixed* by a permutation from $K(X)$ if each of the two matched pairs is mapped onto itseld, and we say that it is *flipped* if the matched pairs are mapped onto each other. Denote the matchings on X_j by A_j, B_j, C_j. An element of $K(X_j)$ is uniquely determined by a triple (a_j, b_j, c_j), where $a_j = 1$ if A_j is flipped and $a_j = 0$ if A_j is fixed, and similarly for b_j and c_j. Since a non-identity element of $K(X)$ fixes one matching and flips the other two, we have

$$a_j \oplus b_j \oplus c_j = 0. \tag{E_j}$$

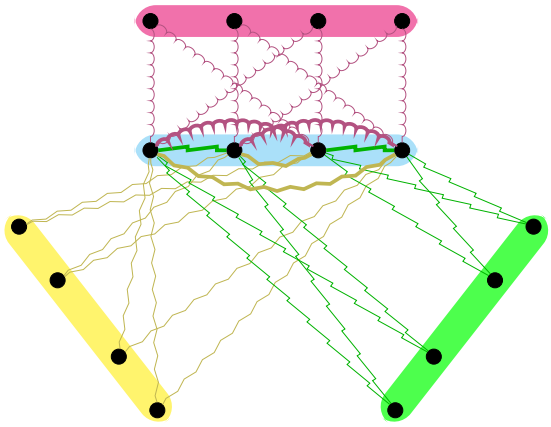


Figure 4 Three pairwise skew-connected interspaces and the matchings they induce.

Another constraint on ϕ_j is imposed by each pair X_j, X_k such that $G^i[X_j, X_k]$ is non-empty. To be specific, suppose that $G^i[X_j, X_k]$ determines the matching A_j in X_j and the matching B_k in X_k. Then

$$a_j \oplus b_k = d_{j,k}, \qquad (E_{j,k})$$

where $d_{j,k} = 0$ if $H^i[X_j, X_k]$ is equal to $G^i[X_j, X_k]$ and $d_{j,k} = 1$ if $H^i[X_j, X_k]$ is the bipartite complement of $G^i[X_j, X_k]$. It remains to notice that a set of permutations $\{\, \phi_j \,\}_{j=1}^{s}$ composing an isomorphism from G^i to H^i exists if and only if the system of equations consisting of (E_j) for all $j \le s$ and $(E_{j,k})$ for all non-empty $G^i[X_j, X_k]$ has a solution.

The rank of an $M \times N$ matrix over a finite field is computable in time $O(MN^{\omega-1})$, where $N \le M$ (see [7, 23]), or in randomized time $O(N^3 \log N)$ (see [11]). Since $|F(\mathcal{C}_i)| = n_i/4$, we can test separability of $\mathcal{C}_i$ in time $O((n_i)^{2+\omega})$ deterministically or in time $O((n_i)^4 \log n_i)$ using randomization. Taking into account the inequality $\sum_{i=1}^{t} n_i \le n$, which follows from (4), and the general inequality $\sum_{i=1}^{t} (n_i)^\alpha \le \left(\sum_{i=1}^{t} n_i\right)^\alpha$ for any real $\alpha \ge 1$, we conclude that separability of $\mathcal{C}(G)$ is decidable in deterministic time $O(n^{2+\omega})$ or in randomized time $O(n^4 \log^2 n)$, where an extra logarithmic factor corresponds to the number of repetitions needed to make the failure probability an arbitrarily small constant.

6 Examples of graphs hard for 2-WL

In Section 4 we constructed a vertex-colored graph $G(\mathcal{C})$ underlying the structure of an irredundant coherent configuration $\mathcal{C}$. As easily seen, if $\mathcal{C}$ contains no $\vec{C}_4$-cells, then $\mathcal{C}$ is the coherent closure of $G(\mathcal{C})$. Theorem 2.1, therefore, implies that every non-separable $\mathcal{C}$ of this kind yields a graph not identifiable by 2-WL.

6.1 The Cai-Fürer-Immerman construction

Skew-connected irredundant coherent configurations correspond to the seminal CFI construction. Indeed, Part 3 of the following theorem is reminiscent of [9, Lemma 6.2]. It shows that, if all connections between non-uniform interspaces are skewed and exactly 3 non-uniform interspaces emanate from each fiber (which is a direct analog of the famous CFI gadget; see Fig. 4), then the coherent configuration is non-separable.

As usually, $\delta(G)$ denotes the minimum degree of a vertex in the graph G. We also use the notation introduced in Section 4.

▶ **Theorem 6.1.** *If $\mathcal{C}$ is skew-connected, then the following is true.*

1. $\mathbb{A}(\mathcal{C}) = \{ f_S : S \subseteq E(F_{\mathcal{C}})\}$.
2. *If $\delta(F_{\mathcal{C}}) \leq 2$, then every f_S is induced by a combinatorial automorphism of $\mathcal{C}$ and, hence, $\mathcal{C}$ is separable.*
3. *If $\delta(F_{\mathcal{C}}) = 3$, i.e., $F_{\mathcal{C}}$ is a regular graph of degree 3, then f_S is induced by a combinatorial automorphism of $\mathcal{C}$ exactly when $|S|$ is even. Hence, $\mathcal{C}$ is non-separable in this case.*

6.2 Examples coming from incidence geometry

Lemma 4.4 says exactly that, if $\mathcal{C}$ is irredundant, then the hypergraph of direct connections $D_{\mathcal{C}}$ is a configuration known in incidence geometry [13, 28] as a partial linear space. Here vertices of the hypergraph are interpreted as *points* and hyperedges as *lines*, even though not every partial linear space admits a geometric realization. More precisely, a hypergraph is called *linear* if every two hyperedges have at most one common vertex. A *partial linear space* is a linear hypergraph with each hyperedge of size at least 2.

A relationship between partial linear spaces and reduced Klein configurations was noticed in [10, Corollary 4.1.19]. The following lemma implies that, if the hypergraph $D_{\mathcal{C}}$ is 3-regular, that is, every fiber of $\mathcal{C}$ belongs to exactly three cliques in $D_{\mathcal{C}}$, then the coherent configuration $\mathcal{C}$ is uniquely determined by $D_{\mathcal{C}}$. Moreover, every partial linear space D where each point belongs to at most 3 lines is the hypergraph of direct connections for some coherent configuration. As a consequence, partial linear spaces are a rich source of templates for constructing coherent configurations.

Specifically, a hypergraph is called *connected* if its Gaifman graph is connected. The degree of a vertex v in a hypergraph H is the number of hyperedges of H containing v. Similarly to graphs, $\Delta(H)$ (resp., $\delta(H)$) denotes the maximum (resp., minimum) degree of a vertex in the hypergraph H. Note that $1 \leq \delta(D_{\mathcal{C}}) \leq \Delta(D_{\mathcal{C}}) \leq 3$.

▶ **Lemma 6.2.**

1. *Let $\mathcal{C}$ be an irredundant configuration. If $\mathcal{C} \cong_{alg} \mathcal{C}'$, then $D_{\mathcal{C}} \cong D_{\mathcal{C}'}$, where $\cong$ denotes isomorphism of hypergraphs.*
2. *Under the condition $\delta(D_{\mathcal{C}}) \geq 2$, $D_{\mathcal{C}} \cong D_{\mathcal{C}'}$ implies that $\mathcal{C} \cong_{comb} \mathcal{C}'$.*
3. *For any connected partial linear space D with $\Delta(D) \leq 3$ there is an irredundant configuration $\mathcal{C}$ such that $D_{\mathcal{C}} \cong D$.*

Proof.

1. This part follows from the fact that an algebraic isomorphism respects fibers, non-uniformity of interspaces, and direct connections of interspaces.
2. Let $h : F(\mathcal{C}) \to F(\mathcal{C}')$ be an isomorphism from the hypergraph $D_{\mathcal{C}}$ to the hypergraph $D_{\mathcal{C}'}$. Based on h, we define a bijection $\bar{h}$ from the set of all matchings of $\mathcal{C}$ to the set of all matchings of $\mathcal{C}'$. Consider a fiber $X \in F(\mathcal{C})$. Let C_1 and C_2 be two hyperedges of $D_{\mathcal{C}}$ containing X. All interspaces $\mathcal{C}[Y, X]$ for $Y \in C_1$ determine the same matching in the cell $\mathcal{C}[X]$, which we denote by M_1. All interspaces $\mathcal{C}[Y, X]$ for $Y \in C_2$ determine a matching M_2, different from M_1. Denote the third matching in $\mathcal{C}[X]$ by M_3. Similarly, the interspaces $\mathcal{C}'[Y', h(X)]$ for $Y' \in h(C_1)$ determine a matching M_1', and the interspaces $\mathcal{C}'[Y', h(X)]$ for $Y' \in h(C_2)$ determine a matching $M_2' \neq M_1'$ in $\mathcal{C}'[h(X)]$. Denote the third matching in $\mathcal{C}'[h(X)]$ by M_3' and set $\bar{h}(M_i) = M_i'$ for $i = 1, 2, 3$. Let ψ_X be a bijection from X onto $h(X)$ such that $\phi_X(M) = \bar{h}(M)$ for each matching M in $\mathcal{C}[X]$. Combining all ψ_X over $X \in F(\mathcal{C})$, we obtain a bijection from $V(\mathcal{C})$ onto $V(\mathcal{C}')$ which is a combinatorial isomorphism from $\mathcal{C}$ to $\mathcal{C}'$.

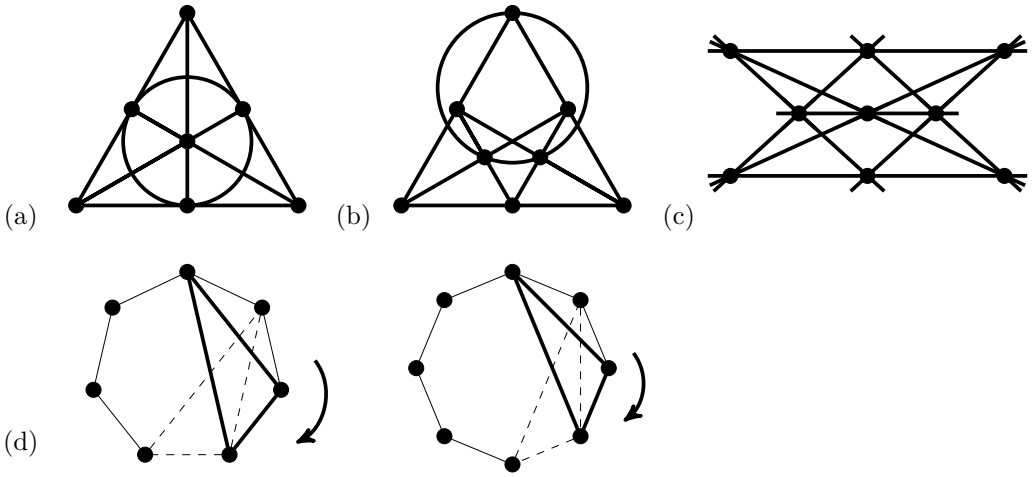


Figure 5 (a) The Fano plane. (b) The Möbius-Kantor configuration. One 3-point "line" in (a) and in (b) is drawn as a circle. (c) The Pappus configuration. (d) Construction of the cyclic versions D_7 and D_8 of the Fano and the Möbius-Kantor configurations.

3. Given D, we construct $\mathcal{C}$ as follows. Each point p of D gives rise to a 4-point fiber X_p in $\mathcal{C}$, with the cell $\mathcal{C}[X_p]$ being of type F_4. With each hyperedge C of D containing p, we associate a matching relation $M_{p,C}$ in $\mathcal{C}[X_p]$ such that $M_{p,C} \neq M_{p,C'}$ if $C \neq C'$. For each pair of points p and q in the same hyperedge C, we make the interspace $\mathcal{C}[X_p, X_q]$ non-uniform so that it determines the matching $M_{p,C}$ in $\mathcal{C}[X_p]$ and the matching $M_{q,C}$ in $\mathcal{C}[X_q]$. ◀

We now give examples of non-amenable graphs (or, equivalently, examples of non-separable irredundant coherent configurations) arising from classical incidence geometries. Partial linear spaces with n points where every line contains exactly 3 points and every point is incident to exactly 3 lines are known as (n_3)-*configurations*; see [20, 32]. There is no (n_3)-configuration for $n \leq 6$. There are a unique (7_3)-configuration, namely the *Fano plane*, and a unique (8_3)-configuration, namely the *Möbius-Kantor configuration*; see Figure 5. We denote the coherent configurations whose hypergraphs of direct connections are isomorphic to these two line-point configurations by $\mathcal{C}_{\text{Fano}}$ and $\mathcal{C}_{\text{MK}}$ respectively. These configurations exist by Part 3 of Lemma 6.2 and are unique by Part 2 of this lemma.

▶ **Theorem 6.3.** *$\mathcal{C}_{\text{Fano}}$ is non-separable, and $\mathcal{C}_{\text{MK}}$ is separable.*

Thus, the 7-point Fano plane gives rise to a graph of color multiplicity 4 with 28 vertices which is not identifiable by 2-WL and is not a CFI graph. Theorem 6.3 is actually a particular instance of a more general statement.

Let $n \geq 7$. The *cyclic* (n_3)-*configurations* D_n is constructed as follows [20, Section 2.1]. Let F_n be the Cayley graph of $\mathbb{Z}_n$ with the difference set $\{\pm 1, \pm 2, \pm 3\}$ and D_n be the hypergraph formed by 3-cliques $\{i, i+2, i+3\}$ in F_n, where $i \in \mathbb{Z}_n$. It is straightforward to see that D_n is really an (n_3)-configuration. By the uniqueness of (n_3)-configurations for $n = 7, 8$ (see, e.g., [32, Theorem 5.13]), the Fano plane is isomorphic, as a hypergraph, to D_7, and the Möbius-Kantor configuration is isomorphic to D_8. Let $\mathcal{C}_n$ be the coherent configuration constructed from D_n as in the proof of Part 3 of Lemma 6.2. This lemma implies that $\mathcal{C}_{\text{Fano}} \cong_{comb} \mathcal{C}_7$ and $\mathcal{C}_{\text{MK}} \cong_{comb} \mathcal{C}_8$. Thus, Theorem 6.3 is equivalent to the statement that $\mathcal{C}_n$ is non-separable if $n = 7$ and separable if $n = 8$.

▶ **Theorem 6.4.** *Let $n \geq 7$. The coherent configuration $\mathcal{C}_n$ is non-separable if and only if n is a multiple of 7.*

▶ Remark 6.5. There are exactly three (9_3)-configurations [20, 32]. The most famous of them is the *Pappus configuration* shown in Fig. 5(c). Computer-assisted verification shows that the corresponding 36-point coherent configuration is non-separable. Of the other two (9_3)-configurations, one is the cyclic (9_3)-configuration defined above, and the other is obtained similarly by rotating the triangle $\{0, 3, 4\}$ (instead of $\{0, 2, 3\}$) in $\mathbb{Z}_9$. These two produce separable coherent configurations.

▶ Remark 6.6. Curiously, Lemma 6.2 reveals a connection between irredundant coherent configurations and the multipede graphs introduced by Neuen and Schweitzer in [30]. Let $\mathcal{C}$ be an irredundant configuration and assume for the hypergraph of direct connections of $\mathcal{C}$ that $\delta(D_{\mathcal{C}}) = 3$. Consistently with the notation in [30], denote the incidence graph of the hypergraph $D_{\mathcal{C}}$ by $G = G(V, W)$, where $V = F(\mathcal{C})$ is the vertex set of $D_{\mathcal{C}}$, i.e., the set of all fibers of $\mathcal{C}$, and W is the set of the hyperedges of $D_{\mathcal{C}}$, i.e., the cliques of directly connected fibers. Two vertices $v \in V$ and $w \in W$ are adjacent in G if v belongs to w. Thus, every vertex in V has degree 3 in G. Any such bipartite graph G determines a multipede graph denoted in [30] by $R(G)$. This is a vertex-colored graph with vertex classes of size 4 and 2. Since we started from an irredundant configuration $\mathcal{C}$, the coloring of $R(G)$ is not refinable by 2-WL, and each color class of $R(G)$ stays as a fiber in the coherent closure $\mathcal{C}(R(G))$. Let $\mathcal{C}'$ be the coherent configuration obtained from $\mathcal{C}(R(G))$ by cutting down all fibers of size 2 (cf. Part 2 of Lemma 3.3). Lemma 6.2 implies that $\mathcal{C}'$ is combinatorially isomorphic to $\mathcal{C}$.

6.3 Small graphs

Our amenability criteria behind Theorem 1.2 are rather practical, which is illustrated by the following result.

▶ **Theorem 6.7.**

1. *All graphs of color multiplicity 4 with at most 15 vertices are amenable.*
2. *Up to isomorphism and color renaming, there are 434 non-amenable graphs of color multiplicity 4 with 16 vertices. More precisely, the number of non-trivial $\equiv_{2\text{-WL}}$-equivalence classes is 217, each consisting of exactly two non-isomorphic graphs.*

7 Conclusion and further questions

Our results raise questions about the parameterized complexity of recognizing the amenability of a given graph with the largest color multiplicity m taken as the parameter. The problem is trivial for $m = 3$ due to [24]. We show that it is solvable in polynomial time for $m = 4$. Our analysis surely generalizes to a few subsequent values of m. For any fixed m, the problem is in coNP, and it is open whether it is in P if m is large.

Another open question, that naturally arises in light of Theorem 1.2, concerns the next dimension of the Weisfeiler-Leman algorithm: Can the amenability to 3-WL be decided in polynomial time on input graphs with the largest color multiplicity 4?

The *WL dimension* of a graph G is defined as the minimum k such that G is amenable to k-WL. The graphs with large WL dimension are of significant interest in the study of the graph isomorphism problem. When we seek such graphs among graphs with color multiplicity 4, note that they must be at least non-amenable to 2-WL. Cai, Fürer, and Immerman [9] give conditions ensuring linear WL dimension for graphs whose coherent closure is, in our

terminology, skew-connected. Further such conditions are identified by the line of research [12, 21, 30, 31]. Can we achieve high WL dimension in other cases, say, for graphs whose coherent closure corresponds to a line-point (n_3)-configuration (see Section 6)?

References

1 Vikraman Arvind and Johannes Köbler. On hypergraph and graph isomorphism with bounded color classes. In *23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS'06)*, volume 3884 of *Lecture Notes in Computer Science*, pages 384–395. Springer, 2006. doi:10.1007/11672142_31.

2 Vikraman Arvind, Johannes Köbler, Gaurav Rattan, and Oleg Verbitsky. Graph isomorphism, color refinement, and compactness. *Computational Complexity*, 26(3):627–685, 2017. doi:10.1007/s00037-016-0147-6.

3 Vikraman Arvind, Piyush P. Kurur, and T. C. Vijayaraghavan. Bounded color multiplicity graph isomorphism is in the #L hierarchy. In *20th Annual IEEE Conference on Computational Complexity (CCC'05)*, pages 13–27. IEEE Computer Society, 2005. doi:10.1109/CCC.2005.7.

4 László Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC'16)*, pages 684–697, 2016. doi:10.1145/2897518.2897542.

5 Béla Bollobás. Distinguishing vertices of random graphs. *Annals of Discrete Mathematics*, 13:33–49, 1982. doi:10.1016/S0304-0208(08)73545-X.

6 R. C. Bose and Dale M. Mesner. On linear associative algebras corresponding to association schemes of partially balanced designs. *Ann. Math. Statist.*, 30:21–38, 1959. doi:10.1214/aoms/1177706356.

7 James R. Bunch and John E. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Math. Comp.*, 28:231–236, 1974. doi:10.2307/2005828.

8 Gerhard Buntrock, Carsten Damm, Ulrich Hertrampf, and Christoph Meinel. Structure and importance of logspace-mod classes. *Mathematical systems theory*, 25(3):223–237, 1992. doi:10.1007/BF01374526.

9 Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4):389–410, 1992. doi:10.1007/BF01305232.

10 Gang Chen and Ilia Ponomarenko. *Coherent configurations*. Central China Normal University Press, 2019.

11 Ho Yee Cheung, Tsz Chiu Kwok, and Lap Chi Lau. Fast matrix rank algorithms and applications. *J. ACM*, 60(5):31:1–31:25, 2013. doi:10.1145/2528404.

12 Anuj Dawar and Kashif Khan. Constructing hard examples for graph isomorphism. *J. Graph Algorithms Appl.*, 23(2):293–316, 2019. doi:10.7155/jgaa.00492.

13 Bart de Bruyn. *An introduction to incidence geometry*. Front. Math. Basel: Birkhäuser/Springer, 2016.

14 S. Evdokimov and I. Ponomarenko. Characterization of cyclotomic schemes and normal schur rings over a cyclic group. *St. Petersburg Math. J.*, 14(2):189–221, 2003.

15 Sergei Evdokimov and Ilia Ponomarenko. On highly closed cellular algebras and highly closed isomorphisms. *Electr. J. Comb.*, 6, 1999. URL: http://www.combinatorics.org/Volume_6/Abstracts/v6i1r18.html.

16 Sergei Evdokimov, Ilia Ponomarenko, and Gottfried Tinhofer. Forestal algebras and algebraic forests (on a new class of weakly compact graphs). *Discrete Mathematics*, 225(1-3):149–172, 2000. doi:10.1016/S0012-365X(00)00152-7.

17 Frank Fuhlbrück, Johannes Köbler, and Oleg Verbitsky. Identifiability of graphs with small color classes by the Weisfeiler-Leman algorithm. Technical report, https://arxiv.org/abs/1907.02892, 2019.

18 Martin Fürer. On the combinatorial power of the Weisfeiler-Lehman algorithm. In *Algorithms and Complexity — 10th International Conference (CIAC'17) Proceedings*, volume 10236 of *Lecture Notes in Computer Science*, pages 260–271, 2017. doi:10.1007/978-3-319-57586-5_22.

19 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. of the Int. Symposium on Symbolic and Algebraic Computation (ISSAC'14)*, pages 296–303. ACM, 2014. doi:10.1145/2608628.2608664.

20 Branko Grünbaum. *Configurations of points and lines*, volume 103 of *Grad. Stud. Math.* Providence, RI: American Mathematical Society (AMS), 2009.

21 Yuri Gurevich and Saharon Shelah. On finite rigid structures. *J. Symb. Log.*, 61(2):549–562, 1996. doi:10.2307/2275675.

22 D.G. Higman. Finite permutation groups of rank 3. *Math. Z.*, 86:145–156, 1964. doi:10.1007/BF01111335.

23 Oscar H. Ibarra, Shlomo Moran, and Roger Hui. A generalization of the fast LUP matrix decomposition algorithm and applications. *J. Algorithms*, 3(1):45–56, 1982. doi:10.1016/0196-6774(82)90007-4.

24 N. Immerman and E. Lander. Describing graphs: A first-order approach to graph canonization. In *Complexity Theory Retrospective*, pages 59–81. Springer, 1990.

25 Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999. doi:10.1007/978-1-4612-0539-5.

26 Sandra Kiefer, Pascal Schweitzer, and Erkal Selman. Graphs identified by logics with counting. In *Proceedings of the 40th International Symposium on Mathematical Foundations of Computer Science (MFCS'15)*, volume 9234 of *Lecture Notes in Computer Science*, pages 319–330. Springer, 2015. doi:10.1007/978-3-662-48057-1_25.

27 Eugene M. Luks. Parallel algorithms for permutation groups and graph isomorphism. In *27th Annual Symposium on Foundations of Computer Science (FOCS'86)*, pages 292–302. IEEE Computer Society, 1986. doi:10.1109/SFCS.1986.39.

28 Klaus Metsch. *Linear spaces with few lines*, volume 1490 of *Lect. Notes Math.* Berlin etc.: Springer-Verlag, 1991.

29 Mikhail Muzychuk and Ilya Ponomarenko. On quasi-thin association schemes. *J. Algebra*, 351(1):467–489, 2012.

30 Daniel Neuen and Pascal Schweitzer. Benchmark graphs for practical graph isomorphism. In *25th Annual European Symposium on Algorithms (ESA'17)*, volume 87 of *LIPIcs*, pages 60:1–60:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.60.

31 Daniel Neuen and Pascal Schweitzer. An exponential lower bound for individualization-refinement algorithms for graph isomorphism. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC'18)*, pages 138–150. ACM, 2018. doi:10.1145/3188745.3188900.

32 Tomaž Pisanski and Brigitte Servatius. *Configurations from a graphical viewpoint*. Birkhäuser Adv. Texts, Basler Lehrbüch. New York, NY: Birkhäuser, 2013.

33 Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4):17:1–17:24, 2008. doi:10.1145/1391289.1391291.

34 B.Yu. Weisfeiler and A.A. Leman. The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Ser. 2*, 9:12–16, 1968. English translation is available at https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf.

35 Carme Álvarez and Birgit Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107(1):3–30, 1993. doi:10.1016/0304-3975(93)90252-O.

Better Approximations for General Caching and UFP-Cover Under Resource Augmentation

Andrés Cristi
Universidad de Chile, Chile
andres.cristi@ing.uchile.cl

Andreas Wiese
Universidad de Chile, Chile
awiese@dii.uchile.cl

Abstract

In the Unsplittable Flow on a Path Cover (UFP-cover) problem we are given a path with a demand for each edge and a set of tasks where each task is defined by a subpath, a size and a cost. The goal is to select a subset of the tasks of minimum cost that together cover the demand of each edge. This problem models various resource allocation settings and also the general caching problem. The best known polynomial time approximation ratio for it is 4 [Bar-Noy et al., STOC 2000]. In this paper, we study the resource augmentation setting in which we need to cover only a slightly smaller demand on each edge than the compared optimal solution. If the cost of each task equals its size (which represents the natural bit-model in the related general caching problem) we provide a polynomial time algorithm that computes a solution of *optimal* cost. We extend this result to general caching and to the packing version of Unsplittable Flow on a Path in their respective natural resource augmentation settings. For the case that the cost of each task equals its "area", i.e., the product of its size and its path length, we present a polynomial time $(1+\epsilon)$-approximation for UFP-cover. If additionally the edge capacities are in a constant range we compute even a solution of optimal cost and also obtain a PTAS *without* resource augmentation.

2012 ACM Subject Classification Theory of computation → Packing and covering problems

Keywords and phrases General caching, unsplittable flow cover, approximation algorithm, resource augmentation

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.44

Funding *Andrés Cristi*: Supported by CONICYT-PFCHA/Doctorado Nacional/2018-21180347.
Andreas Wiese: Partially supported by FONDECYT Regular grant 1170223.

1 Introduction

Caching is one of the most classical problems in computer science. We are given a value $M \in \mathbb{N}$ that denotes the size of the cache and we are given a set of unit size pages $\mathcal{P}$. Also, we are given a set of requests $\mathcal{R}$ where each request $j \in \mathcal{R}$ is characterized by a time $t_j \geq 0$ and a page $q_j \in \mathcal{P}$ meaning that at time t_j the page q_j has to be present in the cache. The goal is to decide at what times we bring each page into the cache in order to minimize the total number of these transfers, assuming that initially the cache is empty. Caching is a very well-studied problem in computer science with research on it dating back to the 1960s, see e.g., [8, 16, 9] and references therein. It admits a polynomial time algorithm in the offline setting [14] and in the online case there are several deterministic M-competitive algorithms [21, 9] and a randomized $O(\log M)$-competitive algorithm [15].

A natural generalization is the *general caching* problem where additionally each page $i \in \mathcal{P}$ has a (not necessarily unit) size $p_i \in \mathbb{N}$ and additionally a cost $w_i \in \mathbb{N}$ that we have to pay each time we bring i into the cache, the goal being to minimize the total cost. General caching can be modeled by a covering problem which turns out to be the natural covering

variant of the well-studied Unsplittable Flow on a Path problem (UFP) [13, 6, 1, 18]. We denote this covering problem by *UFP-cover*. Its input consists of a path $G = (V, E)$ and a set of tasks T. Each task $i \in T$ is characterized by a start vertex $s_i \in V$, an end vertex $t_i \in V$, a size $p_i \in \mathbb{N}$ and a cost $w_i \in \mathbb{N}$. For each edge $e \in E$ we are given a demand u_e and we denote by $T_e \subseteq T$ the set of tasks i such that e lies on the path P_i between s_i and t_i. Our goal is to select a subset of the tasks $\bar{T} \subseteq T$ such that $p(\bar{T} \cap T_e) \geq u_e$ for each edge e where for any set of tasks $T' \subseteq T$ we define $p(T') := \sum_{i \in T'} p_i$ and $w(T') := \sum_{i \in T'} w_i$. Our objective is to minimize $w(\bar{T})$. When we reduce general caching to UFP-cover each time t_j of some request j is represented by an edge of G and there is a task i for each two consecutive requests of a page i' where intuitively selecting i represents loading i' again into the cache at the time of the second request and $p_i = p_{i'}$ and $w_i = w_{i'}$; see [6, 1] for details. Hence, if we restrict the sizes and costs in the considered instances of general caching then this restricts the sizes and costs of the resulting instance of UFP-cover in the same way. Additionally, UFP-cover is motivated by resource allocation settings where, e.g., the edge demands represent minimum requirements for energy, bandwidth, or workers and the tasks represent possibilities to satisfy a part of this demand during some given time interval at a certain cost.

General caching and UFP-cover are NP-hard which motivates studying approximation algorithms for them. The best known polynomial time approximation ratio for both problems is 4 [6] and there has been no improvement on this in almost 20 years. In this paper, we study the resource augmentation setting in which we are given a value $\delta > 0$ such that for each edge e we need to cover only a demand of $(1 - \delta)u_e$ while the compared optimum needs to cover u_e. No better algorithm is known for this setting.

1.1 Our Contribution

We first study the case of UFP-cover where $p_i = w_i$ for each task i for which the best known result is still the mentioned 4-approximation for the general case [6]. We present a polynomial time algorithm that computes a solution (that is feasible under resource augmentation) whose cost is at most the cost of the optimal solution (without resource augmentation). Hence, intuitively we solve the problem optimally under resource augmentation. We consider first the special case where the edge demands are in a constant range. We prove that there are solutions of cost at most OPT that are feasible under resource augmentation and which have the following structure: we can partition E into subpaths where on each subpath there are constantly many special edges such that each task of relatively small size (*small task*) in the solution uses one of these edges. This drastically simplifies the computations for the small tasks: we design an algorithm that guesses the partition of E into subpaths and then on each subpath it approximately guesses the small tasks crossing the constantly many special edges in OPT. After that, it additionally selects tasks with relatively large sizes (*large tasks*) via a dynamic program. For the case of arbitrary edge demands we consider for each edge e the amount by which the optimal solution covers e and partition the edges according to ranges of these values. For each range we show that there is a partition of E into subpaths with constantly many special edges for the small tasks, like in the case of a constant range of edge demands. Then, we design a dynamic program that intuitively patches the solutions for these ranges together.

▶ **Theorem 1.** *For any constant $\delta > 0$ there is a polynomial time algorithm for the case of UFP-cover where $w_i = p_i$ for each task i, that computes a solution with optimal cost that is feasible under $(1 - \delta)$-resource augmentation.*

Using our techniques, we derive an algorithm for the case of general caching where $p_i = w_i$ for each page $i \in \mathcal{P}$ which is known as the *bit-model* [20], meaning that the cost w_i of bringing a page i into the cache is proportional to its size p_i (which is a natural assumption). Our algorithm computes a solution that is feasible for a slightly increased cache of size $(1+\delta)M$ and whose cost is at most the cost of the optimal solution for a cache of size M. The notions of resource augmentation for UFP-cover and general caching are not equivalent w.r.t. the known reduction from general caching to UFP-cover, i.e., an algorithm for UFP-cover under resource augmentation does *not* imply an algorithm for general caching under resource augmentation. Therefore, we derive a reduction from general caching to UFP (instead of UFP-cover) in which we have the same input as for UFP-cover but we want to select a set of tasks $\bar{T}$ of *maximum* total weight such that on each edge e the tasks $\bar{T}$ do not exceed the capacity, i.e., $p(\bar{T} \cap T_e) \leq u_e$. We argue that if we increase the size of the cache in a general caching instance by a factor $1+\delta$ then in the reduced UFP instance the capacity of each edge increases by at least a factor $1+\delta$. We adapt our new techniques for UFP-cover above to UFP and obtain an algorithm for UFP that computes a solution of value OPT if we can increase the capacity of each edge by a factor $1+\delta$ and if $p_i = w_i$ for each task i. This yields an algorithm for general caching under resource augmentation for the case that $p_i = w_i$ for each page i, computing again a solution of cost at most OPT.

▶ **Theorem 2.** *For any constant $\delta > 0$ there are algorithms with polynomial running time for the cases of general caching and UFP where $w_i = p_i$ for each page/task i that compute solutions with optimal cost that are feasible under $(1+\delta)$-resource augmentation.*

Then we study the case of UFP-cover in which the cost w_i of each task i equals its "area", i.e., its size p_i multiplied by the length of its path P_i. We first prove that if the edge capacities are in a constant range then we can compute a $(1+\epsilon)$-approximation under resource augmentation by extending techniques from [17]. Then we turn this routine into a PTAS *without* resource augmentation for the same setting. To this end, we prove that there are $(1+\epsilon)$-approximate solutions in which for each edge e either all small input tasks using it are selected or e is covered to an extent of at least $(1+\delta^2)u_e$ which yields some slack. We intuitively guess the edges e of the former type, select all input tasks using them, and then apply our algorithm for resource augmentation on the remaining edges. With similar ideas, we construct an algorithm that computes a solution with optimal cost under resource augmentation for a constant range of edge capacities.

Then we present a polynomial time $(1+\epsilon)$-approximation under resource augmentation for arbitrary edge demands, under the same assumption on the task's costs. To construct this algorithm, we provide a reduction that essentially turns a polynomial time α-approximation for the special case of a *constant range* of edge capacities into a polynomial time $(1+\epsilon)\alpha$-approximation algorithm for *arbitrary* edge capacities under resource augmentation. We apply this reduction to the previous algorithm which yields a $(1+\epsilon)$-approximation under resource augmentation. The reduction works for arbitrary cost functions and in particular it might be useful for future work. To derive such a reduction, it might seem natural to split the overall problem into subproblems corresponding to the different ranges of the edge capacities. However, in UFP-cover there can be an edge e with very small demand which in the optimal solution is covered by tasks whose total size is very large. Hence, the demand of an edge might not give us a good estimate for how much the optimal solution covers it. Therefore, our reduction is guided by the (unknown) amount by which the optimal solution covers each edge, instead of the edge demands themselves. The resulting algorithm is a dynamic program which makes repeated calls to the given algorithm for a constant range of edge capacities and in which solutions of some DP-cells yield input tasks of other cells.

▶ **Theorem 3.** *Consider the case of UFP-cover where $w_i = |P_i| \cdot p_i$ for each task i. For any constants $\epsilon, \delta > 0$ there is a polynomial time algorithm that computes*

- *a $(1+\epsilon)$-approximate solution that is feasible under $(1-\delta)$-resource augmentation,*
- *a $(1+\epsilon)$-approximate solution without resource augmentation, if the edge capacities are in a constant range,*
- *a solution with cost at most OPT that is feasible under $(1-\delta)$-resource augmentation, if the edge capacities are in a constant range.*

Due to space constraints almost all proofs are deferred to the full version of the paper.

1.2 Other related work

UFP-cover is a generalization of the knapsack-cover problem. For the latter, an LP-formulation with a constant integrality gap is known [10] based on the knapsack-cover inequalities which are also used in other settings [12, 5, 11]. On the other hand, UFP-cover is a special case of the capacitated set cover problem, e.g., [11, 4], in which we are given a set of elements with demands and a family of sets where each set has a size and one seeks to select sets such that each element is covered by sets whose total size is at least the demand of the element.

For UFP-cover there is a QPTAS if the input data is quasi-polynomially bounded [19] and with the reduction in [6, 1] the same holds for general caching. A related problem is the general scheduling problem on one machine without release dates in which we are given a set of jobs where for each job we have to pay a cost that depends on its completion time. The best known approximation algorithm for this problem is a $(4+\epsilon)$-approximation [12] that generalizes the 4-approximation for UFP-cover in [6] and there is a QPTAS for quasi-polynomially bounded input data [2].

For UFP (packing) the best known polynomial time approximation ratio is $5/3+\epsilon$ [18] and there is a QPTAS [3, 7]. For the cases that the weight of each task is proportional to its size or to its "area" even PTASs are known [7, 17]. In this paper we extend the PTAS for the latter case to UFP-cover under resource augmentation for bounded edge demands. However, for the case where $p_i = w_i$ for each task i we need a completely different approach.

2 Task costs proportional to size

Given a constant $\delta > 0$, we present a polynomial time algorithm for UFP-cover for the case that $p_i = w_i$ for each task i. Our algorithm computes a solution that is feasible under $(1-\delta)$-resource augmentation whose cost is at most the cost of the optimal solution without resource augmentation.

By adding edges with demand 0 we can assume w.l.o.g. that the start and end vertices of the input tasks of any considered instance are pairwise distinct. First, we describe an algorithm for the special case that there is a value U such that $u_e \in [\delta U, U)$ for each edge e and later we extend this algorithm to the general case.

We start by showing that there is a well-structured solution whose cost is at most $w(OPT)$. Our algorithm will later compute a solution with this structure. We classify tasks into large and small tasks. A task i is *large* if $p_i \geq \delta^3 U$ and *small* otherwise. We denote by T_L and T_S the large and small input tasks, respectively. First, we establish some properties of the optimal solution OPT that in fact hold for arbitrary task costs (assuming that $u_e \in [\delta U, U)$ for each edge e).

▶ **Lemma 4.** *Let OPT be an optimal solution. For each edge e it holds that $p(OPT \cap T_S \cap T_e) \leq (2+2\delta^3)U$ and $|OPT \cap T_L \cap T_e| \leq O(1/\delta^3)$.*

We want to cut the given instance into simpler subinstances via a partition of E into subpaths $E = E_1 \dot\cup E_2 \dot\cup ... \dot\cup E_k$ such that intuitively we can compute an optimal solution for each subpath E_j separately and then output the union. For each subpath E_j we require that there are $9/\delta$ *special edges* $e_{j,1}, e_{j,2}, ..., e_{j,j'} \in E_j$ and that there is a set $T'_j \subseteq T_S$ such that each task in T'_j uses at least one of the edges $e_{j,1}, e_{j,2}, ..., e_{j,j'}$ and the tasks in T'_j, together with a global set of large tasks $T'_L \subseteq T_L$, form a feasible solution for E_j under resource augmentation. Then we define a solution T' to be the union of the sets T'_j together with T'_L. Note that a small task i might be contained in several sets T'_j and in this case we add it several times to T', i.e., we allow T' to be a multiset. Formally, we look for solutions T' that are *nice*. Figure 1 gives some intuition on how such a solution looks like.

▶ **Definition 5.** *A multiset T' is* nice *if there exists a partition of E into subpaths $E = E_1 \dot\cup E_2 \dot\cup ... \dot\cup E_k$ and partition of T' into sets $T' = T'_L \dot\cup T'_1 \dot\cup T'_2 \dot\cup ... \dot\cup T'_k$ such that*

- *$T'_L = T' \cap T_L$,*
- *for each j we have that $T'_j \subseteq T_S$ and T'_j contains each task at most once,*
- *for each subpath E_j there are at most most $9/\delta$ edges $e_{j,1}, e_{j,2}, ..., e_{j,j'} \in E_j$ such that each task $i \in T'_j$ uses at least one of them, and for each $e \in E_j$ we have that $p(T_e \cap (T'_j \cup T'_L)) \geq (1-\delta/2)u_e$.*

▶ **Lemma 6.** *There exists a nice multiset T' with a corresponding partition $T' = T'_L \dot\cup T'_1 \dot\cup T'_2 \dot\cup ... \dot\cup T'_k$ such that $w(T'_L) + \sum_{k'=1}^{k} w(T'_{k'}) \leq w(OPT)$.*

Proof sketch. We define $T'_L := OPT \cap T_L$. For any two vertices $u, v \in V$ denote by $P_{u,v}$ the path between u and v. We define the partition $E = E_1 \dot\cup E_2 \dot\cup ... \dot\cup E_k$ and the corresponding sets T'_j inductively. Suppose that we have already defined $k'-1$ paths $E_1 \dot\cup E_2 \dot\cup ... \dot\cup E_{k'-1}$. Let v_0 denote the rightmost vertex of $E_{k'-1}$ and for the case that $k'=1$ let v_0 be the leftmost vertex of V. We define $e_{k',1} = \{u_1, v_1\}$ to be the leftmost edge such that the total size of small tasks $T_S \cap OPT$ whose path is contained in P_{v_0,u_1} is at least $\delta^2 U/4$. The total size of those tasks is at most $\delta^2 U/3$ since $p_i \leq \delta^3 U$ for each small task i and the end vertices of the input tasks are pairwise distinct. Inductively, suppose that we have defined j' edges $e_{k',1}, e_{k',2}, ..., e_{k',j'}$ in this way and let $e_{k',j'} = \{u_{j'}, v_{j'}\}$. We define $e_{k',j'+1} = \{u_{j'+1}, v_{j'+1}\}$ to be the leftmost edge on the right of $e_{k',j'}$ such that the total size of small tasks $T_S \cap OPT$ whose path is contained in $P_{v_{j'},u_{j'+1}}$ is at least $\delta^2 U/4$ (and hence at most $\delta^2 U/3$). We stop after defining $e_{k',9/\delta^2} = \{u_{9/\delta^2}, v_{9/\delta^2}\}$ and define $E_{k'} := P_{v_0,v_{9/\delta^2}}$. We define $T'_{k'}$ to be all tasks in $T_S \cap OPT$ whose path contains one of the edges $e_{k',1}, e_{k',2}, ..., e_{k',9/\delta^2}$. Note that the total cost of tasks in $OPT \cap T_S \setminus T'_{k'}$ that use an edge of $E_{k'}$ (i.e., small tasks of OPT that we did not add to $T'_{k'}$) is at least $\frac{9}{\delta^2} \cdot \frac{\delta^2 U}{4} \geq (2+\delta)U$. This justifies that tasks in $OPT \cap T_S$ using the rightmost edge of $E_{k'}$ might be added to $T'_{k'}$ and to $T'_{k'+1}$ and hence we have to pay twice for them (their total size and hence their total cost is at most $(2+2\delta^3)U$ by Lemma 4). We stop if during some iteration k we cannot find a next edge $e_{k,j'+1} = \{u_{j'+1}, v_{j'+1}\}$ according to our definition. In this case we define E_k to be the path between v_0 and the rightmost vertex of V and stop the construction procedure. One can show that the set $T'_L \cup \bigcup_{k'} T'_{k'}$ is feasible under resource augmentation, i.e., that $p(T_e \cap (T'_L \cup T'_j)) \geq (1-\delta/2)u_e$ for each $e \in E_j$ for each j and that $w(T') = w(OPT \cap T_L) + \sum_{k'} w(T'_{k'}) \leq w(OPT)$. ◀

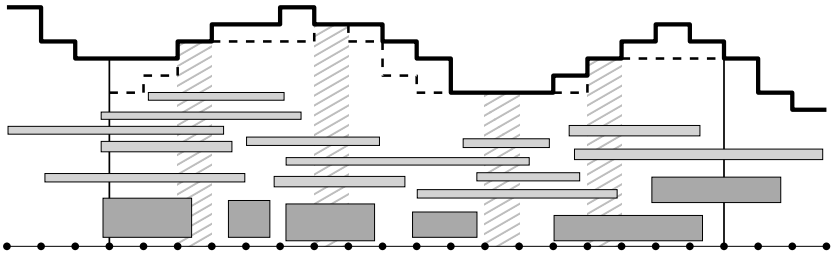


Figure 1 Some of the tasks of a nice solution covering a subpath E_j. The vertical lines are the boundaries of E_j and the shaded columns represent the few special edges $e_{j,1}, e_{j,2}, ..., e_{j,j'} \in E_j$. All small tasks (depicted in light gray) cross one of those edges. The demand covered by the solution (dashed curve) might be by a factor $(1 - \delta/2)$ smaller than the demand of the edges (thick curve). Note that the complete nice solution contains many more tasks covering E_j than the ones shown above.

The algorithm

We present now an algorithm that intuitively computes a nice solution $\bar{T}$ whose cost is at most $w(T')$. We first present such an algorithm for the case that for the partition $E = E_1 \dot\cup E_2 \dot\cup ... \dot\cup E_k$ of T' it holds that $k = 1$ and then extend it later to the case that $k > 1$.

Assume that $k = 1$. We guess the at most $9/\delta^2$ edges $e_{1,1}, e_{1,2}, ..., e_{1,j'}$ in time $n^{O(1/\delta^2)}$, i.e., we enumerate all possibilities. We guess an estimate for the capacity needed by the tasks in T'_1 on each edge. To this end, for each edge $e \in E$ let $f_1(e) := p\left(T'_1 \cap T_e \cap T_{e_{1,1}}\right)$ denote the total size of the tasks in $T'_1 \cap T_{e_{1,1}}$ that use e. Intuitively, we would like to guess the function f_1, however, there are too many possibilities for it. Therefore, instead we guess the estimate $\hat{f}_1(e) := \left\lfloor \frac{f_1(e)}{\delta^4 U/36} \right\rfloor \delta^4 U/36$. Using that f_1 is non-decreasing on the left of $e_{1,1}$ and non-increasing on the right of $e_{1,1}$, we will show that $\hat{f}_1(e)$ has only $O(1/\delta^4)$ many steps. We define inductively functions $\hat{f}_2, ..., \hat{f}_{j'}$ where each function $\hat{f}_{j''}$ is an estimate for the size of the tasks in T'_1 that use $e_{1,j''}$ but *not* $e_{1,j''-1}$. Formally, we define $f_{j''}(e) := p\left(\left(T'_1 \cap T_e \cap T_{e_{1,j''}} \cap T_S\right) \setminus T_{e_{1,j''-1}}\right)$ and $\hat{f}_{j''}(e) := \left\lfloor \frac{f_{j''}(e)}{\delta^4 U/36} \right\rfloor \delta^4 U/36$ for each $j'' = 2, ..., j'$.

▶ **Lemma 7.** *For each $j'' \in \{1, ..., j'\}$ the function $\hat{f}_{j''}$ is a step function with only $O(1/\delta^4)$ many steps whose values are all integral multiples of $\delta^4 U/36$ bounded by $(2 + 2\delta^3)U$. Also, for each edge $e \in E$ we have that $\sum_{j''} \hat{f}_{j''}(e) \geq \sum_{j''} f_{j''}(e) - \delta^2 U/4$.*

We guess each function $\hat{f}_{j''}$ with $j'' \in \{1, ..., j'\}$ in time $n^{O(1/\delta^4)}$ which gives $n^{O(1/\delta^6)}$ many guesses in total. For each function $\hat{f}_{j''}$ we invoke a polynomial time algorithm that computes a set of tasks $\bar{T}_{1,j''}$ that essentially covers $\hat{f}_{j''}$ and that is at most as costly as the tasks $T'_1 \cap T_{e_{1,j''} \cap T_S} \setminus T_{e_{1,j''-1}}$ (that define the profile $f_{j''}$).

▶ **Lemma 8.** *Let $j'' \in \{1, ..., j'\}$. There is an algorithm with a running time of $n^{O(1/\delta^4)}$ that computes a set of tasks $\bar{T}_{1,j''} \subseteq T_S \cap T_{e_{1,j''}} \setminus T_{e_{1,j''-1}}$ with $p(\bar{T}_{1,j''} \cap T_e) \geq \hat{f}_{j''}(e) - \delta^4 U/36$ for each edge e, and $w(\bar{T}_{1,j''}) \leq w(T'_1 \cap T_{e_{1,j''}} \setminus T_{e_{1,j''-1}})$.*

We define $\bar{T}_1 := \bigcup_{j''} \bar{T}_{1,j''}$ to be the small tasks that we select in order to cover E_1. It remains to select the large tasks. Due to Lemma 4 each edge is used by at most $O(1/\delta^3)$ tasks in T'_L. Therefore, we can use a dynamic program that computes the cheapest set of large tasks $\bar{T}_L$ that covers the demand that is not already covered by $\bar{T}_1$ (taking into account that we have resource augmentation). Intuitively, it sweeps the path from left to right and for each edge e it guesses the at most $O(1/\delta^3)$ many large tasks in $T'_L \cap T_e$. Finally, we output $\bar{T} := \bar{T}_1 \cup \bar{T}_L$.

▶ **Lemma 9.** *There is an algorithm with a running time of $n^{O(1/\delta^3)}$ that computes a set of tasks $\bar{T}_L \subseteq T_L$ such that $p((\bar{T}_L \cup \bar{T}_1) \cap T_e) \geq (1-\delta)u_e$ for each edge $e \in E$ and $w(\bar{T}_L) \leq w(T'_L)$.*

For the case that $k > 1$ we define a dynamic program that intuitively guesses the partition $E = E_1 \dot\cup E_2 \dot\cup ... \dot\cup E_k$ step by step and the large tasks at the boundaries of the subpaths. After guessing a subpath E_j and the large tasks using its boundary edges it invokes the algorithm for $k = 1$ as a subroutine on E_j and then continues with the guessing.

Formally, our DP has a cell (E'', T'') for each combination of a subpath E'' of E that contains the rightmost edge of E and a set of at most $O(1/\delta^3)$ large tasks T'' that use the leftmost edge of E'', denoted by e''_L. The reader may imagine that $E'' = E_j \cup E_{j+1} \cup ... \cup E_k$ for some j (where the subpaths E_j correspond to the nice solution T') and that T'' are the large tasks in T' that use the leftmost edge of E_j. The goal is to compute a set $\bar{T}''$ of tasks of low cost such that $\bar{T}'' \cup T''$ forms a feasible solution for E'' under resource augmentation, i.e., $p(T_e \cap (\bar{T}'' \cup T'')) \geq (1-\delta)u_e$. Given a cell (E'', T'') we intuitively guess $E_{j+1} \cup ... \cup E_k$, i.e., we try all subpaths $\bar{E}'' \subseteq E''$ that contain the rightmost edge of E and all sets $\bar{T}''$ of $O(1/\delta^3)$ large tasks that use the leftmost edge of $\bar{E}''$, denoted by $\bar{e}''_L$, such that T'' and $\bar{T}''$ are compatible, i.e., $T'' \cap T_{\bar{e}''_L} \subseteq \bar{T}''$ and $\bar{T}'' \cap T_{e''_L} \subseteq T''$. Let $\tilde{E}'' := E'' \setminus \bar{E}''$ (the reader may imagine that $\tilde{E}'' = E_j$). On $\tilde{E}''$ we apply the procedure above for the case of $k = 1$ and we slightly change the algoritm due to Lemma 9 such that we require that the large tasks $T'' \cup \bar{T}''$ are included in the computed set $\bar{T}_L$. Let $\hat{T}$ denote the resulting tasks. With the guess $(\bar{E}'', \bar{T}'')$ we associate the solution $\hat{T} \cup (\bar{T}'' \setminus T'') \cup DP(\bar{E}'', \bar{T}'')$ where $DP(\bar{E}'', \bar{T}'')$ denotes the solution stored in the cell $(\bar{E}'', \bar{T}'')$. We store in the cell (E'', T'') the solution of minimum cost among all guesses. Assume for convenience that we append an edge on the left of E with zero demand. We output the solution stored in the cell $(E, \emptyset)$.

Arbitrary demands

We generalize the above algorithm to the case of arbitrary edge demands. First, we change the definition of large and small tasks and in particular make it dependent on the (unknown) optimal solution. For each edge $e \in E$ we define $\hat{u}_e := p(OPT \cap T_e)$ and we define its level $\ell(e)$ to be the integer ℓ such that $\hat{u}_e \in [(1/\delta)^\ell, (1/\delta)^{\ell+1})$. For each task $i \in T$ we define its level $\ell(i)$ by $\ell(i) := \min_{e \in P_i} \ell(e)$. We say that a task $i \in T$ is *large* if $p_i \geq \delta^3 (1/\delta)^{\ell(i)+1}$ and *small* otherwise. For each level ℓ we define T^ℓ_S and T^ℓ_L to be the small and large tasks of level ℓ, respectively, and $T^\ell = T^\ell_S \cup T^\ell_L$. We extend the notion of a nice multiset T' to the case of arbitrary demands. Again, we have a partition of E into subpaths $E = E_1 \dot\cup E_2 \dot\cup ... \dot\cup E_k$ and a partition of T' into sets $T' = T'_1 \dot\cup T'_2 \dot\cup ... \dot\cup T'_k$ but now for each subpath E_j there can be up to $18/\delta^2$ special edges *for each level* ℓ. Similarly as before, for each edge $e \in E_j$ the small tasks in T'_j crossing one of these edges cover e together with the large tasks in T' (assuming that we have resource augmentation). Due to the resource augmentation we can even require that already the tasks in T' of levels $\ell(e) - 1$ and $\ell(e)$ are sufficient for covering e.

▶ **Definition 10.** *A multiset $T' \subseteq T$ is* nice-by-levels *if there exists a partition into subpaths $E = E_1 \dot\cup ... \dot\cup E_k$ and sets $T' = T'_1 \dot\cup ... \dot\cup T'_k$ with the property that each set T'_j contains each task i at most once and for each subpath E_j and each level ℓ there is a set of at most $18/\delta^2$ edges $e_{j,1,\ell}, e_{j,2,\ell}, ... \in E_j$ of level ℓ such that $T'_j \cap T^\ell_S \subseteq \bigcup_{j'} T_{e_{j,j',\ell}} \cap T^\ell_S$. Moreover, for each edge $e \in E_j$ of some level ℓ we have that $p(T_e \cap T'_j \cap (T^\ell \cup T^{\ell-1})) \geq (1-3\delta)\hat{u}_e \geq (1-3\delta)u_e$ and $|T' \cap T_e \cap T^\ell_L| \leq 1/\delta^3$.*

▶ **Lemma 11.** *There is a set T' that is nice-by-levels with $\sum_{k'} w(T'_{k'}) \leq w(OPT)$.*

Proof sketch. Assume that we already defined a set of vertices V' and a set of special vertices for each level $\ell' \in \{0, ..., \ell - 1\}$ such that the vertices V' separate E into subpaths E_j such that for each level $\ell' \in \{0, ..., \ell - 1\}$ each subpath E_j contains at most $18/\delta^2$ special edges of level ℓ'. For each subpath E_j we consider its edges of level ℓ and apply a slight adaptation of the procedure from the proof of Lemma 6. Let v_0 be the leftmost vertex of E_j. We define $e_{j,1,\ell} = \{u_1, v_1\}$ to be the leftmost edge such that the total size of the small tasks of level ℓ ending in P_{v_0,u_1} is at least $\delta^2(1/\delta)^{\ell+1}/4$ and hence at most $\delta^2(1/\delta)^{\ell+1}/3$. Inductively, we define special edges $\{e_{j,j',\ell}\}_{j\in[k],j'\in\mathbb{N}}$ in this way. If an edge e of level $\ell' > \ell$ is selected as a special edge we replace it by the two closest edges of level ℓ. We do this operation for each level ℓ. If we defined more than $18/\delta^2$ special edges of level ℓ, then we add to V' the right vertex of one in every $18/\delta^2$ special edges, so we partition E_j into subpaths such that each subpath contains at most $18/\delta^2$ special edges of level ℓ.

Let E_j be a subpath of the final partition, let $\{e_{j,j',\ell}\}_{j',\ell\in\mathbb{N}}$ be the special edges contained in E_j, and let $T'_j := OPT \cap \left(\{i \in T_L | P_i \cap E_j \neq \emptyset\} \cup \bigcup_\ell \bigcup_{j'} T_{e_{j,j',\ell}}\right)$ be the tasks corresponding to E_j. With a similar reasoning as in Lemma 6 before we argue that for each edge $e \in E_j$ the tasks in T'_j that cross e and additionally at least one of the edges $\{e_{j,j',\ell}\}_{j',\ell\in\mathbb{N}}$ have a total size of at least $(1-\delta)\hat{u}(e)$, i.e., essentially cover e. One can show that the tasks of level $\ell(e) - 2$ or smaller covering e have a total size of at most $2\delta\hat{u}_e$, using that each of them must use the closest edge on the left or on the right of e that is of level $\ell(e) - 2$ and each of them is used by tasks in OPT of total size at most $(1/\delta)^{\ell-1} \leq \delta\hat{u}_e$. We define $T' = \bigcup_{j=1}^{k} T'_j$ to be the constructed multiset. With a similar argumentation as in Lemma 6 one can show that $\sum_{k'} w(T'_{k'}) \leq w(OPT)$, arguing that some tasks in $OPT \cap T_S$ are not included in T' and that they compensate for the additional cost of tasks in OPT that are included several times in T'. ◀

We describe now a dynamic program that intuitively computes a nice-by-levels solution $\bar{T}$ with $w(\bar{T}) \leq w(T')$. Consider first the case that $k = 1$ and let $j = 1$. Unlike the case of a constant range of edge capacities, we cannot guess all edges $\{e_{j,j',\ell}\}_{j',\ell\in\mathbb{N}}$ in one step since they can be more than constantly many. Instead, we start with $\ell := 0$ we first guess all $O(1/\delta^2)$ special edges $\{e_{j,j',\ell}\}_{j'\in\mathbb{N}}$ of level ℓ and for each of these edges $e_{j,j',\ell}$ we guess the large tasks in T' using $e_{j,j',\ell}$, i.e., $T' \cap T_L^\ell \cap T_{e_{j,j',\ell}}$ and an approximation of the profiles of the small tasks that use $e_{j,j',\ell}$ and no special edge $e_{j,\hat{j}',\hat{\ell}}$ with $\hat{\ell} < \ell$ or with $\hat{\ell} = \ell$ and $\hat{j}' < j'$ (like in Section 2).

Formally, we define $f_1(e) := p\left(T'_j \cap T_e \cap T_{e_{j,1,\ell}} \setminus \bigcup_{\ell'<\ell} \bigcup_{\hat{j}} T_{e_{j,\hat{j},\ell'}}\right)$ and for each $j' > 1$ we define $f_{j'}(e) := p\left(T'_1 \cap T_e \cap T_{e_{j,j',\ell}} \setminus \left(T_{e_{j,j'-1,\ell}} \cup \bigcup_{\ell'<\ell} \bigcup_{\hat{j}} T_{e_{j,\hat{j},\ell'}}\right)\right)$. Then we define the approximative profiles by $\hat{f}_{j'}(e) := \left\lfloor \frac{f_{j'}(e)}{\delta^4(1/\delta)^\ell/36} \right\rfloor (1/\delta)^\ell\delta^4/36$ for each j'. Note that each function $\hat{f}_{j'}(e)$ has only $O(1/\delta^5)$ many steps. Since there are at most $18/\delta^2$ special edges of each level, we can guess all functions $\hat{f}_{j'}(e)$ in time $n^{O(1/\delta^7)}$. For each guessed profile we compute small tasks that essentially cover it, like in Lemma 8.

▶ **Lemma 12.** *For each j' there is an algorithm with a running time of $n^{O(1/\delta^5)}$ that computes a set of tasks $\bar{T}_{j,j',\ell} \subseteq T_{e_{j,j',\ell}} \setminus \left(T_{e_{j,\ell'-1,\ell}} \cup \bigcup_{\ell'<\ell} \bigcup_{\hat{j}} T_{e_{j,\hat{j},\ell'}} \cup \left(T' \cap T_L^\ell \cap T_{e_{j,j',\ell}}\right)\right)$ with $p(\bar{T}_{j,j',\ell} \cap T_e) \geq \hat{f}_{j'}(e) - \delta^4(1/\delta)^\ell/36$ for each edge e, and $w(\bar{T}_{j,j',\ell}) \leq f_{j'}(e_{j,j',\ell})$.*

Let E' be a subpath between two consecutive special edges in $\{e_{j,j',\ell}\}_{j'\in\mathbb{N}}$. It might be that E' contains edges of level ℓ. Denote these edges by E'_ℓ. We want to guess the large tasks of level ℓ that use an edge in E'_ℓ. In order to do this we invoke a dynamic program that intuitively sweeps in E' from left to right and in each step guesses an edge $e \in E'_\ell$ together

with the tasks $T' \cap T_e \cap T_L^\ell$. We recurse in each subpath E'' between two consecutive edges in E'_ℓ, between the leftmost edge of E' and the leftmost edge in E'_ℓ, and between the rightmost edge of E'_ℓ and the rightmost edge of E'. In each recursive call, the arguments consists of the subpath E'', the next level $\ell+1$, and the guessed $O(1/\delta^2)$ special edges of level ℓ and their profiles, and the large tasks of level ℓ using the leftmost edge of E'' or the rightmost edge of E''. In principle, by doing this we forget (and thus lose) the amount that tasks from levels $\ell-1$ and below cover on edges in E''. However, T' is nice-by-levels and thus we have that on each edge e of some level ℓ the tasks in $T' \cap (T^\ell \cup T^{\ell-1})$ are sufficient to cover the demand u_e (under resource augmentation). For the arguments in our recursive call there is only a polynomial number of options. Therefore, we can embed this recursion into a polynomial time dynamic program. The base case is given when the path E'' is empty or if the level ℓ is so large that $T_S^\ell = T_L^\ell = \emptyset$, i.e., if $\max_{e \in E} \hat{u}_e \leq \sum_{i \in T} p_i < (1/\delta)^\ell$.

Finally, if $k > 1$ then we use the same dynamic program as before in order to guess the partition $E = E_1 \dot\cup ... \dot\cup E_k$. Thus we have the following theorem. Its proof and the complete description of the algorithm is deferred to the full version of the paper.

▶ **Theorem 13.** *For any $\delta > 0$ there is an algorithm with a running time of $n^{(1/\delta)^{O(1)}}$ for the case of UFP-cover where $w_i = p_i$ for each task i that computes a solution T' with $w(T') \leq w(OPT)$ and that satisfies that $p(T_e \cap T') \geq (1-\delta)u_e$ for each edge e.*

2.1 General caching and UFP

We use the techniques from the previous algorithm to obtain algorithms for UFP and general caching under resource augmentation. First, we show how to reduce general caching to UFP.

▶ **Lemma 14.** *Given an instance $(\mathcal{P}, \mathcal{R}, M)$ of general caching such that $p_i = w_i$ for each page $q_i \in \mathcal{P}$. In polynomial time we can compute an instance (V, E, T, u) of UFP with $u_e \leq M$ for each edge $e \in E$ and $p_i = w_i$ for each task $i \in T$ such that*

1. *for any solution to $(\mathcal{P}, \mathcal{R}, M)$ with cost C there is a solution $T' \subseteq T$ to (V, E, T, u) with $w(T') = w(T) - C$ and vice versa,*
2. *for any solution to $(\mathcal{P}, \mathcal{R}, M(1+\delta))$ with cost C there is a solution T' to $(V, E, T, u + \mathbb{1}\delta M)$ with $w(T') = w(T) - C$ and vice versa.*

Lemma 14 implies that in order to obtain an algorithm for general caching under resource augmentation it sufficies to provide an algorithm for UFP under $(1+\delta)$-resource augmentation, i.e., where the capacity of each edge e is increased to $(1+\delta)u_e$. We construct an algorithm for UFP of the latter type. We begin with the case of a constant range of edge capacities, i.e., $u_e \in [\delta U, U)$. We define a task $i \in T$ to be *large* if $p_i \geq \delta^3 U$ and *small* otherwise. Denote by T_L and T_S the set of large and small input tasks, respectively. Like in the case of UFP-cover we are looking for solutions that are *nice* which in this case means that there is a partition of E into subpaths $E = E_1 \dot\cup E_2 \dot\cup ... \dot\cup E_k$ such that we restrict ourselves to tasks i such that $P_i \subseteq E_j$ for some $j \in \{1, ..., k\}$ and additionally for each set E_j there are some special edges $e_{j,1}, e_{j,2}, ..., e_{j,\ell}$ such that we select *each* task $i \in T_S$ with $P_i \subseteq E_j$ that does *not* use any of these special edges.

▶ **Definition 15.** *A set $T' \subseteq T$ is* nice *if there exists a partition of E into subpaths $E = E_1 \dot\cup E_2 \dot\cup ... \dot\cup E_k$ such that*

- *for each task $i \in T'$ we have that $P_i \subseteq E_j$ for some $j \in \{1, ..., k\}$,*
- *for each E_j there are at most $4/\delta^2$ edges $e_{j,1}, e_{j,2}, ..., e_{j,\ell} \in E_j$ such that $\{i \in T_S : P_i \subseteq E_j\} \setminus \bigcup_{\ell'=1}^{\ell} T_{e_{j,\ell'}} \subseteq T'$ and for each edge $e \in E_j$ we have that $p(T' \cap T_e) \leq u_e + \delta^2 U/2$.*

▶ **Lemma 16.** *There is a nice set T' that satisfies $w(T') \geq w(OPT)$.*

Our algorithm is now similar as the algorithm for UFP-cover above. If $k > 1$ then we first invoke a dynamic program that guesses the partition $E = E_1 \dot\cup E_2 \dot\cup ... \dot\cup E_k$. Let T'_j be the tasks $i \in T'$ with $P_i \subseteq E_j$. For each subpath E_j we guess the special edges $e_{j,1}, e_{j,2}, ...$ For each j' we define the profile of the small tasks using $e_{j,j'}$ and none of the edges $e_{j,1}, ..., e_{j,j'-1}$ by $f_{j'}(e) := p\left(T'_j \cap T_S \cap T_e \cap T_{e_{j,j'}} \setminus T_{e_{j,j'-1}}\right)$ and a corresponding *overestimating* profile $\hat f_{j'}(e) := \left\lceil \frac{f_{j'}(e)}{\delta^4 U/16} \right\rceil \delta^4 U/16$. The latter has $O(1/\delta^4)$ many steps for each j' so we guess all profiles $\hat f_{j'}(e)$ in time $n^{O(1/\delta^6)}$. We compute tasks for the profiles similarly as in Lemma 8.

▶ **Lemma 17.** *Let $j' \in \mathbb{N}$. There is an algorithm with a running time of $n^{O(1/\delta^4)}$ that computes a set of tasks $\bar T_{j,j'} \subseteq T_S \cap T_{e_{j,j'}} \setminus T_{e_{j,j'-1}}$ with $p(\bar T_{j,j'} \cap T_e) \leq \hat f_{j'}(e) + \delta^4 U/16$ for each edge e, and $w(\bar T_{j,j'}) \geq w(T'_j \cap T_S \cap T_{e_{j,j'}} \setminus T_{e_{j,j'-1}})$.*

Then we add all small tasks whose path is contained in E_j and that do not use any of the special edges $e_{j,1}, e_{j,2}, ...$. Let $\bar T_j$ denote the selected small tasks. If all our guesses were correct then one can show that on each edge e we have that $p(\bar T_j \cap T_e) \leq (1+\delta)u_e$. Finally, we invoke a dynamic program that selects a maximum weight set of large tasks that fits in the remaining edge capacities, similarly as in Lemma 9. We define $T'_L = T' \cap T_L$.

▶ **Lemma 18.** *There is an algorithm with a running time of $n^{O(1/\delta^3)}$ that computes a set of tasks $\bar T_L \subseteq T_L$ such that $p((\bar T_L \cup \bar T_j) \cap T_e) \leq (1+\delta)u_e$ for each edge $e \in E$ and $w(\bar T_L) \geq w(T'_L)$.*

In fact, using Lemma 14 one can show that our algorithm for UFP for a constant range of edge capacities is already sufficient for general caching under resource augmentation.

▶ **Theorem 19.** *For any $\delta > 0$ there is an algorithm with a running time of $n^{(1/\delta)^{O(1)}}$ for general caching instances $(\mathcal{P}, \mathcal{R}, M)$ where $w_i = p_i$ for each page $i \in \mathcal{P}$ that computes a solution whose cost is at most the cost of the optimal solution and that is feasible if the cache has size $(1+\delta)M$.*

For the case of arbitrary edge capacities in UFP we can use a similar generalization as for UFP-cover to obtain the following theorem.

▶ **Theorem 20.** *For any $\delta > 0$ there is an algorithm with a running time of $n^{(1/\delta)^{O(1)}}$ for the case of UFP where $w_i = p_i$ for each task i, that computes a solution T' with $w(T') \geq w(OPT)$ and that satisfies that $p(T_e \cap T') \leq (1+\delta)u_e$ for each edge e.*

3 Task costs proportional to area

In this section we study the case of UFP-cover in which $w_i = |P_i| \cdot p_i$ for each task $i \in T$. For the respective case of UFP a PTAS is known [17] that uses a sparsification step. For UFP-cover we cannot use this technique, however, in the next theorem we extend the methodology in [17] to a $(1+\epsilon)$-approximation for UFP-cover under resource augmentation if the edge capacities are in a constant range (its proof is deferred to the full version of the paper). Suppose that $u_e \in [\delta U, U)$ for some global value U. We define a task i to be *small* if $p_i \leq \delta^3 \cdot U$ and *large* otherwise, denote by T_S and T_L the respective sets of tasks.

▶ **Theorem 21.** *Let $\epsilon, \delta > 0$ and $U > 0$. Given an instance of UFP-cover for the case that $w_i = |P_i| \cdot p_i$ for each task $i \in T_S$ and $u_e \in \{0\} \cup [\delta U, U)$ for each edge e. There is an algorithm with a running time of $n^{O_{\epsilon,\delta}(1)}$ that computes a solution T' with $p(T' \cap T_e) \geq (1-\delta)u_e$ for each edge e and such that $w(T') \leq w(T_L \cap OPT) + (1+\epsilon)w(T_S \cap OPT) \leq (1+\epsilon)OPT$.*

Next, we construct a PTAS *without* resource augmentation for a constant range of edge capacities. The PTAS will use the algorithm due to Theorem 21 as a black-box. Recall that in OPT each edge is used by at most $O(1/\delta^3)$ large tasks (see Lemma 4). We prove that there is a solution OPT' with $w(OPT') \leq (1+O(\delta))w(OPT)$ such that for every edge $e \in E$ it holds that either OPT' contains all small tasks using e or OPT' covers e to an extent of at least $(1+\delta^2)u_e$. Intuitively, we construct OPT' by taking OPT and adding tasks from T_S greedily until the property is satisfied. We show that in this way each edge is used by additional tasks of size at most $O(\delta^2)$. Hence, the total cost of these additional tasks is at most $O(\delta^2 U)|E|$ while $w(OPT) \geq \delta U|E|$.

▶ **Lemma 22.** *There is a set OPT' such that $w(OPT') \leq (1+O(\delta))w(OPT)$ and on each edge $e \in E$ it holds that $p(OPT' \cap T_e) \geq (1+\delta^2)u_e$ or $T_e \cap T_S \subseteq OPT'$.*

Let E' denote the set of edges e such that OPT' contains all small tasks using e. Using a standard dynamic program we guess the edges E' step by step (see the full version of the paper for details). Then, for each edge $e \in E'$ we guess the $O(1/\delta^3)$ large tasks in $OPT' \cap T_e$ and we selects all small tasks in $T_e \cap T_S$. Observe that in this way we select all tasks in OPT' that use some edge in E'. For each subpath E'' between two consecutive edges $e_1, e_2 \in E'$ we compute a set of tasks that cover the remaining demand on E'' (i.e., the demand not covered by the tasks in $(T_{e_1} \cup T_{e_2}) \cap OPT'$). To this end, we invoke the algorithm due to Theorem 21 on an auxiliary instance defined as follows. We start with E and contract all edges that are not in E''. If for some edge $e \in E''$ the tasks in $(T_{e_1} \cup T_{e_2}) \cap OPT'$ already cover e, i.e., if $p(T_e \cap (T_{e_1} \cup T_{e_2}) \cap OPT') \geq u_e$, then we contract e as well. Otherwise, note that OPT' covers e to an extent of at least $(1+\delta^2)u_e$ and therefore the tasks in $OPT' \setminus (T_{e_1} \cup T_{e_2})$ cover e to an extent of at least $\bar{u}_e := (1+\delta^2)u_e - p\left(T_e \cap (T_{e_1} \cup T_{e_2}) \cap OPT'\right)$. Therefore, we define the demand of each edge e to be $\bar{u}_e$. One can show that $\bar{u}_e \in [\delta^2 U, U)$ (since we did not contract e). Let T' denote the solution that we obtain if we apply the algorithm due to Theorem 21 on this instance. One can show that then $T' \cup ((T_{e_1} \cup T_{e_2}) \cap OPT')$ covers the original demand u_e on each edge $e \in E''$, i.e., $p(T_e \cap T' \cup ((T_{e_1} \cup T_{e_2}) \cap OPT')) \geq u_e$ for each edge $e \in E''$. We apply this routine on each subpath E'' between two consecutive edges in E'.

Above, we constructed OPT' by adding tasks to OPT in order to create some slack. If instead we remove tasks from OPT we can construct a solution OPT'' which is cheaper than OPT, feasible under resource augmentation, and in which on each edge e we removed small tasks of total size $\delta^2 u_e$ or we removed all small tasks using e. Being guided by OPT'' instead of OPT' we can construct an algorithm that computes a solution of cost at most OPT that is feasible under resource augmentation.

▶ **Theorem 23.** *Consider the case of UFP-cover where $w_i = |P_i| \cdot p_i$ for each task i and in which the edge capacities are in a constant range. This case admits a PTAS and for any $\delta > 0$ there is an algorithm with a running time of $n^{(1/\delta)^{O(1)}}$ that computes a solution T' with $w(T') \leq w(OPT)$ and that satisfies that $p(T_e \cap T') \geq (1-\delta)u_e$ for each edge e.*

4 Reducing to a constant range of edge demands

We describe a black-box procedure that intuitively turns an α-approximation algorithm for the case of UFP-cover of a bounded range of edge demands (independently of the costs of the tasks) into an $\alpha(1+\epsilon)$-approximation algorithm under $(1-\delta)$-resource augmentation for arbitrary edge demands. As a technicality, we need that the former algorithm works on instances with some normal tasks and some artificial huge tasks T_H where each task $i \in T_H$

has the property that it alone covers the complete demand on each edge of its path P_i and we require that the algorithm loses the factor of α only on the cost of the normal tasks. Note that any restriction on the set of normal tasks, like cost proportional to the area or to the size, is preserved in the reduction.

▶ **Lemma 24.** *Let $\epsilon, \delta > 0$ and given an instance (T, E, u) of UFP-cover. Suppose there is a polynomial time approximation algorithm for instances of the form $(T \cup T_H, E', \bar{u})$ where $E' \subseteq E$ and there is a value U such that for each edge $e \in E'$ it holds that $\delta^2 U \leq \bar{u}_e \leq U$ or that $\bar{u}_e = M := 1 + \sum_{i \in T} p_i$ and that $p_{i'} = M$ for each $i' \in T_H$ and assume that this algorithm computes solutions of cost at most $w(OPT \cap T_H) + \alpha w(OPT \cap T)$. Then there is a polynomial time algorithm that computes a solution $T' \subseteq T$ for (T, E, u) with $w(T') \leq \alpha(1+\epsilon)OPT$ and $p(T_e \cap T') \geq (1-\delta)u_e$ for each edge e.*

Proof sketch. Similarly as in the algorithm for UFP-cover in Section 2 we group the edges into levels according to the extent by which they are covered in OPT. For each edge $e \in E$ we define $\hat{u}_e := p(OPT \cap T_e)$ and we define its level $\ell(e)$ to be the integer ℓ such that $\hat{u}_e \in [(1/\delta)^\ell, (1/\delta)^{\ell+1})$. For each task $i \in T$ we define its level $\ell(i) := \min_{e \in P_i} \ell(e)$. Let $T^{(\ell)}$ denote the tasks of level ℓ and let $E^{(\ell)}$ denote the edges of level ℓ. We assign the tasks into groups $\mathcal{T}^{(s)}$ such that each group contains the tasks from $1/\epsilon + 1$ consecutive levels, intuitively most tasks are contained in exactly one group and some few tasks are contained in two groups. For an offset $\beta \in \{0, ..., 1/\epsilon - 1\}$ to be defined later we define $\mathcal{T}^{(s)} := \bigcup_{\ell=\beta+s/\epsilon}^{\beta+(s+1)/\epsilon} T^{(\ell)}$ for each $s \in \mathbb{Z}$. In particular, for each $s \in \mathbb{Z}$ the tasks in $T^{(\beta+(s+1)/\epsilon)}$ are contained in two groups, $\mathcal{T}^{(s)}$ and $\mathcal{T}^{(s+1)}$. By a shifting argument there is a choice for β such that $\sum_s w(\mathcal{T}^{(s)} \cap OPT) \leq (1+\epsilon)OPT$. Similarly, we group the edges into groups. However, now each edge will be contained in only one group. For each $s \in \mathbb{Z}$ we define $\mathcal{E}^{(s)} := \bigcup_{\ell=\beta+s/\epsilon+1}^{\beta+(s+1)/\epsilon} E^{(\ell)}$. One key observation is that due to the resource augmentation the tasks in $\mathcal{T}^{(s)} \cap OPT$ are sufficient to cover the demand of all edges in $\mathcal{E}^{(s)}$.

▷ Claim 25. Let $s \in \mathbb{N}$. For each edge $e \in \mathcal{E}^{(s)}$ it holds that $T_e \cap \mathcal{T}^{(s)} \cap OPT \geq \hat{u}_e(1 - O(\delta))$.

Hence, if we knew the level of each edge then we could generate one subinstance for each group s whose input contains only the edges $\mathcal{E}^{(s)}$ and the tasks in $\mathcal{T}^{(s)}$ and then take the union of these solutions. Unfortunately, we do not know the levels of the edges. Instead, we define a dynamic program. Our DP has a cell (E', s) for each $E' \subseteq E$ and each level $s \in \mathbb{Z}$. For each $s \in \mathbb{Z}$ let $\mathcal{P}(s)$ denote (unknown) the maximal subpaths consisting of edges in $\bigcup_{s':s' \geq s} \mathcal{E}^{(s')}$.

Consider a cell (E', s), where the reader may imagine that $E' \in \mathcal{P}(s)$. The goal is to compute a set of tasks that cover E' where we restrict ourselves to solutions such that each edge $e \in E'$ is covered to an extent of at least $(1-\delta)(1/\delta)^{\beta+s/\epsilon+1}$ (even though the real demand u_e of e might be smaller). Note that if $E' \in \mathcal{P}(s)$ then $p(T_e \cap \{i \in OPT \cap \bigcup_{s'=s}^{\infty} \mathcal{T}^{(s')} | P_i \cap E' \neq \emptyset\}) \geq (1-\delta)(1/\delta)^{\beta+s/\epsilon+1}$, for each $e \in E'$. Inductively, assume that there is a value s such that for each $E' \in \mathcal{P}(s')$ with $s' > s$ the DP-cell (E', s') stores a solution of cost at most $\alpha(1+\epsilon)w\left(\{i \in OPT \cap \bigcup_{s'=s+1}^{\infty} \mathcal{T}^{(s')} | P_i \cap E' \neq \emptyset\}\right)$ that covers each edge $e \in E'$ to an extent of at least $u_e(1-\delta)$. Ideally, we would like to guess the paths $E'' \subseteq E'$ such that $E'' \in \mathcal{P}(s+1)$, take the solutions stored in the respective cells $(E'', s+1)$, and then call the α-approximation on the remaining edges on E''. However, the number of such paths E'' can be too large. Instead, for each solution in a cell $(E'', s+1)$ with $E'' \subseteq E'$ we introduce an artificial input task $i(E'', s+1)$ with $p_{i(E'',s+1)} = M$ and the weight $w_{i(E'',s+1)}$ of $i(E'', s+1)$ is defined as the cost of the solution stored in the cell $(E'', s+1)$. Let T_H denote the set of all these (polynomially many) artifical tasks. We

define demand of the edges such that for each edge $e \in E'$ we define its demand $\bar{u}_e$ to be $\bar{u}_e := (1-\delta)(1/\delta)^{\beta+s/\epsilon+1}$ if $u_e < (1/\delta)^{\beta+s/\epsilon+1}$ (recall that we restrict ourselves to solutions that cover at least this amount), $\bar{u}_e := (1-\delta)u_e$ if $(1/\delta)^{\beta+s/\epsilon+1} \leq u_e < (1/\delta)^{\beta+(s+1)/\epsilon+1}$, and $\bar{u}_e := M$ if $u_e \geq (1/\delta)^{\beta+(s+1)/\epsilon+1}$. Observe that if $u_e \geq (1/\delta)^{\beta+(s+1)/\epsilon+1}$ then $\ell(e) > s$ and hence the tasks in $OPT \cap \mathcal{T}^{(s)}$ are not needed for covering e (due to the resource augmentation). Note that one solution to this subproblem is to select the task $i(E'', s+1)$ for each $E'' \in \mathcal{P}(s+1)$ and the tasks in $OPT \cap \mathcal{T}^{(s)}$ that use E'. From Claim 25 we know that the tasks in sets $\mathcal{T}^{(s')}$ with $s' < s$ are not needed to cover the demand in E'. We invoke the α-approximation algorithm and store the solution in the cell (E', s'). One can show that the cell $(E, -1)$ then stores a $\alpha(1+\epsilon)$-approximative solution T' that satisfies $p(T_e \cap T') \geq (1-\delta)u_e$ for each $e \in E$. ◀

▶ **Corollary 26.** *Consider the case of UFP-cover where $w_i = |P_i| \cdot p_i$ for each task i. For any $\epsilon, \delta > 0$ there is a polynomial time algorithm that computes a solution T' with $w(T') \leq (1+\epsilon)w(OPT)$ and that satisfies that $p(T_e \cap T') \geq (1-\delta)u_e$ for each edge e.*

5 Conclusion and open problems

In this paper we studied approximation algorithms for UFP-cover under resource augmentation. We gave algorithms for the cases where the task costs are proportional to their "areas" or sizes, computing solutions whose costs are $(1+\epsilon)$-approximate or even optimal, respectively. It is an open question whether one can obtain such algorithms also for the general case under resource augmentation. An interesting first step would be to get an algorithm for this setting with a better approximation ratio than 4.

Our results imply that under resource augmentation UFP-cover is no longer NP-hard if the cost of each task equals its size (unless P = NP), and the same holds for the respective settings of UFP (packing) and general caching. This raises the question whether the general case of these problems is still NP-hard in the resource augmentation setting or whether one can compute a solution with optimal cost in polynomial time. Another natural open question is whether one can obtain results like ours also *without* resource augmentation. We hope that our new techniques help constructing such algorithms.

References

1. Susanne Albers, Sanjeev Arora, and Sanjeev Khanna. Page replacement for general caching problems. In *SODA*, volume 99, pages 31–40. Citeseer, 1999.
2. Antonios Antoniadis, Ruben Hoeksma, Julie Meißner, José Verschae, and Andreas Wiese. A QPTAS for the General Scheduling Problem with Identical Release Dates. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2017.31.
3. Nikhil Bansal, Amit Chakrabarti, Amir Epstein, and Baruch Schieber. A quasi-PTAS for unsplittable flow on line graphs. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC 2006)*, pages 721–729. ACM, 2006.
4. Nikhil Bansal, Ravishankar Krishnaswamy, and Barna Saha. On capacitated set cover problems. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 38–49. Springer, 2011.
5. Nikhil Bansal and Kirk Pruhs. The geometry of scheduling. *SIAM Journal on Computing*, 43(5):1684–1698, 2014.

6 Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph (Seffi) Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, September 2001. doi:10.1145/502102.502107.

7 Jatin Batra, Naveen Garg, Amit Kumar, Tobias Mömke, and Andreas Wiese. New approximation schemes for unsplittable flow on a path. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*, pages 47–58, 2015. doi:10.1137/1.9781611973730.5.

8 Laszlo A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal*, 5(2):78–101, 1966.

9 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 2005.

10 Robert D Carr, Lisa K Fleischer, Vitus J Leung, and Cynthia A Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *Proceedings of the 11th annual ACM-SIAM symposium on Discrete algorithms (SODA 2000)*, pages 106–115. Society for Industrial and Applied Mathematics, 2000.

11 Deeparnab Chakrabarty, Elyot Grant, and Jochen Könemann. On column-restricted and priority covering integer programs. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 355–368. Springer, 2010.

12 Maurice Cheung, Julián Mestre, David B Shmoys, and José Verschae. A primal-dual approximation algorithm for min-sum single-machine scheduling problems. *SIAM Journal on Discrete Mathematics*, 31(2):825–838, 2017.

13 M. Chrobak, G. Woeginger, K. Makino, and H. Xu. Caching is hard, even in the fault model. In *ESA*, pages 195–206, 2010.

14 Marek Chrobak, H Karloof, Tom Payne, and S Vishwnathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991.

15 Amos Fiat, Richard M Karp, Michael Luby, Lyle A McGeoch, Daniel D Sleator, and Neal E Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.

16 P. A. Franaszek and T. J. Wagner. Some distribution-free aspects of paging algorithm performance. *J. ACM*, 21(1):31–39, January 1974. doi:10.1145/321796.321800.

17 Fabrizio Grandoni, Tobias Mömke, Andreas Wiese, and Hang Zhou. To augment or not to augment: Solving unsplittable flow on a path by creating slack. In *Proc. of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, 2017. To appear.

18 Fabrizio Grandoni, Tobias Mömke, Andreas Wiese, and Hang Zhou. A $(5/3+\varepsilon)$-approximation for unsplittable flow on a path: placing small tasks into boxes. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC 2018)*, pages 607–619. ACM, 2018.

19 Wiebke Höhn, Julián Mestre, and Andreas Wiese. How unsplittable-flow-covering helps scheduling with job-dependent cost functions. *Algorithmica*, 80(4):1191–1213, 2018.

20 Sandy Irani. Page replacement with multi-size pages and applications to web caching. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 701–710. ACM, 1997.

21 Daniel D Sleator and Robert E Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

Improved Bounds on Fourier Entropy and Min-Entropy

Srinivasan Arunachalam
MIT, Cambridge, MA, USA
arunacha@mit.edu

Sourav Chakraborty
Indian Statistical Institute, Kolkata, India
sourav@isical.ac.in

Michal Koucký
Computer Science Institute of Charles University, Prague, Czech Republic
koucky@iuuk.mff.cuni.cz

Nitin Saurabh
Max Planck Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany
nsaurabh@mpi-inf.mpg.de

Ronald de Wolf
QuSoft, CWI and University of Amsterdam, the Netherlands
rdewolf@cwi.nl

Abstract

Given a Boolean function $f : \{-1,1\}^n \to \{-1,1\}$, define the *Fourier distribution* to be the distribution on subsets of $[n]$, where each $S \subseteq [n]$ is sampled with probability $\widehat{f}(S)^2$. The Fourier Entropy-Influence (FEI) conjecture of Friedgut and Kalai [24] seeks to relate two fundamental measures associated with the Fourier distribution: does there exist a universal constant $C > 0$ such that $\mathbb{H}(\hat{f}^2) \leq C \cdot \mathsf{Inf}(f)$, where $\mathbb{H}(\hat{f}^2)$ is the Shannon entropy of the Fourier distribution of f and $\mathsf{Inf}(f)$ is the total influence of f?

In this paper we present three new contributions towards the FEI conjecture:

(i) Our first contribution shows that $\mathbb{H}(\hat{f}^2) \leq 2 \cdot \mathsf{aUC}^{\oplus}(f)$, where $\mathsf{aUC}^{\oplus}(f)$ is the average unambiguous parity-certificate complexity of f. This improves upon several bounds shown by Chakraborty et al. [16]. We further improve this bound for unambiguous DNFs.

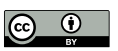
(ii) We next consider the weaker *Fourier Min-entropy-Influence* (FMEI) conjecture posed by O'Donnell and others [43, 40] which asks if $\mathbb{H}_\infty(\hat{f}^2) \leq C \cdot \mathsf{Inf}(f)$, where $\mathbb{H}_\infty(\hat{f}^2)$ is the min-entropy of the Fourier distribution. We show $\mathbb{H}_\infty(\hat{f}^2) \leq 2 \cdot \mathsf{C}^{\oplus}_{\min}(f)$, where $\mathsf{C}^{\oplus}_{\min}(f)$ is the minimum parity certificate complexity of f. We also show that for all $\varepsilon \geq 0$, we have $\mathbb{H}_\infty(\hat{f}^2) \leq 2\log(\|\widehat{f}\|_{1,\varepsilon}/(1-\varepsilon))$, where $\|\widehat{f}\|_{1,\varepsilon}$ is the approximate spectral norm of f. As a corollary, we verify the FMEI conjecture for the class of read-k DNFs (for constant k).

(iii) Our third contribution is to better understand implications of the FEI conjecture for the structure of polynomials that 1/3-approximate a Boolean function on the Boolean cube. We pose a conjecture: no *flat polynomial* (whose non-zero Fourier coefficients have the same magnitude) of degree d and sparsity $2^{\omega(d)}$ can 1/3-approximate a Boolean function. This conjecture is known to be true assuming FEI and we prove the conjecture unconditionally (i.e., without assuming the FEI conjecture) for a class of polynomials. We discuss an intriguing connection between our conjecture and the constant for the Bohnenblust-Hille inequality, which has been extensively studied in functional analysis.

2012 ACM Subject Classification Computing methodologies → Representation of Boolean functions; Theory of computation → Models of computation; Mathematics of computing → Information theory

Keywords and phrases Fourier analysis of Boolean functions, FEI conjecture, query complexity, polynomial approximation, approximate degree, certificate complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.45



37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020).
Editors: Christophe Paul and Markus Bläser; Article No. 45; pp. 45:1–45:19

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

Related Version A full version [5] of the paper is available at `https://arxiv.org/abs/1809.09819`.

Funding *Srinivasan Arunachalam*: Work done when at QuSoft, CWI, Amsterdam, supported by ERC Consolidator Grant 615307 QPROGRESS and MIT-IBM Watson AI Lab under the project *Machine Learning in Hilbert space*.
Michal Kouckỳ: Partially supported by ERC Consolidator Grant 616787 LBCAD, and GAČR grant 19-27871X.
Nitin Saurabh: Part of the work was done when the author was at IUUK, Prague and supported by the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement no. 616787.
Ronald de Wolf: Partially supported by ERC Consolidator Grant 615307 QPROGRESS (ended Feb 2019) and by NWO under QuantERA project QuantAlgo 680-91-034 and the Quantum Software Consortium.

Acknowledgements Part of this work was carried out when NS and SC visited CWI, Amsterdam and SA was part of MIT (supported by MIT-IBM Watson AI Lab under the project *Machine Learning in Hilbert Space*) and University of Bristol (partially supported by EPSRC grant EP/L021005/1). SA thanks Ashley Montanaro for his hospitality. NS and SC would like to thank Satya Lokam for many helpful discussions on the Fourier entropy-Influence conjecture. SA and SC thank Jop Briët for pointing us to the literature on unbalancing lights and many useful discussions regarding Section 3.3. We also thank Penghui Yao and Avishay Tal for discussions during the course of this project, and Fernando Vieira Costa Júnior for pointing us to the reference [6]. Finally, we thank the anonymous reviewers for many helpful comments.

1 Introduction

Boolean functions $f : \{-1,1\}^n \to \{-1,1\}$ naturally arise in many areas of theoretical computer science and mathematics such as learning theory, complexity theory, quantum computing, inapproximability, graph theory, extremal combinatorics, etc. Fourier analysis over the Boolean cube $\{-1,1\}^n$ is a powerful technique that has been used often to analyze problems in these areas. For a survey on the subject, see [40, 54]. One of the most important and longstanding open problems in this field is the *Fourier Entropy-Influence* (FEI) conjecture, first formulated by Ehud Friedgut and Gil Kalai in 1996 [24]. The FEI conjecture seeks to relate the following two fundamental properties of a Boolean function f: the *Fourier entropy* of f and the *total influence* of f, which we define now.

For a Boolean function $f : \{-1,1\}^n \to \{-1,1\}$, Parseval's identity relates the *Fourier coefficients* $\{\widehat{f}(S)\}_S$ and the values $\{f(x)\}_x$ by

$$\sum_{S\subseteq[n]} \widehat{f}(S)^2 = \mathbb{E}_x[f(x)^2] = 1,$$

where the expectation is taken uniformly over the Boolean cube $\{-1,1\}^n$. An immediate implication of this equality is that the squared Fourier coefficients $\{\widehat{f}(S)^2 : S \subseteq [n]\}$ can be viewed as a *probability distribution* over subsets $S \subseteq [n]$, which we often refer to as the *Fourier distribution*. The *Fourier entropy of* f (denoted $\mathbb{H}(\hat{f}^2)$) is then defined as the Shannon entropy of the Fourier distribution, i.e.,

$$\mathbb{H}(\hat{f}^2) := \sum_{S\subseteq[n]} \widehat{f}(S)^2 \log \frac{1}{\widehat{f}(S)^2}.$$

The *total influence of* f (denoted $\mathsf{Inf}(f)$) measures the *expected size* of a subset $S \subseteq [n]$,

where the expectation is taken according to the Fourier distribution, i.e.,

$$\mathsf{Inf}(f) = \sum_{S\subseteq[n]} |S|\ \widehat{f}(S)^2.$$

Combinatorially $\mathsf{Inf}(f)$ is the same as the average sensitivity $\mathsf{as}(f)$ of f. In particular, for $i \in [n]$, define $\mathsf{Inf}_i(f)$ to be the probability that on a uniformly random input flipping the i-th bit changes the function value. Then, $\mathsf{Inf}(f)$ is defined to be $\sum_{i=1}^{n} \mathsf{Inf}_i(f)$.

Intuitively, the Fourier entropy measures how "spread out" the Fourier distribution is over the 2^n subsets of $[n]$ and the total influence measures the concentration of the Fourier distribution on the "high" level coefficients. Informally, the FEI conjecture states that Boolean functions whose Fourier distribution is well "spread out" (i.e., functions with large Fourier entropy) must have significant Fourier weight on the high-degree monomials (i.e., their total influence is large). Formally, the FEI conjecture can be stated as follows:

▶ **Conjecture 1.1** (FEI Conjecture). *There exists a universal constant $C > 0$ such that for every Boolean function $f : \{-1,1\}^n \to \{-1,1\}$,*

$$\mathbb{H}(\hat{f}^2) \leq C \cdot \mathsf{Inf}(f). \tag{1}$$

The original motivation of Friedgut and Kalai for the FEI conjecture came from studying threshold phenomena of monotone graph properties in random graphs [24]. For example, resolving the FEI conjecture would imply that every threshold interval of a monotone graph property on n vertices is of length at most $c(\log n)^{-2}$ (for some universal constant $c > 0$). The current best upper bound, proven by Bourgain and Kalai [11], is $c_\varepsilon(\log n)^{-2+\varepsilon}$ for every $\varepsilon > 0$.

Besides this application, the FEI conjecture is known to imply the famous Kahn-Kalai-Linial theorem [30] (otherwise referred to as the KKL theorem). The KKL theorem was one of the first major applications of Fourier analysis to understanding properties of Boolean functions and has since found many application in various areas of theoretical computer science.

▶ **Theorem 1.2** (KKL theorem). *For every $f : \{-1,1\}^n \to \{-1,1\}$, there exists an $i \in [n]$ such that $\mathsf{Inf}_i(f) \geq \mathsf{Var}(f) \cdot \Omega\left(\frac{\log n}{n}\right)$.*

See Section 2 for the definitions of these quantities. Another motivation to study the FEI conjecture is that a positive answer to this conjecture would resolve the notoriously hard conjecture of Mansour [37] from 1995.

▶ **Conjecture 1.3** (Mansour's conjecture). *Suppose $f : \{-1,1\}^n \to \{-1,1\}$ is computed by a t-term* DNF.[1] *Then for every $\varepsilon > 0$, there exists a family $\mathcal{T}$ of subsets of $[n]$ such that $|\mathcal{T}| \leq t^{O(1/\varepsilon)}$ (i.e., size of $\mathcal{T}$ is polynomial in t) and $\sum_{T\in\mathcal{T}} \widehat{f}(T)^2 \geq 1 - \varepsilon$.*

A positive answer to Mansour's conjecture, along with the query algorithm of Gopalan et al. [26], would resolve a long-standing open question in computational learning theory of agnostically learning DNFs under the uniform distribution in polynomial time (up to any constant accuracy).

More generally, the FEI conjecture implies that every Boolean function can be approximated (in ℓ_2-norm) by *sparse* polynomials over $\{-1,1\}$. In particular, for a Boolean function

[1] A t-term DNF is a disjunction of at most t conjunctions of variables and their negations.

f and $\varepsilon > 0$, the FEI conjecture implies the existence of a polynomial p with $2^{O(\mathsf{Inf}(f)/\varepsilon)}$ monomials such that $\mathbb{E}_x[(f(x)-p(x))^2] \le \varepsilon$. The current best known bound in this direction is $2^{O(\mathsf{Inf}(f)^2/\varepsilon^2)}$, proven by Friedgut [23].[2]

Given the inherent difficulty in answering the FEI conjecture for arbitrary Boolean functions, there have been many recent works studying the conjecture for specific classes of Boolean functions. We give a brief overview of these results in the next section. Alongside the pursuit of resolving the FEI conjecture, O'Donnell and others [43, 40] have asked if a weaker question than the FEI conjecture, the *Fourier Min-entropy-Influence* (FMEI) conjecture can be resolved. The FMEI conjecture asks if the entropy-influence inequality in Eq. (1) holds when the entropy of the Fourier distribution is replaced by the *min-entropy* of the Fourier distribution (denoted $\mathbb{H}_\infty(\hat{f}^2)$). The min-entropy of $\{\widehat{f}(S)^2\}_S$ is defined as

$$\mathbb{H}_\infty(\hat{f}^2) := \min_{\substack{S\subseteq[n]:\\ \widehat{f}(S)\neq 0}} \left\{ \log \frac{1}{\widehat{f}(S)^2} \right\}$$

and thus it is easily seen that $\mathbb{H}_\infty(\hat{f}^2) \le \mathbb{H}(\hat{f}^2)$. In fact, $\mathbb{H}_\infty(\hat{f}^2)$ could be much smaller compared to $\mathbb{H}(\hat{f}^2)$. For instance, consider the function $f(x) := x_1 \vee \mathsf{IP}(x_1,\dots,x_n)$; then $\mathbb{H}_\infty(\hat{f}^2) = O(1)$ whereas $\mathbb{H}(\hat{f}^2) = \Omega(n)$. ($\mathsf{IP}$ is the inner-product-mod-2 function.) So the FMEI conjecture could be strictly weaker than the FEI conjecture, making it a natural candidate to resolve first.

▶ **Conjecture 1.4** (FMEI Conjecture)**.** *There exists a universal constant $C > 0$ such that for every Boolean function $f : \{-1,1\}^n \to \{-1,1\}$, we have $\mathbb{H}_\infty(\hat{f}^2) \le C \cdot \mathsf{Inf}(f)$.*

Another way to formulate the FMEI conjecture is, suppose $f : \{-1,1\}^n \to \{-1,1\}$, then does there exist a Fourier coefficient $\widehat{f}(S)$ such that $|\widehat{f}(S)| \ge 2^{-O(\mathsf{Inf}(f))}$? By the *granularity* of Fourier coefficients it is well-known that *every* Fourier coefficient of a Boolean function f is an integral multiple of $2^{-\mathsf{deg}(f)}$, see [40, Exercise 1.11] for a proof of this. (Here the $\mathsf{deg}(f)$ refers to the degree of the unique multilinear polynomial that represents f.) The FMEI conjecture asks if we can prove a lower bound of $2^{-O(\mathsf{Inf}(f))}$ on *any one* Fourier coefficient, and even this remains open. Proving the FMEI conjecture seems to require proving interesting structural properties of Boolean functions. In fact, as observed by [43], the FMEI conjecture suffices to imply the KKL theorem.

Understanding the min-entropy of a Fourier distribution is important in its own right too. It was observed by Akavia et al. [2] that for a circuit class $\mathcal{C}$, tighter relations between min-entropy of $f \in \mathcal{C}$ and f_A defined as $f_A(x) := f(Ax)$, for an arbitrary linear transformation A, could enable us to translate lower bounds against the class $\mathcal{C}$ to the class $\mathcal{C} \circ \mathsf{MOD}_2$. In particular, they conjectured that min-entropy of f_A is only polynomially larger than f when $f \in \mathsf{AC}^0[\mathsf{poly}(n), O(1)]$. ($\mathsf{AC}^0[s,d]$ is the class of unbounded fan-in circuits of size at most s and depth at most d.) It is well-known that when $f \in \mathsf{AC}^0[s,d]$, $\mathbb{H}_\infty(\hat{f}^2)$ is at most $O((\log s)^{d-1} \cdot \log\log s)$ [35, 10, 51]. Depending on the tightness of the relationship between $\mathbb{H}_\infty(\hat{f}^2)$ and $\mathbb{H}_\infty(\widehat{f_A}^2)$, one could obtain near-optimal lower bound on the size of $\mathsf{AC}^0[s,d]\circ\mathsf{MOD}_2$ circuits computing IP (inner-product-mod-2). This problem has garnered a lot of attention in recent times for a variety of reasons [48, 46, 2, 18, 17]. The current best known lower bound for IP against $\mathsf{AC}^0[s,d]\circ\mathsf{MOD}_2$ is quadratic when $d = 4$, and only super-linear for all $d = O(1)$ [17].

[2] Friedgut's Junta theorem says that f is ε-close to a junta on $2^{O(\mathsf{Inf}(f)/\varepsilon)}$ variables. We refer to [40, Section 9.6, page 269, Friedgut's Junta Theorem] for details.

Organization. We end this introduction with an overview of prior work on the FEI and FMEI conjecture in Section 1.1. We then describe our contributions and sketch the proofs in Section 3. We conclude in Section 4. Due to lack of space the proofs have been omitted. We refer to the full version [5] for any omission from this version.

1.1 Prior work

After Friedgut and Kalai [24] posed the FEI conjecture in 1996, there was not much work done towards resolving it, until the work of Klivans et al. [33] in 2010. They showed that the FEI conjecture holds true for random DNF formulas. Since then, there have been many significant steps taken in the direction of resolving the FEI conjecture. We review some recent works here, referring the interested reader to the blog post of Kalai [31] for additional discussions on the FEI conjecture.

The FEI conjecture is known to be true when we replace the universal constant C with $\log n$ in Eq. (1). In fact we know $\mathbb{H}(\hat{f}^2) \leq O(\mathsf{Inf}(f) \cdot \log n)$ for *real-valued* functions $f : \{-1,1\}^n \to \mathbb{R}$ (see [43, 32] for a proof and [16] for an improvement of this statement).[3] If we strictly require C to be a universal constant, then the FEI conjecture is known to be false for real-valued functions. Instead, for real-valued functions an analogous statement called the *logarithmic Sobolev Inequality* [28] is known to be true. The logarithmic Sobolev inequality states that for every $f : \{-1,1\}^n \to \mathbb{R}$, we have $\mathsf{Ent}(f^2) \leq 2 \cdot \mathsf{Inf}(f)$, where $\mathsf{Ent}(f)$ is defined as $\mathsf{Ent}(f) = \mathbb{E}[f \ln(f)] - \mathbb{E}[f] \ln(\mathbb{E}[f])$, where the expectation is taken over uniform $x \in \{-1,1\}^n$.

Restricting to Boolean functions, the FEI conjecture is known to be true for the "standard" functions that arise often in analysis, such as AND, OR, Majority, Parity, Bent functions and Tribes. There have been many works on proving the FEI conjecture for specific classes of Boolean functions. O'Donnell et al. [43] showed that the FEI conjecture holds for symmetric Boolean functions and read-once decision trees. Keller et al. [32] studied a generalization of the FEI conjecture when the Fourier coefficients are defined on biased product measures on the Boolean cube. Then, Chakraborty et al. [16] and O'Donnell and Tan [41], independently and simultaneously, proved the FEI conjecture for read-once formulas. In fact, O'Donnell and Tan proved an interesting composition theorem for the FEI conjecture (we omit the definition of composition theorem here, see [41] for more). For general Boolean functions, Chakraborty et al. [16] gave several upper bounds on the Fourier entropy in terms of combinatorial quantities larger than the total influence, e.g., average decision tree depth, etc., and sometimes even quantities that could be much smaller than influence, namely, average parity-decision tree depth.

Later Wan et al. [53] used Shannon's source coding theorem [49] (which characterizes entropy) to establish the FEI conjecture for read-k decision trees for constant k. Using their novel interpretation of the FEI conjecture they also reproved O'Donnell-Tan's composition theorem in an elegant way. Recently, Shalev [47] improved the constant in the FEI inequality for read-k decision trees, and further verified the conjecture when either the influence is *too low*, or the entropy is *too high*. The FEI conjecture is also verified for random Boolean functions by Das et al. [20] and for random linear threshold functions (LTFs) by Chakraborty et al. [15].

[3] For Boolean functions, the $\log n$-factor was improved by [27] to $\log(\mathsf{s}(f))$ (where $\mathsf{s}(f)$ is the sensitivity of the Boolean function f).

There has also been some work in giving lower bounds on the constant C in the FEI conjecture. Hod [29] gave a lower bound of $C > 6.45$ (the lower bound holds even when considering the class of monotone functions), improving upon the lower bound of O'Donnell and Tan [41].

However, there has not been much work on the FMEI conjecture. It was observed in [43, 15] that the KKL theorem implies the FMEI conjecture for monotone functions and linear threshold functions. Finally, the FMEI conjecture for "regular" read-k DNFs was recently established by Shalev [47].

2 Preliminaries

Notation. We denote the set $\{1, 2, \ldots, n\}$ by $[n]$. A *partial assignment* of $[n]$ is a map $\tau : [n] \to \{-1, 1, *\}$. Define $|\tau| = |\tau^{-1}(1) \cup \tau^{-1}(-1)|$. A subcube of the Boolean cube $\{-1,1\}^n$ is a set of $x \in \{-1,1\}^n$ that agrees with some partial assignment τ, i.e., $\{x \in \{-1,1\}^n : x_i = \tau(i)$ for every i with $\tau(i) \neq *\}$.

Fourier Analysis. We recall some definitions and basic facts from analysis of Boolean functions, referring to [40, 54] for more. Consider the space of all functions from $\{-1,1\}^n$ to $\mathbb{R}$ equipped with the inner product defined as

$$\langle f, g\rangle := \mathbb{E}_x[f(x)g(x)] = \frac{1}{2^n}\sum_{x\in\{-1,1\}^n} f(x)g(x).$$

For $S \subseteq [n]$, the character function $\chi_S : \{-1,1\}^n \to \{-1,1\}$ is defined as $\chi_S(x) := \prod_{i\in S} x_i$. Then the set of character functions $\{\chi_S\}_{S\subseteq[n]}$ forms an orthonormal basis for the space of all real-valued functions on $\{-1,1\}^n$. Hence, every real-valued function $f : \{-1,1\}^n \to \mathbb{R}$ has a unique Fourier expansion

$$f(x) = \sum_{S\subseteq[n]} \widehat{f}(S)\chi_S(x).$$

The *degree* of f, denoted $\mathsf{deg}(f)$, is defined as $\max\{|S| : \widehat{f}(S) \neq 0\}$. The *spectral norm* of f is defined to be $\sum_S |\widehat{f}(S)|$. The *Fourier weight* of a function f on a set of coefficients $\mathcal{S} \subseteq 2^{[n]}$ is defined as $\sum_{S\in\mathcal{S}} \widehat{f}(S)^2$. The *approximate spectral norm* of a Boolean function f is defined as

$$\|\widehat{f}\|_{1,\varepsilon} = \min\Big\{\sum_S |\widehat{p}(S)| : |p(x) - f(x)| \leq \varepsilon \text{ for every } x \in \{-1,1\}^n\Big\}.$$

We note a well-known fact that follows from the orthonormality of the character functions.

▶ **Fact 2.1** (Plancherel's Theorem). *For any f, g: $\{-1,1\}^n \to \mathbb{R}$,*

$$\mathbb{E}_x[f(x)g(x)] = \sum_S \widehat{f}(S)\widehat{g}(S).$$

In particular, if f: $\{-1,1\}^n \to \{-1,1\}$ is Boolean-valued and $g = f$, we have Parseval's Identity $\sum_S \widehat{f}(S)^2 = \mathbb{E}[f(x)^2]$, which in turn equals 1. Hence $\sum_S \widehat{f}(S)^2 = 1$ and we can view $\{\widehat{f}(S)^2\}_S$ as a probability distribution, which allows us to discuss the Fourier entropy and min-entropy of the distribution $\{\widehat{f}(S)^2\}_S$, defined as

▶ **Definition 2.2.** *For a Boolean function $f : \{-1,1\}^n \to \{-1,1\}$, its Fourier entropy (denoted $\mathbb{H}(\hat{f}^2)$) and min-entropy (denoted $\mathbb{H}_\infty(\hat{f}^2)$) are*

$$\mathbb{H}(\hat{f}^2) := \sum_{S\subseteq[n]} \widehat{f}(S)^2 \log \frac{1}{\widehat{f}(S)^2}, \quad \text{and} \quad \mathbb{H}_\infty(\hat{f}^2) := \min_{\substack{S\subseteq[n]:\\ \widehat{f}(S)\neq 0}} \left\{ \log \frac{1}{\widehat{f}(S)^2} \right\}.$$

Similarly, we can also define the Rényi Fourier entropy.

▶ **Definition 2.3** (Rényi Fourier entropy)**.** *For $f : \{-1,1\}^n \to \{-1,1\}$, $\alpha \geq 0$ and $\alpha \neq 1$, the Rényi Fourier entropy of f of order α is defined as*

$$\mathbb{H}_\alpha(\hat{f}^2) := \frac{1}{1-\alpha} \log \left(\sum_{S\subseteq[n]} |\widehat{f}(S)|^{2\alpha} \right).$$

It is known that in the limit as $\alpha \to 1$, $\mathbb{H}_\alpha(\hat{f}^2)$ is the (Shannon) Fourier entropy $\mathbb{H}(\hat{f}^2)$ (see [19, Chapter 17, Section 8]) and when $\alpha \to \infty$, observe that $\mathbb{H}_\alpha(\hat{f}^2)$ converges to $\mathbb{H}_\infty(\hat{f}^2)$. It is easily seen that $\mathbb{H}_\infty(\hat{f}^2) \leq \mathbb{H}(\hat{f}^2) \leq \mathbb{H}_{\frac{1}{2}}(\hat{f}^2) \leq \mathbb{H}_0(\hat{f}^2)$.

For $f : \{-1,1\}^n \to \{-1,1\}$, the *influence* of a coordinate $i \in [n]$, denoted $\mathsf{Inf}_i(f)$, is defined as

$$\mathsf{Inf}_i(f) = \Pr_{x\in\{-1,1\}^n}[f(x) \neq f(x^{(i)})] = \mathbb{E}_x\left[\left(\frac{f(x)-f(x^{(i)})}{2}\right)^2\right],$$

where the probability and expectation is taken according to the uniform distribution on $\{-1,1\}^n$ and $x^{(i)}$ is x with the i-th bit flipped. The *total influence* of f, denoted $\mathsf{Inf}(f)$, is

$$\mathsf{Inf}(f) = \sum_{i\in[n]} \mathsf{Inf}_i(f).$$

In terms of the Fourier coefficients of f, it can be shown, e.g., [30], that $\mathsf{Inf}_i(f) = \sum_{S\ni i} \widehat{f}(S)^2$, and therefore

$$\mathsf{Inf}(f) = \sum_{S\subseteq[n]} |S|\widehat{f}(S)^2.$$

The *variance* of a real-valued function f is given by $\mathsf{Var}(f) = \sum_{S\neq\emptyset} \widehat{f}(S)^2$. It easily follows that $\mathsf{Var}(f) \leq \mathsf{Inf}(f)$. We will also need the following version of the well-known KKL theorem.

▶ **Theorem 2.4** (KKL Theorem, [30])**.** *There exists a universal constant $c > 0$ such that for every $f\colon \{-1,1\}^n \to \{-1,1\}$, we have*

$$\mathsf{Inf}(f) \geq c \cdot \mathsf{Var}(f) \cdot \log \frac{1}{\max_i \mathsf{Inf}_i(f)}.$$

We now introduce some basic complexity measures of Boolean functions which we use often, referring to [13] for more.

Sensitivity. For $x \in \{-1,1\}^n$, the *sensitivity* of f at x, denoted $\mathsf{s}_f(x)$, is defined to be the number of neighbors y of x in the Boolean hypercube (i.e., y is obtained by flipping *exactly* one bit of x) such that $f(y) \neq f(x)$. The sensitivity $\mathsf{s}(f)$ of f is $\max_x\{\mathsf{s}_f(x)\}$. The *average sensitivity* $\mathsf{as}(f)$ of f is defined to be $\mathbb{E}_x[\mathsf{s}_f(x)]$. By the linearity of expectation observe that

$$\mathbb{E}_x[\mathsf{s}_f(x)] = \sum_{i=1}^n \Pr_x[f(x) \neq f(x^{(i)})] = \sum_{i=1}^n \mathsf{Inf}_i(f) = \mathsf{Inf}(f),$$

so the average sensitivity of f equals the total influence of f. As a result, the FEI conjecture asks if $\mathbb{H}(\hat{f}^2) \leq C \cdot \mathsf{as}(f)$ for every Boolean function f.

Certificate complexity. For $x \in \{-1,1\}^n$, the *certificate complexity* of f at x, denoted $\mathsf{C}(f,x)$, is the minimum number of bits in x that needs to be fixed to ensure that the value of f is constant. The certificate complexity $\mathsf{C}(f)$ of f is $\max_x\{\mathsf{C}(f,x)\}$. The minimum certificate complexity of f is $\mathsf{C}_{\min}(f) = \min_x\{\mathsf{C}(f,x)\}$. The 0-certificate complexity $\mathsf{C}^0(f)$ of f is $\max_{x:f(x)=1}\{\mathsf{C}(f,x)\}$. Similarly, the 1-certificate complexity $\mathsf{C}^1(f)$ of f is $\max_{x:f(x)=-1}\{\mathsf{C}^1(f,x)\}$. Observe that for every $x \in \{-1,1\}^n$, $\mathsf{s}(f,x) \le \mathsf{C}(f,x)$. This gives $\mathsf{s}(f) \le \mathsf{C}(f)$ and $\mathsf{as}(f) \le \mathsf{aC}(f)$ where $\mathsf{aC}(f)$ denotes the *average* certificate complexity of f. As before, the average is taken with respect to the uniform distribution on $\{-1,1\}^n$.

Parity certificate complexity. Analogously, we define the *parity certificate complexity* $\mathsf{C}^{\oplus}(f,x)$ of f at x as the minimum number of parities on the input variables one has to fix in order to fix the value of f at x, i.e.,

$$\mathsf{C}^{\oplus}(f,x) \coloneqq \min\{\mathsf{co\text{-}dim}(H) \mid H \text{ is an affine subspace on which } f \text{ is constant and } x \in H\},$$

where $\mathsf{co\text{-}dim}(H)$ is the *co-dimension* of the affine subspace H. It is easily seen that $\mathsf{C}^{\oplus}(f,x) \le \mathsf{C}(f,x)$. We also define $\mathsf{C}^{\oplus}(f) \coloneqq \max_x\{\mathsf{C}^{\oplus}(f,x)\}$, and $\mathsf{C}^{\oplus}_{\min}(f) \coloneqq \min_x \mathsf{C}^{\oplus}(f,x)$.

Unambiguous certificate complexity. We now define the *unambiguous certificate complexity* of f. Let $\tau : [n] \to \{-1,1,*\}$ be a partial assignment. We refer to $S_\tau = \{x \in \{-1,1\}^n : x_i = \tau(i) \text{ for every } i \in [n]\backslash\tau^{-1}(*)\}$ as the subcube generated by τ. We call $C \subseteq \{-1,1\}^n$ a *subcube* of $\{-1,1\}^n$ if there exists a partial assignment τ such that $C = S_\tau$ and the co-dimension of C is the number of bits fixed by τ, i.e., $\mathsf{co\text{-}dim}(C) = |\{i \in [n] : \tau(i) \neq *\}|$. A set of subcubes $\mathcal{C} = \{C_1,\dots,C_m\}$ *partitions* $\{-1,1\}^n$ if the subcubes are disjoint and they cover $\{-1,1\}^n$, i.e., $C_i \cap C_j = \emptyset$ for $i \neq j$ and $\cup_i C_i = \{-1,1\}^n$.

An *unambiguous certificate* $\mathcal{U} = \{C_1,\dots,C_m\}$ (also referred to as a *subcube partition*) is a set of subcubes partitioning $\{-1,1\}^n$. We say $\mathcal{U}$ computes a Boolean function $f : \{-1,1\}^n \to \{-1,1\}$ if f is constant on each C_i (i.e., $f(x)$ is the same for all $x \in C_i$). For an unambiguous certificate $\mathcal{U}$, the *unambiguous certificate complexity* on input x (denoted $\mathsf{UC}(\mathcal{U},x)$), equals $\mathsf{co\text{-}dim}(C_i)$ for the C_i satisfying $x \in C_i$. Define the *average unambiguous certificate complexity* of f with respect to $\mathcal{U}$ as $\mathsf{aUC}(f,\mathcal{U}) \coloneqq \mathbb{E}_x[\mathsf{UC}(\mathcal{U},x)]$. Then, the *average unambiguous certificate complexity* of f is defined as

$$\mathsf{aUC}(f) \coloneqq \min_{\mathcal{U}} \mathsf{aUC}(f,\mathcal{U}),$$

where the minimization is over all unambiguous certificates for f. Finally, the *unambiguous certificate complexity of* f is

$$\mathsf{UC}(f) \coloneqq \min_{\mathcal{U}} \max_x \mathsf{UC}(\mathcal{U},x).$$

Note that since unambiguous certificates are more restricted than general certificates, we have $\mathsf{C}(f) \le \mathsf{UC}(f)$.

An unambiguous $\oplus$-certificate $\mathcal{U} = \{C_1,\dots,C_m\}$ for f is defined to be a collection of monochromatic *affine subspaces* that together partition the space $\{-1,1\}^n$. It is easily seen that a subcube is also an affine subspace. Analogously, for an unambiguous $\oplus$-certificate $\mathcal{U}$, on an input x, $\mathsf{UC}^{\oplus}(\mathcal{U},x) \coloneqq \mathsf{co\text{-}dim}(C_i)$ for the C_i satisfying $x \in C_i$, and $\mathsf{aUC}^{\oplus}(f,\mathcal{U}) \coloneqq \mathbb{E}_x[\mathsf{UC}^{\oplus}(\mathcal{U},x)]$. Similarly, we define $\mathsf{aUC}^{\oplus}(f)$ and $\mathsf{UC}^{\oplus}(f)$.

DNFs. A DNF (disjunctive normal form) is a disjunction (OR) of conjunctions (ANDs) of variables and their negations. An *unambiguous* DNF is a DNF that satisfies the additional property that: on every (-1)-input, exactly one of the conjunctions outputs -1.

Approximate degree. The *ε-approximate degree* of $f : \{-1,1\}^n \to \mathbb{R}$, denoted $\deg_\varepsilon(f)$, is defined to be the minimum degree among all multilinear real polynomials p such that $|f(x) - p(x)| \leq \varepsilon$ for all $x \in \{-1,1\}^n$. Usually ε is chosen to be $1/3$, but it can be chosen to be any constant in $(0,1)$, without significantly changing the model.

Deterministic decision tree. A deterministic decision tree for $f : \{-1,1\}^n \to \{-1,1\}$ is a rooted binary tree where each node is labelled by $i \in [n]$ and the leaves are labelled with an output bit $\{-1,1\}$. On input $x \in \{-1,1\}^n$, the tree proceeds at the i-th node by evaluating the bit x_i and continuing with the subtree corresponding to the value of x_i. Once a leaf is reached, the tree outputs a bit. We say that a deterministic decision tree computes f if for all $x \in \{-1,1\}^n$ its output equals $f(x)$.

A parity-decision tree for f is similar to a deterministic decision tree, except that each node in the tree is labelled by a subset $S \subseteq [n]$. On input $x \in \{-1,1\}^n$, the tree proceeds at the i-th node by evaluating the parity of the bits x_i for $i \in S$ and continues with the subtree corresponding to the value of $\oplus_{i \in S} x_i$. Note that if the subsets at each node have size $|S| = 1$, then we get the standard deterministic decision tree model.

Randomized decision tree. A randomized decision tree for f is a probability distribution R_μ over deterministic decision trees for f. On input x, a decision tree is chosen according to R_μ, which is then evaluated on x. The complexity of the randomized tree is the largest depth among all deterministic trees with non-zero probability of being sampled according to R_μ. One can similarly define a randomized parity-decision tree as a probability distribution $R_\mu^\oplus$ over deterministic parity-decision trees for f.

We say that a randomized decision tree computes f with bounded-error if for all $x \in \{-1,1\}^n$ its output equals $f(x)$ with probability at least $2/3$. $R_2(f)$ (resp. $R_2^\oplus(f)$) denotes the complexity of the optimal randomized (resp. parity) decision tree that computes f with bounded-error, i.e., errs with probability at most $1/3$.

Information Theory. We now recall the following consequence of the law of large numbers, called the *Asymptotic Equipartition Property (AEP)* or *the Shannon-McMillan-Breiman theorem*. See Chapter 3 in the book [19] for more details.

▶ **Theorem 2.5** (Asymptotic Equipartition Property (AEP) Theorem)**.** *Let* $\mathbf{X}$ *be a random variable drawn from a distribution* P *and suppose* $\mathbf{X_1}, \mathbf{X_2}, \ldots, \mathbf{X_M}$ *are independently and identically distributed copies of* $\mathbf{X}$*, then*

$$-\frac{1}{M} \log P(\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_M) \longrightarrow \mathbb{H}(\mathbf{X})$$

in probability as $M \to \infty$*.*

▶ **Definition 2.6.** *Fix* $\varepsilon \geq 0$*. The* typical set $T_\varepsilon^{(M)}(\mathbf{X})$ *with respect to a distribution* P *is defined to be the set of sequences* $(x_1, x_2, \ldots, x_M) \in \mathbf{X_1} \times \mathbf{X_2} \times \cdots \times \mathbf{X_M}$ *such that*

$$2^{-M(\mathbb{H}(\mathbf{X})+\varepsilon)} \leq P(x_1, x_2, \ldots, x_M) \leq 2^{-M(\mathbb{H}(\mathbf{X})-\varepsilon)}.$$

The following properties of the typical set follows from the AEP.

▶ **Theorem 2.7** ([19, Theorem 3.1.2])**.** *Let* $\varepsilon \geq 0$ *and* $T_\varepsilon^{(M)}(\mathbf{X})$ *be a typical set with respect to* P*, then*

(i) $|T_\varepsilon^{(M)}(\mathbf{X})| \leq 2^{M(\mathbb{H}(\mathbf{X})+\varepsilon)}$.

(ii) *Suppose* $x_1, \ldots, x_M$ *are drawn i.i.d. according to* $\mathbf{X}$*, then* $\Pr[(x_1, \ldots, x_M) \in T_\varepsilon^{(M)}(\mathbf{X})] \geq 1 - \varepsilon$ *for* M *sufficiently large.*
(iii) $|T_\varepsilon^{(M)}(\mathbf{X})| \geq (1-\varepsilon)2^{M(\mathbb{H}(\mathbf{X})-\varepsilon)}$ *for* M *sufficiently large.*

We also require the following stronger version of typical sequences and asymptotic equipartition property.

▶ **Definition 2.8** ([19, Chapter 11, Section 2])**.** *Let* $\mathbf{X}$ *be a random variable drawn according to a distribution* P*. Fix* $\varepsilon > 0$*. The* strongly typical set $T_\varepsilon^{*(M)}(\mathbf{X})$ *is defined to be the set of sequences* $\rho = (x_1, x_2, \ldots, x_M) \in \mathbf{X_1} \times \mathbf{X_2} \times \cdots \times \mathbf{X_M}$ *such that* $N(x;\rho) = 0$ *if* $P(x) = 0$*, and otherwise*

$$\left| \frac{N(x;\rho)}{M} - P(x) \right| \leq \frac{\varepsilon}{|\mathbf{X}|},$$

where $N(x;\rho)$ *is defined as the number of occurrences of* x *in* ρ*.*

The strongly typical set shares similar properties with its (weak) typical counterpart which we state now. See [19, Chapter 11, Section 2] for a proof of this theorem.

▶ **Theorem 2.9** (Strong AEP Theorem)**.** *Following the notation in Definition 2.8, let* $T_\varepsilon^{*(M)}(\mathbf{X})$ *be a strongly typical set. Then, there exists* $\delta > 0$ *such that* $\delta \to 0$ *as* $\varepsilon \to 0$*, and the following hold:*

(i) *Suppose* $x_1, \ldots, x_M$ *are drawn i.i.d. according to* $\mathbf{X}$*, then* $\Pr[(x_1, \ldots, x_M) \in T_\varepsilon^{*(M)}(\mathbf{X})] \geq 1 - \varepsilon$ *for* M *sufficiently large.*
(ii) *If* $(x_1, \ldots, x_M) \in T_\varepsilon^{*(M)}(\mathbf{X})$*, then*

$$2^{-M(\mathbb{H}(\mathbf{X})+\delta)} \leq P(x_1, \ldots, x_M) \leq 2^{-M(\mathbb{H}(\mathbf{X})-\delta)}.$$

(iii) *For* M *sufficiently large,*

$$(1-\varepsilon)2^{M(\mathbb{H}(\mathbf{X})-\delta)} \leq |T_\varepsilon^{*(M)}(\mathbf{X})| \leq 2^{M(\mathbb{H}(\mathbf{X})+\delta)}.$$

3 Our Contributions

Our contributions in this paper are threefold, which we elaborate on below:

3.1 Better upper bounds for the FEI conjecture

Our first and main contribution of this paper is to give a better upper bound on the Fourier entropy $\mathbb{H}(\hat{f}^2)$ in terms of $\mathsf{aUC}(f)$, the *average unambiguous certificate complexity* of f. Informally, the unambiguous certificate complexity $\mathsf{UC}(f)$ of f is similar to the standard certificate complexity measure, except that the collection of certificates is now required to be *unambiguous*, i.e., every input should be consistent with a *unique* certificate. In other words, an unambiguous certificate is a monochromatic subcube partition of the Boolean cube. By the average unambiguous certificate complexity, $\mathsf{aUC}(f)$, we mean the expected number of bits set by an unambiguous certificate on a uniformly random input.

There have been many recent works on query complexity, giving upper and lower bounds on $\mathsf{UC}(f)$ in terms of other combinatorial measures such as decision-tree complexity, sensitivity, quantum query complexity, etc., see [25, 4, 8] for more. It follows from definitions that $\mathsf{UC}(f)$ lower bounds decision tree complexity. However, it is known that $\mathsf{UC}(f)$ can be quadratically smaller than decision tree complexity [4]. Our main contribution here is an improved upper bound of *average unambiguous certificate complexity* $\mathsf{aUC}(f)$ on $\mathbb{H}(\hat{f}^2)$. This improves upon the previously known bound of average decision tree depth on $\mathbb{H}(\hat{f}^2)$ [16].

▶ **Theorem 3.1.** *Let $f : \{-1,1\}^n \to \{-1,1\}$ be a Boolean function. Then,*

$$\mathbb{H}(\hat{f}^2) \leq 2 \cdot \mathsf{aUC}(f).$$

A new and crucial ingredient employed in the proof of the theorem is an analog of the law of large numbers in information theory, usually referred to as the *Asymptotic Equipartition Property (AEP)* theorem (Theorem 2.5). Employing information-theoretic techniques for the FEI conjecture seems very natural given that the conjecture seeks to bound the entropy of a distribution. Indeed, Keller et al. [32, Section 3.1] envisioned a proof of the FEI conjecture itself using large deviation estimates and the tensor structure (explained below) in a stronger way, and Wan et al. [53] used Shannon's source coding theorem [49] to verify the conjecture for bounded-read decision trees.

In order to prove Theorem 3.1, we study the *tensorized* version of f, $f^M \colon \{-1,1\}^{Mn} \to \{-1,1\}$, which is defined as follows,

$$f^M(x^1,\ldots,x^M) := f(x^1_1,\ldots,x^1_n) \cdot f(x^2_1,\ldots,x^2_n) \cdots f(x^M_1,\ldots,x^M_n).$$

Similarly we define a *tensorized* version $\mathcal{C}^M$ of an unambiguous certificate $\mathcal{C}$ of f,[4] i.e., a direct product of M independent copies of $\mathcal{C}$. It is not hard to see that $\mathcal{C}^M$ is also an unambiguous certificate of f^M. To understand the properties of $\mathcal{C}^M$ we study $\mathcal{C}$ in a probabilistic manner. We observe that $\mathcal{C}$ naturally inherits a distribution $\mathbf{C}$ on its certificates when the underlying inputs $x \in \{-1,1\}^n$ are distributed uniformly. Using the asymptotic equipartition property with respect to $\mathbf{C}$, we infer that for every $\delta > 0$, there exists $M_0 > 0$ such that for all $M \geq M_0$, there are at most $2^{M(\mathsf{aUC}(f,\mathcal{C})+\delta)}$ certificates in $\mathcal{C}^M$ that together cover at least $1-\delta$ fraction of the inputs in $\{-1,1\}^{Mn}$. Furthermore, each of these certificates fixes at most $M(\mathsf{aUC}(f,\mathcal{C})+\delta)$ bits. Hence, a particular certificate can contribute to at most $2^{M(\mathsf{aUC}(f,\mathcal{C})+\delta)}$ Fourier coefficients of f^M. Combining both these bounds, all these certificates can overall contribute to at most $2^{2M(\mathsf{aUC}(f,\mathcal{C})+\delta)}$ Fourier coefficients of f^M. Let's denote this set of Fourier coefficients by $\mathcal{B}$. We then argue that the Fourier coefficients of f^M that are *not* in $\mathcal{B}$ have Fourier weight at most δ. This now allows us to bound the Fourier entropy of f^M as follows,

$$\mathbb{H}(\widehat{f^M}^2) \leq \log|\mathcal{B}| + \delta nM + \mathsf{H}(\delta),$$

where $\mathsf{H}(\delta)$ is the binary entropy function. Since $\mathbb{H}(\widehat{f^M}^2) = M \cdot \mathbb{H}(\hat{f}^2)$, we have

$$\mathbb{H}(\hat{f}^2) \leq 2(\mathsf{aUC}(f,\mathcal{C}) + \delta) + \delta n + \frac{\mathsf{H}(\delta)}{M}.$$

By the AEP theorem, note that $\delta \to 0$ as $M \to \infty$. Thus, taking the limit as $M \to \infty$ we obtain our theorem.

Looking finely into how certificates contribute to Fourier coefficients in the proof above, we further strengthen Theorem 3.1 by showing that we can replace $\mathsf{aUC}(f)$ by the *average unambiguous parity-certificate complexity* $\mathsf{aUC}^{\oplus}(f)$ of f. Here $\mathsf{aUC}^{\oplus}(f)$ is defined similar to $\mathsf{aUC}(f)$ except that instead of being defined in terms of monochromatic subcube partitions of f, we now partition the Boolean cube with monochromatic *affine subspaces*. (Observe that subcubes are also affine subspaces.) This strengthening also improves upon the previously known bound of average parity-decision tree depth on $\mathbb{H}(\hat{f}^2)$ [16]. It is easily seen that $\mathsf{aUC}^{\oplus}(f)$ lower bounds the average parity-decision tree depth.

[4] Recall an unambiguous certificate is a collection of certificates that partitions the Boolean cube $\{-1,1\}^n$.

▶ **Theorem 3.2.** *Let $f : \{-1,1\}^n \to \{-1,1\}$ be any Boolean function. Then,*

$$\mathbb{H}(\hat{f}^2) \leq 2 \cdot \mathsf{aUC}^{\oplus}(f).$$

The proof outline remains the same as in Theorem 3.1. However, a particular certificate in $\mathcal{C}^M$ no longer fixes just variables. Instead these parity certificates now fix parities over variables, and so potentially could involve all variables. Hence we cannot directly argue that all the certificates contribute to at most $2^{M(\mathsf{aUC}^{\oplus}(f,\mathcal{C})+\delta)}$ Fourier coefficients of f^M. Nevertheless, by the AEP theorem we still obtain that a typical parity-certificate fixes at most $M(\mathsf{aUC}^{\oplus}(f,\mathcal{C}) + \delta)$ parities. Looking closely at the Fourier coefficients that a parity-certificate can contribute to, we now argue that such coefficients must lie in the linear span of the parities fixed by the parity-certificate. Therefore, a typical parity-certificate can overall contribute to at most $2^{M(\mathsf{aUC}^{\oplus}(f,\mathcal{C})+\delta)}$ Fourier coefficients of f^M. The rest of the proof now follows analogously.

▶ Remark 3.3. As a corollary to the theorem we obtain that the FEI conjecture holds for the class of functions f with constant $\mathsf{aUC}^{\oplus}(f)$, and $\mathsf{Inf}(f) \geq 1$. That is, for a Boolean function f with $\mathsf{Inf}(f) \geq 1$, we have

$$\mathbb{H}(\hat{f}^2) \leq 2 \cdot \mathsf{aUC}^{\oplus}(f) \cdot \mathsf{Inf}(f).$$

We note that the reduction in [53, Proposition E.2] shows that removing the requirement $\mathsf{Inf}(f) \geq 1$ from the above inequality will prove the FEI conjecture for all Boolean functions with $\mathsf{Inf}(f) \geq \log n$. Furthermore, if we could show the FEI conjecture for Boolean functions f where $\mathsf{aUC}^{\oplus}(f) = \omega(1)$ is a slow-growing function of n, again the padding argument in [53] shows that we would be able to establish the FEI conjecture for all Boolean functions.

Further extension to unambiguous DNFs

Consider an unambiguous certificate $\mathcal{C} = \{C_1, \ldots, C_t\}$ of f. It covers both 1 and -1 inputs of f. Suppose $\{C_1, \ldots, C_{t_1}\}$ for some $t_1 < t$ is a partition of $f^{-1}(-1)$ and $\{C_{t_1+1}, \ldots, C_t\}$ is a partition of $f^{-1}(1)$. To represent f, it suffices to consider $\bigvee_{i=1}^{t_1} C_i$. This is a DNF representation of f with an additional property that $\{C_1, \ldots, C_{t_1}\}$ forms a partition of $f^{-1}(-1)$. We call such a representation an *unambiguous* DNF. In general, a DNF representation need not satisfy this additional property.

Using the equivalence of total influence and average sensitivity, one can easily observe that

$$\mathsf{Inf}(f) \leq 2 \cdot \min\left\{ \sum_{i=1}^{t_1} \mathsf{co\text{-}dim}(C_i) \cdot 2^{-\mathsf{co\text{-}dim}(C_i)}, \sum_{i=t_1+1}^{t} \mathsf{co\text{-}dim}(C_i) \cdot 2^{-\mathsf{co\text{-}dim}(C_i)} \right\} \leq \mathsf{aUC}(f,\mathcal{C}),$$

where $\mathsf{co\text{-}dim}(\cdot)$ denotes the co-dimension of an affine space. Note that the quantity $\sum_{i=1}^{t_1} \mathsf{co\text{-}dim}(C_i) \cdot 2^{-\mathsf{co\text{-}dim}(C_i)}$, in a certain sense, is "average unambiguous 1-certificate complexity" and, similarly, $\sum_{i=t_1+1}^{t} \mathsf{co\text{-}dim}(C_i) \cdot 2^{-\mathsf{co\text{-}dim}(C_i)}$ captures "average unambiguous 0-certificate complexity".

Building on our ideas from the main theorem in the previous section and using a *stronger* version of the AEP theorem (Theorem 2.9) we essentially establish the aforementioned improved bound of the smaller quantity between "average unambiguous 1-certificate complexity" and "average unambiguous 0-certificate complexity" on the Fourier entropy. Formally, we prove the following.

▶ **Theorem 3.4.** *Let $f : \{-1,1\}^n \to \{-1,1\}$ be a Boolean function and $\mathcal{C} = \{C_1, \ldots, C_t\}$ be a monochromatic affine subspace partition of $\{-1,1\}^n$ with respect to f such that $\{C_1, \ldots, C_{t_1}\}$ for some $t_1 < t$ is an affine subspace partition of $f^{-1}(-1)$ and $\{C_{t_1+1}, \ldots, C_t\}$ is an affine subspace partition of $f^{-1}(1)$. Further, $p := \Pr_x[f(x) = 1]$. Then,*

$$\mathbb{H}(\hat{f}^2) \leq \begin{cases} 2\left(\sum_{i=1}^{t_1} \mathsf{co\text{-}dim}(C_i) \cdot 2^{-\mathsf{co\text{-}dim}(C_i)} + p \cdot \max_{i\in\{1,\ldots,t_1\}} \mathsf{co\text{-}dim}(C_i)\right), \\ 2\left(\sum_{i=t_1+1}^{t} \mathsf{co\text{-}dim}(C_i) \cdot 2^{-\mathsf{co\text{-}dim}(C_i)} + (1-p) \cdot \max_{i\in\{t_1+1,\ldots,t\}} \mathsf{co\text{-}dim}(C_i)\right). \end{cases}$$

We remark that to truly claim the bound of "average unambiguous 1-certificate complexity" one would like to remove the additive term $p \cdot \max_{i\in\{1,\ldots,t_1\}} \mathsf{co\text{-}dim}(C_i)$ from the stated bound in the above theorem. This is because when the $\max_i \mathsf{co\text{-}dim}(C_i)$ term is *not* weighted by p, it becomes a trivial bound on entropy. Ideally, one would like to get rid of this term altogether, possibly at the expense of increasing the constant factor in the first summand.

We also note that a similar bound for the general DNF representation, i.e., when $\{C_1, \ldots, C_{t_1}\}$ is an arbitrary DNF representation of f where the C_is need not be disjoint, suffices to establish Mansour's conjecture (Conjecture 1.3). In fact, following the analogy, Theorem 3.4 implies a bound of "average 1-certificate complexity" in the general case. In this direction, we observe that a weaker bound of 1-certificate complexity, i.e., showing $\mathbb{H}(\widehat{f^2}) \leq O(\mathsf{C}_1(f))$, would already suffice to answer Mansour's conjecture positively. We refer to the full version [5] for a detailed discussion on this.

The outline for the proof of Theorem 3.4 remains the same as before, but it differs in implementation details. We sketch them now. Analogous to the proof of the main theorem we consider a partition of inputs with respect to f and its tensorized version. Motivated by the DNF representation, we study the following partition $\{C_1, \ldots, C_{t_1}, f^{-1}(1)\}$ which naturally inherits a distribution $\mathbf{C}$ given by the uniform distribution on the underlying inputs. Again we build a "small" set $\mathcal{B}$ of Fourier coefficients of f^M based on the Fourier expansions of strongly typical sequences. However, unlike before, the probability of observing a strongly typical sequence doesn't capture the number of coefficients it could contribute to $\mathcal{B}$. Here, we use stronger properties guaranteed by the strong AEP. In particular, it guarantees that the *empirical* distribution of a typical sequence is close to the distribution of $\mathbf{C}$. In contrast, the (weak) AEP only guarantees that the *empirical* entropy of a typical sequence is close to the entropy of $\mathbf{C}$. Using the stronger property we can now lower bound the magnitude of any non-zero Fourier coefficient in the Fourier expansion of the indicator function of a strongly typical sequence. We then use Parseval's Identity (Fact 2.1) to deduce an upper bound on its Fourier sparsity, which in turn is used to bound the size of $\mathcal{B}$. We also need to argue that coefficients *not* in $\mathcal{B}$ have negligible Fourier weight, which can be done as before. Using the two properties, we can now complete the proof.

3.2 New upper bounds for the FMEI conjecture

Given the hardness of obtaining better upper bounds on the Fourier entropy of a Boolean function, we make progress on a weaker conjecture, the FMEI conjecture. The FMEI conjecture is much less studied than the FEI conjecture. In fact, we are aware of only one recent paper [47] which studies the FMEI conjecture for a particular class of functions. Our second contribution is to give upper bounds on the min-entropy of general Boolean functions in terms of the minimum parity-certificate complexity (denoted $\mathsf{C}^{\oplus}_{\min}(f)$) and the

approximate spectral norm of Boolean functions (denoted $\|\widehat{f}\|_{1,\varepsilon}$). The minimum parity-certificate complexity $\mathsf{C}^{\oplus}_{\min}(f)$ is also referred to as the parity kill number by O'Donnell et al. [42] and is closely related to the communication complexity of XOR functions [56, 39, 52]. The approximate spectral norm $\|\widehat{f}\|_{1,\varepsilon}$ is related to the *quantum* communication complexity of XOR functions [34, 55]. In particular, it characterizes the bounded-error quantum communication complexity of XOR functions with constant $\mathbb{F}_2$-degree [55]. (By $\mathbb{F}_2$-degree, we mean the degree of a function when viewed as a polynomial over $\mathbb{F}_2$.)

▶ **Theorem 3.5.** *Let $f : \{-1,1\}^n \to \{-1,1\}$ be a Boolean function. Then,*

(1) *For every $\varepsilon \geq 0$, $\mathbb{H}_\infty(\hat{f}^2) \leq 2 \cdot \log\left(\|\widehat{f}\|_{1,\varepsilon}/(1-\varepsilon)\right)$.*

(2) $\mathbb{H}_\infty(\hat{f}^2) \leq 2 \cdot \mathsf{C}^{\oplus}_{\min}(f)$.

(3) $\mathbb{H}_\infty(\hat{f}^2) \leq 2(1+\log_2 3) \cdot R^{\oplus}_2(f)$.[5]

The proof of Theorem 3.5 (1) expresses the quantity $\|\widehat{f}\|_{1,\varepsilon}$ as a (minimization) linear program. We consider the dual linear program and exhibit a feasible solution that achieves an optimum of $(1-\varepsilon)/\max_S |\widehat{f}(S)|$. This proves the desired inequality. In order to prove part (2) and (3) of the theorem, the idea is to consider a "simple" function g that has "good" correlation with f, and then upper bound the correlation between f and g using Plancherel's theorem (Fact 2.1) and the fact that g has a "simple" Fourier structure. For part (2), g is chosen to be the indicator function of an (affine) subspace where f is constant, whereas for part (3) the randomized parity-decision tree computing f itself plays the role of g.[6]

As a corollary of this theorem we also obtain upper bounds on the Rényi Fourier entropy $\mathbb{H}_{1+\delta}(\hat{f}^2)$ of order $1+\delta$ for all $\delta > 0$. Recall that $\mathbb{H}_{1+\delta}(\hat{f}^2) \geq \mathbb{H}_\infty(\hat{f}^2)$ for every $\delta \geq 0$ and as $\delta \to \infty$, $\mathbb{H}_{1+\delta}(\hat{f}^2)$ converges to $\mathbb{H}_\infty(\hat{f}^2)$. Also $\mathbb{H}_1(\hat{f}^2)$ is the standard Shannon entropy of the Fourier distribution. We refer to the full version [5] for a detailed treatment of it.

We believe that these improved bounds on min-entropy of the Fourier distribution give a better understanding of Fourier coefficients of Boolean functions, and could be of independent interest. As a somewhat non-trivial application of Theorem 3.5 (in particular, part (2)) we verify the FMEI conjecture for read-k DNFs, for constant k. (A read-k DNF is a formula where each variable appears in at most k terms.)

▶ **Theorem 3.6.** *For every Boolean function $f\colon \{-1,1\}^n \to \{-1,1\}$ that can be expressed as a read-k DNF, we have*

$$\mathbb{H}_\infty(\hat{f}^2) \;\leq\; O(\log k) \cdot \mathsf{Inf}(f).$$

This theorem improves upon a recent (and independent) result of Shalev [47] that establishes the FMEI conjecture for "regular" read-k DNFs (where regular means each term in the DNF has more or less the same number of variables, see [47] for a precise definition). In order to prove Theorem 3.6, we essentially show that $\mathsf{Inf}(f)$ is at least as large as the minimum certificate complexity $\mathsf{C}_{\min}(f)$ of f.

▶ **Lemma 3.7.** *There exists a universal constant $c > 0$ such that for all $f\colon \{-1,1\}^n \to \{-1,1\}$ that can be expressed as a read-k DNF, we have*

$$\mathsf{Inf}(f) \geq c \cdot \mathsf{Var}(f) \cdot (\mathsf{C}_{\min}(f) - 1 - \log k)\,.$$

---

[5] $R^{\oplus}_2(f)$ is the randomized parity-decision tree complexity of f (we define this formally in Section 2).

[6] We remark here that there exists simpler proof of part (1), along the lines of parts (2) and (3). However, we believe that the linear-programming formulation of $\mathbb{H}_\infty(\hat{f}^2)$ might help obtain better bounds, such as fractional block sensitivity.

The proof of this lemma is an application of the KKL theorem (Theorem 2.4). Now the proof of Theorem 3.6 follows with an application of the lemma in conjunction with Theorem 3.5 (2).

3.3 Implications of the FEI conjecture and connections to the Bohnenblust-Hille inequality

Our final contribution is to understand better the structure of polynomials that ε-approximate Boolean functions on the Boolean cube. To be more specific, for simplicity we fix $\varepsilon = 1/3$ and we consider polynomials p such that $|p(x) - f(x)| \leq 1/3$ for all $x \in \{-1,1\}^n$, where f is a Boolean function. Such polynomials have proved to be powerful and found diverse applications in theoretical computer science. The single most important measure associated with such polynomials is its *degree*. The *least* degree of a polynomial that 1/3-approximates f is referred to as the *approximate degree* of f. Tight bounds on approximate degree have both algorithmic and complexity-theoretic implications, see for instance Sherstov's recent paper [50] and references therein.

In this work we ask, suppose the FEI conjecture were true, what can be said about approximating polynomials? For instance, are these approximating polynomials p sparse in their Fourier domain, i.e., is the number of monomials in p, $|\{S\colon \widehat{p}(S) \neq 0\}|$, small? Do approximating polynomials have small spectral norm (i.e., small $\sum_S |\widehat{p}(S)|$)? In order to understand these questions better, we restrict ourselves to a class of polynomials called *flat* polynomials over $\{-1,1\}$, i.e., polynomials whose non-zero coefficients have the same magnitude.

We first observe that if a flat polynomial p 1/3-approximates a Boolean function f, then the entropy of the Fourier distribution of f must be "large". In particular, we show that $\mathbb{H}(\hat{f}^2)$ must be at least as large as the logarithm of the Fourier sparsity of p.

▷ Claim 3.8. If p is a flat polynomial with sparsity T that 1/3-approximates a Boolean function f, then

$$\mathbb{H}(\hat{f}^2) = \Omega(\log T).$$

It then follows that assuming the FEI conjecture, a flat polynomial of degree d and sparsity $2^{\omega(d)}$ cannot 1/3-approximate a Boolean function. However, it is not clear to us how to obtain the same conclusion *unconditionally* (i.e., without assuming that the FEI conjecture is true) and, so we pose the following conjecture.

▶ **Conjecture 3.9.** *No flat polynomial of degree d and sparsity $2^{\omega(d)}$ can 1/3-approximate a Boolean function.*

▶ Remark 3.10. We remark that there exists degree-d flat *Boolean* functions of sparsity 2^d. One simple example on 4 bits is the function $x_1(x_2 + x_3)/2 + x_4(x_2 - x_3)/2$. By taking a $(d/2)$-fold product of this Boolean function on disjoint variables, we obtain our remark.

Since we could not solve the problem as posed above, we make progress in understanding this conjecture by further restricting ourselves to the class of *block-multilinear* polynomials. An n-variate polynomial is said to be *block-multilinear* if the input variables can be *partitioned* into disjoint blocks such that every monomial in the polynomial has *at most* one variable from each block. Such polynomials have been well-studied in functional analysis since the work of Bohnenblust and Hille [9], but more recently have found applications in quantum computing [1, 38], classical and quantum XOR games [12], and polynomial decoupling [44].

In the functional analysis literature block-multilinear polynomials are known as *multilinear forms*. In an ingenious work [9], Bohnenblust and Hille showed that for every degree-d multilinear form $p : (\mathbb{R}^n)^d \to \mathbb{R}$, we have

$$\Big(\sum_{i_1,\ldots,i_d=1}^{n} |\widehat{p}_{i_1,\ldots,i_d}|^{\frac{2d}{d+1}} \Big)^{\frac{d+1}{2d}} \leq C_d \cdot \max_{x^1,\ldots,x^d \in [-1,1]^n} |p(x^1,\ldots,x^d)|, \tag{2}$$

where C_d is a constant that depends on d. In [9], they showed that it suffices to pick C_d to be exponential in d to satisfy the equation above. For $d = 2$, Eq. (2) generalizes Littlewood's famous 4/3-inequality [36]. Eq. (2) is commonly referred to as the Bohnenblust-Hille (BH) inequality and is known to have deep applications in various fields of analysis such as operator theory, complex analysis, etc. There has been a long line of work on improving the constant C_d in the BH inequality (to mention a few [22, 21, 3, 6, 45]). The best known upper bound on C_d (we are aware of) is polynomial in d. It is also conjectured that it suffices to let C_d be a *universal* constant (independent of d) in order to satisfy Eq. (2).

In our context, using the best known bound on C_d in the BH-inequality implies that a flat block-multilinear polynomial of degree d and sparsity $2^{\omega(d \log d)}$ cannot 1/3-approximate a Boolean function. However, from the discussion before Conjecture 3.9, we know that the FEI conjecture implies the following theorem.

▶ **Theorem 3.11.** *If p is a flat block-multilinear polynomial of degree d and sparsity $2^{\omega(d)}$, then p cannot $1/8$-approximate a Boolean function.*

Moreover, the above theorem is also implied when the BH-constant C_d is assumed to be a universal constant. Our main contribution is to establish the above theorem *unconditionally*, i.e., neither assuming C_d is a universal constant nor assuming the FEI conjecture. In order to show the theorem, we show an inherent weakness of block-multilinear polynomials in approximating Boolean functions. More formally, we show the following.

▶ **Lemma 3.12.** *Let p be a block-multilinear polynomial of degree-d that $1/8$-approximates a Boolean function f. Then, $\mathsf{deg}(f) \leq d$.*

Now using the fact that Fourier entropy of f is at least as large as the logarithm of the sparsity of p (Claim 3.8), we obtain Theorem 3.11.

4 Conclusion

We gave improved upper bounds on Fourier entropy of Boolean functions in terms of average unambiguous (parity)-certificate complexity, and as a corollary verified the FEI conjecture for functions with bounded average unambiguous (parity)-certificate complexity. We established many bounds on Fourier min-entropy in terms of analytic and combinatorial measures, namely minimum certificate complexity, logarithm of the approximate spectral norm and randomized (parity)-decision tree complexity. As a corollary to this, we verified the FMEI conjecture for read-k DNFs. We also studied structural implications of the FEI conjecture on approximating polynomials. In particular, we proved that flat block-multilinear polynomials of degree d and sparsity $2^{\omega(d)}$ can not approximate Boolean functions.

We now list few open problems which we believe are structurally interesting and could lead towards proving the FEI or FMEI conjecture. Let $f : \{-1,1\}^n \to \{-1,1\}$ be a Boolean function.

(1) Does there exist a Fourier coefficient $S \subseteq [n]$ such that $|\widehat{f}(S)| \geq 2^{-O(\mathsf{deg}_{1/3}(f))}$? This would show $\mathbb{H}_\infty(\hat{f}^2) \leq O(\mathsf{deg}_{1/3}(f))$.

(2) Can we show $\mathbb{H}(\hat{f}^2) \leq O(Q(f))$? Or, $\mathbb{H}_\infty(\hat{f}^2) \leq O(Q(f))$? (where $Q(f)$ is the 1/3-error quantum query complexity of f, which Beals et al. [7] showed to be at least $\mathsf{deg}_{1/3}(f)/2$).

(3) Does there exist a universal constant $\lambda > 0$ such that $\mathbb{H}(\hat{f}^2) \leq \lambda \cdot \min\{\mathsf{C}^1(f), \mathsf{C}^0(f)\}$? This would resolve Mansour's conjecture.

In an earlier version of this manuscript we suggested that bounding the logarithm of the approximate spectral norm by $O(\mathsf{deg}_{1/3}(f))$ or $O(Q(f))$ might be an approach to answer Question (1) or (2) above. However, in a very recent work [14] it is shown that $\log(\|\widehat{f}\|_{1,\varepsilon})$ could be as large as $\Omega(Q(f) \cdot \log n)$, thus nullifying the suggested approach.

References

1 S. Aaronson and A. Ambainis. Forrelation: A problem that optimally separates quantum from classical computing. *SIAM Journal of Computing*, 47(3):982–1038, 2018. Earlier in STOC'15. `arXiv:1411.5729`.

2 A. Akavia, A. Bogdanov, S. Guo, A.Kamath, and A.Rosen. Candidate weak pseudorandom functions in $AC^0 \circ MOD2$. In *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science*, ITCS'14, pages 251–260. ACM, 2014.

3 N. Albuquerque, F. Bayart, D. Pellegrino, and J. B. Seoane-Sepúlveda. Sharp generalizations of the multilinear Bohnenblust-Hille inequality. *Journal of Functional Analysis*, 266(6):3276–3740, 2014. `arXiv:1306.3362`.

4 A. Ambainis, M. Kokainis, and R. Kothari. Nearly optimal separations between communication (or query) complexity and partitions. In *31st Conference on Computational Complexity, CCC 2016*, pages 4:1–4:14, 2016. Combines `arXiv:1512.01210` and `arXiv:1512.00661`.

5 S. Arunachalam, S. Chakraborty, M. Koucký, N. Saurabh, and R. de Wolf. Improved bounds on fourier entropy and min-entropy, 2018. `arXiv:1809.09819`.

6 F. Bayart, D. Pellegrino, and J. B. Seoane-Sepúlveda. The Bohr radius of the n-dimensional polydisk is equivalent to $\sqrt{(\log n)/n}$. *Advances in Mathematics*, 264:726–746, 2014.

7 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. Earlier version in FOCS'98. `arXiv:quant-ph/9802049`.

8 S. Ben-David, P. Hatami, and A. Tal. Low-sensitivity functions from unambiguous certificates. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017*, pages 28:1–28:23, 2017. `arXiv:1605.07084`.

9 H. F. Bohnenblust and E. Hille. On the absolute convergence of Dirichlet series. *Annals of Mathematics*, pages 600–622, 1931.

10 R. Boppana. The average sensitivity of bounded-depth circuits. *Information Processing Letters*, 63(5):257–261, 1997.

11 J. Bourgain and G. Kalai. Influences of variables and threshold intervals under group symmetries. *Geometric and Functional Analysis (GAFA)*, 7(3):438–461, 1997.

12 J. Briët, H. Buhrman, T. Lee, and T. Vidick. Multipartite entanglement in XOR games. *Quantum Information & Computation*, 13(3-4):334–360, 2013. `arXiv:0911.4007`.

13 H. Buhrman and R. de Wolf. Complexity measures and decision tree complexity: A survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

14 S. Chakraborty, A. Chattopadhyay, N. Mande, and M. Paraashar. Quantum Query-to-Communication simulation needs a logarithmic overhead, 2019. `arXiv:1909.10428`.

15 S. Chakraborty, S. Karmalkar, S. Kundu, S. V. Lokam, and N. Saurabh. Fourier entropy-influence conjecture for random linear threshold functions. In *LATIN 2018: Theoretical Informatics - 13th Latin American Symposium, 2018*, pages 275–289, 2018.

16 S. Chakraborty, R. Kulkarni, S.V. Lokam, and N. Saurabh. Upper bounds on Fourier entropy. *Theoretical Computer Science*, 654:92–112, 2016. The first version appeared as a technical report TR13-052 on ECCC in 2013.

17 M. Cheraghchi, E. Grigorescu, B. Juba, K. Wimmer, and N. Xie. $AC^0 \circ$ MOD2 lower bounds for the Boolean inner product. *Journal of Computer and System Sciences*, 97:45–59, 2018.

18 G. Cohen and I. Shinkar. The complexity of DNF of parities. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, ITCS '16, pages 47–58. ACM, 2016.

19 T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & Sons, 1991.

20 B. Das, M. Pal, and V. Visavaliya. The entropy influence conjecture revisited, 2011. `arXiv:1110.4301`.

21 A. Defant, L. Frerick, J. Ortega-Cerdá, M. Ounaïes, and K. Seip. The Bohnenblust-Hille inequality for homogeneous polynomials is hypercontractive. *Annals of Mathematics*, 174(1):485–497, 2011. `arXiv:0904.3540`.

22 A. Defant, D. Popa, and U. Schwarting. Coordinatewise multiple summing operators in banach spaces. *Journal of Functional Analysis*, 259(1):220–242, 2010.

23 E. Friedgut. Boolean functions with low average sensitivity depend on few coordinates. *Combinatorica*, 18(1):27–35, 1998.

24 E. Friedgut and G. Kalai. Every monotone graph property has a sharp threshold. *Proceedings of the American Mathematical Society*, 124(10):2993–3002, 1996.

25 M. Göös. Lower bounds for clique vs. independent set. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015*, pages 1066–1076, 2015.

26 P. Gopalan, A. T. Kalai, and A. Klivans. Agnostically learning decision trees. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, STOC '08, pages 527–536, 2008.

27 P. Gopalan, R. A. Servedio, A. Tal, and A. Wigderson. Degree and sensitivity: Tails of two distributions. In *31st Conference on Computational Complexity, CCC 2016*, pages 13:1–13:23, 2016. `arXiv:1604.07432`.

28 L. Gross. Logarithmic Sobolev inequalities. *American Journal of Mathematics*, 97(4):1061–1083, 1975.

29 R. Hod. Improved lower bounds for the Fourier entropy/influence conjecture via lexicographic functions, 2017. `arXiv:1711.00762`.

30 J. Kahn, G. Kalai, and Nathan Linial. The influence of variables on Boolean functions. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 68–80, 1988.

31 G. Kalai. The entropy/influence conjecture. Terence Tao's blog: `https://terrytao.wordpress.com/2007/08/16/gil-kalai-the-entropyinfluence-conjecture/`, 2007.

32 N. Keller, E. Mossel, and T. Schlank. A note on the entropy/influence conjecture. *Discrete Mathematics*, 312(22):3364–3372, 2012. `arXiv:1105.2651`.

33 A. Klivans, H. Lee, and A. Wan. Mansour's conjecture is true for random DNF formulas. In *Proceedings of the 23rd Conference on Learning Theory*, pages 368–380, 2010.

34 T. Lee and A. Shraibman. Lower bounds in communication complexity. *Foundations and Trends in Theoretical Computer Science*, 3(4):263–398, 2009.

35 N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, fourier transform, and learnability. *J. ACM*, 40(3):607–620, July 1993.

36 J. E. Littlewood. On bounded bilinear forms in an infinite number of variables. *The Quarterly Journal of Mathematics*, 1:164–174, 1930.

37 Y. Mansour. An $n^{O(\log \log n)}$ learning algorithm for DNF under the uniform distribution. *Journal of Computer and System Sciences*, 50(3):543–550, 1995.

38 A. Montanaro. Some applications of hypercontractive inequalities in quantum information theory. *Journal of Mathematical Physics*, 53(12):122206, 2012. `arXiv:1208.0161`.

39 A. Montanaro and T. Osborne. On the communication complexity of XOR functions, 2009. `arXiv:0909.3392`.

40 R. O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.

41 R. O'Donnell and L-Y. Tan. A composition theorem for the Fourier entropy-influence conjecture. In *Proceedings of Automata, Languages and Programming - 40th International Colloquium*, pages 780–791, 2013. arXiv:1304.1347.

42 R. O'Donnell, J. Wright, Y. Zhao, X. Sun, and L-Y. Tan. A composition theorem for parity kill number. In *IEEE 29th Conference on Computational Complexity, CCC 2014*, pages 144–154, 2014. arXiv:1312.2143.

43 R. O'Donnell, J. Wright, and Y. Zhou. The Fourier entropy-influence conjecture for certain classes of Boolean functions. In *Proceedings of Automata, Languages and Programming - 38th International Colloquium*, pages 330–341, 2011.

44 R. O'Donnell and Y. Zhao. Polynomial bounds for decoupling, with applications. In *31st Conference on Computational Complexity, CCC 2016*, pages 24:1–24:18, 2016. arXiv:1512.01603.

45 D. Pellegrino and E. V. Teixeira. Towards sharp Bohnenblust-Hille constants. *Communications in Contemporary Mathematics*, 20(3):1750029, 2018. arXiv:1604.07595.

46 R. A. Servedio and E. Viola. On a special case of rigidity. Manuscript: http://eccc.hpi-web.de/report/2012/144, 2012.

47 G. Shalev. On the Fourier Entropy Influence conjecture for extremal classes. arXiv, 2018. arXiv:1806.03646.

48 R. Shaltiel and E. Viola. Hardness amplification proofs require majority. *SIAM Journal on Computing*, 39(7):3122–3154, 2010.

49 C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

50 Alexander A. Sherstov. Algorithmic polynomials. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 311–324, 2018.

51 A. Tal. Tight bounds on the Fourier spectrum of AC^0. In *32nd Computational Complexity Conference, CCC 2017*, pages 15:1–15:31, 2017.

52 H. Y. Tsang, C. H. Wong, N. Xie, and S. Zhang. Fourier sparsity, spectral norm, and the log-rank conjecture. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013*, pages 658–667, 2013. arXiv:1304.1245.

53 A. Wan, J. Wright, and C. Wu. Decision trees, protocols and the entropy-influence conjecture. In *Innovations in Theoretical Computer Science, ITCS'14*, pages 67–80, 2014. arXiv:1312.3003.

54 R. de Wolf. A brief introduction to Fourier analysis on the Boolean cube. *Theory of Computing*, 2008. ToC Library, Graduate Surveys 1.

55 S. Zhang. Efficient quantum protocols for XOR functions. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 1878–1885, 2014. arXiv:1307.6738.

56 Z. Zhang and Y. Shi. Communication complexities of symmetric XOR functions. *Quantum Information & Computation*, 9(3):255–263, 2009.

Information Distance Revisited

Bruno Bauwens
National Research University Higher School of Economics, Moscow, Russia
https://www.hse.ru/en/org/persons/160550073
brbauwens@gmail.com

Abstract

We consider the notion of information distance between two objects x and y introduced by Bennett, Gács, Li, Vitanyi, and Zurek [2] as the minimal length of a program that computes x from y as well as computing y from x, and study different versions of this notion. In the above paper, it was shown that the prefix version of information distance equals $\max(\mathrm{K}(x|y), \mathrm{K}(y|x))$ up to additive logarithmic terms. It was claimed by Mahmud [13] that this equality holds up to additive $O(1)$-precision. We show that this claim is false, but does hold if the distance is at least logarithmic. This implies that the original definition provides a metric on strings that are at superlogarithmically separated.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Kolmogorov complexity, algorithmic information distance

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.46

Related Version https://arxiv.org/abs/1807.11087

Acknowledgements This work was initiated by Alexander (Sasha) Shen, who informed me about the error in [13] during a discussion of the paper [15]. Afterwards I explained the proof of Theorem 13 to Sasha. He simplified it, and he wrote all of the current manuscript and the proof of Theorem 13, which is available in the ArXiv version of this paper [1]. Later, I added Theorem 14. Only this proof is written by me, and is also available on ArXiv [1]. After this was added, Sasha decided that his contribution was no longer proportional, and decided he did no longer want to remain an author. I am especially grateful for his generous permission to publish this nicely written document, with minor modifications suggested by reviewers. I thank the reviewers for these suggestions. All errors in this document are solely my responsability. I thank Mikhail Andreev for the proof of Proposition 7 and many useful discussions. Finally, I thank Artem Grachev and the participants of the Kolmogorov seminar in Moscow state university for useful discussions.

1 Introduction

Informally speaking, Kolmogorov complexity measures the amount of information in an object (say, a bit string) in bits. The complexity $\mathrm{C}(x)$ of x is defined as the minimal bit length of a program that generates x. This definition depends on the programming language used, but one can fix an optimal language that makes the complexity function minimal up to an $O(1)$ additive term. In a similar way one can define the *conditional* Kolmogorov complexity $\mathrm{C}(x|y)$ of a string x given some other string y as a condition. Namely, we consider the minimal length of a program that transforms y to x. Informally speaking, $\mathrm{C}(x|y)$ is the amount of information in x that is missing in y, the number of bits that we should give in addition to y if we want to specify x.

The notion of *information distance* was introduced in [2] as "the length of a shortest binary program that computes x from y as well as computing y from x." It is clear that such a program cannot be shorter than $\mathrm{C}(x|y)$ or $\mathrm{C}(y|x)$ since it performs both tasks; on the other hand, it cannot be much longer than the sum of these two quantities (we can combine the programs that map x to y and vice versa with a small overhead needed to separate the

two parts and to distinguish x from y). As the authors of [2] note, "being shortest, such a program should take advantage of any redundancy between the information required to go from x to y and the information required to go from y to x", and the natural question arises: to what extent is this possible? The main result of [2] gives the strongest upper bound possible and says that the information distance equals $\max(\mathrm{C}(x|y), \mathrm{C}(y|x))$ with logarithmic precision. In many applications, this characterization turned out to be useful, see [11, Section 8.4]. In fact, in [2] the prefix version of complexity, denoted by $\mathrm{K}(x|y)$, and the corresponding definition of information distance were used; see, e.g. [14] for the detailed explanation of different complexity definitions. The difference between prefix and plain versions is logarithmic in the complexity, so it does not matter whether we use plain or prefix versions if we are interested in results with logarithmic precision. However, the prefix version of the above characterization has an advantage: after adding a large enough constant, this distance satisfies the triangle inequality. The plain variant does not have this property, and this follows from Proposition 7 below. However, several inequalities that are true with logarithmic precision for plain complexity, become true with $O(1)$-precision if prefix complexity is used. So one could hope that a stronger result with $O(1)$-precision holds for prefix complexity. If this is true, then also the original definition satisfies the triangle inequality (after a constant increase). In [2, Section VII], this was conjectured to be false, and in [13] it was claimed to be true; in [12] a similar claim is made with reference to [2].[1] Unfortunately, the proof in [13] contains an error, and we show that the result is not valid for prefix complexity with $O(1)$-precision. On the other hand, it is easy to see that the original argument from [2] can be adapted for plain complexity to obtain the result with $O(1)$-precision, as noted in [15].

In this paper we try to clarify the situation and discuss the possible definitions of information distance in plain and prefix versions, and their subtle points (one of these subtle points was the source of the error in [13]). We also discuss some related notions. In Section 2 we consider the easier case of plain complexity; then in Section 3 we discuss the different definitions of prefix complexity (with prefix-free and prefix-stable machines, as well as definitions using the a priori probability) and in Section 4 we discuss their counterparts for the information distance. In Section 5 we use the game approach to show that indeed the relation between information distance (in the prefix version) and conditional prefix complexity is *not* valid with $O(1)$-precision, contrary to what is said in [13]. Finally, we show that if the information distance is at least logarithmic, then equality holds.

2 Plain complexity and information distance

Let us recall the definition of plain conditional Kolmogorov complexity. Let $U(p, x)$ be a computable partial function of two string arguments; its values are also binary strings. We may think of U as an interpreter of some programming language. The first argument p is considered as a program and the second argument is an input for this program. Then we define the complexity function

$$\mathrm{C}_U(y|x) = \min\{|p| \colon U(p, x) = y\};$$

[1] The authors of [12] define (section 2.2) the function $E(x, y)$ as the prefix-free non-bipartite version of the information distance (see the discussion below in section 4.1) and then write: "the following theorem proved in [4] was a surprise: Theorem 1. $E(x, y) = \max\{\mathrm{C}(x|y), \mathrm{C}(y|x)\}$". They do not mention that in the paper they cited as [4] (it is [2] in our list) there is a logarithmic error term; in fact, they do not mention any error terms (though in other statements the constant term is written explicitly). Probably this is a typo, since more general Theorem 2 in [12] does contain a logarithmic error term.

here $|p|$ stands for the length of a binary string p, so the right hand side is the minimal length of a program that produces output y given input x. The classical Solomonoff–Kolmogorov theorem says that there exists an optimal U that makes C_U minimal up to an $O(1)$-additive term. We fix some optimal U and then denote C_U by just C. See, e.g., [11, 14] for the details.

Now we want to define the information distance between x and y. One can try the following approach: take some optimal U from the definition of conditional complexity and then define

$$E_U(x,y) = \min\{|p| \colon U(p,x) = y \text{ and } U(p,y) = x\},$$

i.e., consider the minimal length of a program that both maps x to y and y to x. However, there is a caveat, as the following simple observation shows.

▶ **Proposition 1.** *There exists some computable partial function U that makes C_U minimal up to an $O(1)$ additive term, and still $E_U(x,y)$ is infinite for some strings x and y and therefore not minimal.*

Proof. Consider an optimal function U and then define U' such that $U(\Lambda, x) = \Lambda$ where Λ is the empty string, $U'(0p,x) = 0U(p,x)$ and $U'(1p,x) = 1U(p,x)$. In other terms, U' copies the first bit of the program to the output and then applies U to the rest of the program and the input. It is easy to see that $\mathrm{C}_{U'}$ is minimal up to an $O(1)$ additive term, but $U'(q,\cdot)$ has the same first bit as q, so if x and y have different first bits, there is no q such that $U(q,x) = y$ and $U(q,y) = x$ at the same time. ◀

On the other hand, the following proposition is true (and can be proven in the same way as the existence of the optimal U for conditional complexity):

▶ **Proposition 2.** *There exists a computable partial function U that makes E_U minimal up to $O(1)$ additive term.*

Now we may define information distance for plain complexity as the minimal function E_U. Some examples. If x has small complexity, then $E_U(x,y) = \mathrm{C}(y) + O(1)$. Let x and y be n-bit strings, and let $\oplus$ denote the bitwise xor-operation. We have $E_U(x,y) \leqslant n + O(1)$, because $x \oplus (x \oplus y) = y$ and $y \oplus (y \oplus x) = x$. It turns out that the original argument from [2] can be easily adapted to show the following result (that is a special case of a more general result about several strings proven in [15]):

▶ **Theorem 3.** *The minimal function E_U equals* $\max(\mathrm{C}(x|y), \mathrm{C}(y|x)) + O(1)$.

Proof. We provide the adapted proof here for the reader's convenience. In one direction we have to prove that $\mathrm{C}(x|y) \leqslant E_U(x,y) + O(1)$, and the same for $\mathrm{C}(y|x)$. This is obvious, since the definition of E_U contains more requirements for p: it should map both x to y and y to x, while in $\mathrm{C}(x|y)$ it is enough to map y to x.

To prove the reverse inequality, consider for each n the binary relation R_n on strings (of all lengths) defined as

$$R_n(x,y) \quad \Longleftrightarrow \quad \mathrm{C}(x|y) < n \text{ and } \mathrm{C}(y|x) < n.$$

By definition, this relation is symmetric. It is easy to see that R_n is (computably) enumerable uniformly in n, since we may compute better and better upper bounds for C reaching ultimately its true value. We think of R_n as the set of edges of an undirected graph whose vertices are binary strings. Note that each vertex x of this graph has degree less than 2^n since there are less than 2^n programs of length less than n that map x to its neighbors.

For each n, we enumerate edges of this graph (i.e., pairs in R_n). We want to assign colors to the edges of R_n in such a way that edges that have a common endpoint have different colors. In other terms, we require that for every vertex x all edges of R_n adjacent to x have different colors. For that, 2^{n+1} colors are enough. Indeed, each new edge needs a color that differentiates it from less than 2^n existing edges adjacent to one its endpoint and less than 2^n edges adjacent to other endpoint.

Let us agree to use $(n+1)$-bit strings as colors for edges in R_n, and perform this coloring in parallel for all n. Now we define $U(p,x)$ for a $(n+1)$-bit string p and arbitrary string x as the string y such that the edge (x,y) has color p in the coloring of edges from R_n. Note that n can be reconstructed as $|p|-1$. The uniqueness property for colors guarantees that there is at most one y such that (x,y) has color p, so $U(p,x)$ is well defined. It is easy to see now that if $\mathrm{C}(x|y) < n$ and $\mathrm{C}(y|x) < n$, and p is the color of the edge (x,y), then $U(p,x) = y$ and $U(p,y) = x$ at the same time. This implies the reverse inequality (the $O(1)$ terms appears when we compare our U with the optimal one). ◀

▶ Remark 4. In the definition of information distance given above we look for a program p that transforms x to y and also transforms y to x. Note that we *do not tell the program which of the two transformations is requested.* A weaker definition would provide also this information to p. This modification can be done in several ways. For example, we may require in the definition of E that $U(p,0x) = y$ and $U(p,1y) = x$, using the first input bit as the direction flag. An equivalent approach is to use two computable functions U and U' in the definition and require that $U(p,x) = y$ and $U'(p,y) = x$. This corresponds to using different interpreters for both directions.

It is easy to show that the optimal functions U and U' exist for this two-interpreter version of the definition. A priori we may get a smaller value of information distance in this way, because the program's task is easier when the direction is known, informally speaking. But it is not the case for the following simple reason. Obviously, this new quantity is still an upper bound for both conditional complexities $\mathrm{C}(x|y)$ and $\mathrm{C}(y|x)$ with $O(1)$ precision. Therefore Theorem 3 guarantees that this new definition of information distance coincides with the old one up to $O(1)$ additive terms. For the prefix versions of information distance such a simple argument does not work anymore, see below.

We have seen that different approaches lead to the same notion of plain information distance (up to $O(1)$ additive term). There is also a simple and natural quantitative characterization of this notion as a minimal function in a class of functions.

▶ **Theorem 5.** *Consider the class of functions E that are non-negative, symmetric, upper semicomputable, and for some c, all n and all x, satisfy*

$$\#\{y\colon E(x,y) < n\} \leqslant c2^n. \qquad (*)$$

For every optimal U this class contains E_U, and for any E in this class, we have $E_U \leqslant E + O(1)$.

Recall that upper semicomputability of E means that one can compute a sequence of total upper bounds for E that converges to E. The equivalent requirement: the set of triples (x,y,n) where x,y are strings and n are natural numbers, such that $E(x,y) < n$, is (computably) enumerable.

Proof. The function $\max(\mathrm{C}(x|y), \mathrm{C}(y|x))$ is upper semicomputable and symmetric. The inequality $(*)$ is true for it since it is true for the smaller function $\mathrm{C}(y|x)$ (for $c = 1$; indeed, the number of programs of length less than n is at most 2^n).

On the other hand, if E is some symmetric upper semicomputable function that satisfies $(*)$, then one can for any given x and n enumerate all y such that $E(x,y) < n$. There are less than $c2^n$ strings y with this property, so given x, each such y can be described by a string of $n + \lceil \log c \rceil$ bits, its ordinal number in the enumeration. Note that the value of n can be reconstructed from this string by decreasing its length by $\lceil \log c \rceil$, so $\mathrm{C}(y|x) \leqslant n + O(1)$ if $E(x,y) < n$. It remains to apply the symmetry of E and Theorem 3. ◀

▶ Remark 6. The name "information distance" motivates the following question: does the plain information distance satisfy the triangle inequality? With logarithmic precision the answer is positive, because

$$\mathrm{C}(x|z) \leqslant \mathrm{C}(x|y) + \mathrm{C}(y|z) + O(\log(\mathrm{C}(x|y) + \mathrm{C}(y|z))).$$

However, if we replace the last term by an $O(1)$-term, then the triangle inequality is no more true. Indeed, for all strings x and y, the distance between the empty string Λ and x is $\mathrm{C}(x) + O(1)$, and the distance between x and some encoding of a pair (x,y) is at most $\mathrm{C}(y) + O(1)$, and the triangle inequality for distances with $O(1)$-precision would imply $\mathrm{C}(x,y) \leqslant \mathrm{C}(x) + \mathrm{C}(y) + O(1)$, and this is not true, see, e.g., [14, section 2.1].

One may ask whether a weaker statement saying that there is a maximal (up to an $O(1)$ additive term) function in the class of all symmetric non-negative functions E that satisfy both the condition $(*)$ and the triangle inequality, is true. The answer is negative, as the following proposition shows.

▶ **Proposition 7.** *There are two upper semicomputable symmetric functions E_1, E_2 that both satisfy the condition $(*)$ and the triangle inequality, such that no function that is bounded both by E_1 and E_2 can satisfy $(*)$ and the triangle inequality at the same time.*

Proof. Let us agree that $E_1(x,y)$ and $E_2(x,y)$ are infinite when x and y have different lengths. If x and y are n-bit strings, then $E_1(x,y) \leqslant k$ means that all the bits in x and y outside the first k positions are the same, and $E_2(x,y) \leqslant k$ is defined in a symmetric way for the last k positions. Both E_1 and E_2 satisfy the triangle inequality (and even the ultrametric inequality) and also satisfy the condition $(*)$, since the ball of radius k consist of strings that coincide except for the first/last k bits. If E is bounded both by $E_1 + O(1)$ and $E_2 + O(1)$ and satisfies the triangle inequality, then by changing the first k and the last l positions in a string x we get a string y such that $E(x,y) \leqslant k + l + O(1)$, and it is easy to see that the number of strings that can be obtained in this way is not $O(2^{k+l})$, but $\Theta((k+l)2^{k+l})$. ◀

3 Prefix complexity: different definitions

The notion of prefix complexity was introduced independently by Levin [6, 8, 4] and later by Chaitin [3]. There are several versions of this definition, and they all turn out to be equivalent, so people usually do not care much about technical details that are different. However, if we want to consider the counterparts of these definitions for information distance, their differences become important if we are interested in $O(1)$-precision.

Essentially there are four different definitions of prefix complexity that appear in the literature.

3.1 Prefix-free definition

A computable partial function $U(p, x)$ with two string arguments and string values is called *prefix-free* (with respect to the first argument) if $U(p, x)$ and $U(p', x)$ cannot be defined simultaneously for a string p and its prefix p' *and for the same second argument* x. In other words, for every string x the set of strings p such that $U(p, x)$ is defined is prefix-free, i.e., does not contain a string and its prefix at the same time.

For a prefix-free function U we may consider the complexity function $\mathrm{C}_U(y|x)$. In this way we get a smaller class of complexity functions compared with the definition of plain complexity, and the Solomonoff–Kolmogorov theorem can be easily modified to show that there exists a minimal complexity function in this smaller class (up to $O(1)$ additive term, as usual). This function is called *prefix conditional complexity* and usually is denoted by $\mathrm{K}(y|x)$. It is greater than $\mathrm{C}(y|x)$ since the class of available functions U is more restricted; the relation between C and K is well studied, see, e.g., [14, chapter 4] and references within.

The unconditional prefix complexity $\mathrm{K}(x)$ is defined in the same way, with U that does not have a second argument. We can also define $\mathrm{K}(x)$ as $\mathrm{K}(x|y_0)$ for some fixed string y_0. This string may be chosen arbitrarily; for each choice we have $\mathrm{K}(x) = \mathrm{K}(x|y_0) + O(1)$ but the constant in the $O(1)$ bound depends on the choice of y_0.

3.2 Prefix-stable definition

The prefix-stable version of the definition considers another restriction on the function U. Namely, in this version the function U should be *prefix-stable* with respect to the first argument. This means that if $U(p, x)$ is defined, then $U(p', x)$ is defined and equal to $U(p, x)$ for all p' that are extensions of p (i.e., when p is a prefix of p'). We consider the class of all computable partial prefix-stable functions U and corresponding functions C_U, and observe that there exists an optimal prefix-stable function U that makes C_U minimal in this class.

It is rather easy to see that the prefix-stable definition leads to a version of complexity that does not exceed the prefix-free one, since each prefix-free computable function can be easily extended to a prefix-stable one. The reverse inequality is not so obvious and there is no known direct proof; the standard argument compares both versions with the forth definition of prefix complexity, (the logarithm of a maximal semimeasure, see Section 3.4 below).

Prefix-free and prefix-stable definitions correspond to the same intuitive idea: the program should be "self-delimiting". This means that the machine gets access to an infinite sequence of bits that starts with the program and has no marker indicating the end of a program. The prefix-free and prefix-stable definitions correspond to two possible ways of accessing this sequence. The prefix-free definition corresponds to a blocking read primitive (if the machine needs one more input bit, the computation waits until this bit is provided). The prefix-stable definition corresponds to a non-blocking read primitive (the machine has access to the input bits queue and may continue computations if the queue is currently empty). We do not go into details here; the interested reader could find this discussion in [14, section 4.4].

3.3 A priori probability definition

In this approach we consider the *a priori probability* of y given x, the probability of the event "a random program maps x to y". More precisely, consider a prefix-stable function $U(p, x)$ and an infinite sequence π of independent uniformly distributed random bits (a random variable). We say that $U(\pi, x) = y$ if $U(p, x) = y$ for some p that is a prefix of π. Since U is prefix-stable, the value $U(\pi, x)$ is well defined. For given x and y, we denote by $m_U(y|x)$ the probability of this event (the measure of the set of π such that $U(\pi, x) = y$). For each

prefix-stable U we get some function m_U. It is easy to see that there exists an optimal U that makes m_U maximal (up to an $O(1)$-factor). Then we define prefix complexity $\mathrm{K}(y|x)$ as $-\log m_U(y|x)$ for this optimal U, where the logarithm has base 2.

It is also easy to see that if we use prefix-free functions U instead of prefix-stable ones, we obtain the same definition of prefix complexity. Informally speaking, if we have an infinite sequence of random bits as the first argument, we do not care whether we have blocking or non-blocking read access, the bits are always there. The non-trivial and most fundamental result about prefix complexity is that this definition, as the logarithm of the probability, is equivalent to the two previous ones. As a byproduct of this result we see that the prefix-free and prefix-stable definitions are equivalent. This proof and the detailed discussion of the difference between the definitions can be found, e.g., in [14, chapter 4].

3.4 Semimeasure definition

The semimeasure approach defines a priori probability in a different way, as a convergent series that converges as slow as possible. More precisely, a *lower semicomputable semimeasure* is a non-negative real-valued function $m(x)$ on binary strings such that $m(x)$ is a limit of an increasing sequence of rational numbers and $\sum_x m(x) \leqslant 1$ that is computable uniformly in x. There exists a lower semicomputable semimeasure $\mathsf{m}(x)$ that is maximal up to $O(1)$-factors, and its negative logarithm coincides with unconditional prefix complexity $\mathrm{K}(x)$ up to an $O(1)$ additive term.

We can define conditional prefix complexity in the same way, considering semimeasures with parameter y. Namely, we consider lower semicomputable non-negative real-valued functions $m(x,y)$ such that $\sum_x m(x,y) \leqslant 1$ for every y. Again there exists a maximal function among them, denoted by $\mathsf{m}(x|y)$, and its negative logarithm equals $\mathrm{K}(x|y)$ up to an $O(1)$ additive term.

To prove this equality, we note first that the a priori conditional probability $m_U(x|y)$ is a lower semicomputable conditional semimeasure. The lower semicomputability is easy to see: we can simulate the machine U and discover more and more programs that map y to x. The inequality $\sum_x m_U(x|y)$ also has a simple probabilistic meaning: the events "π maps y to x" for a given y and different x are disjoint, so the sum of their probabilities does not exceed 1. The other direction (starting from a semimeasure, construct a machine) is a bit more difficult, but in fact it is possible (even exactly, without additional $O(1)$-factors). See [14, chapter 4] for details.

The semimeasure definition can be reformulated in terms of complexities (by taking exponents): $\mathrm{K}(x|y)$ is a minimal (up to $O(1)$ additive term) upper semicomputable non-negative integer function $k(x,y)$ such that

$$\sum_x 2^{-k(x,y)} \leqslant 1$$

for all y. A similar characterization of plain complexity would use a weaker requirement

$$\#\{x\colon k(x,y) < n\} < c2^n$$

for some c and all y. (We discussed a similar result for information distance where the additional symmetry requirement was used, but the proof is the same.)

3.5 Warning

There exists a definition of plain conditional complexity that does *not* have a prefix-version counterpart. Namely, the plain conditional complexity $\mathrm{C}(x|y)$ can be equivalently defined as the *minimal unconditional plain complexity of a program that maps y to x*. In this way we

do not need the programming language used to map y to x to be optimal; it is enough to assume that we can computably translate programs in other languages into our language; this property, sometimes called *s-m-n-theorem* or *Gödel property of a computable numbering*, is true for almost all reasonable programming languages. Of course, we still assume that the language used in the definition of unconditional Kolmogorov complexity is optimal.

One may hope that $\mathrm{K}(x|y)$ can be similarly defined as the minimal unconditional prefix complexity of a program that maps y to x. The following proposition shows that it is not the case.

▶ **Proposition 8.** *The prefix complexity $\mathrm{K}(x|y)$ does not exceed the minimal prefix complexity of a program that maps y to x; however, the difference between these two quantities is not bounded.*

Proof. To prove the first part, assume that $U_1(p)$ is a prefix-stable function of one argument that makes the complexity function

$$\mathrm{C}_{U_1}(q) = \min\{|p|\colon U(p) = q\}$$

minimal. Then $\mathrm{C}_U(q) = \mathrm{K}(q) + O(1)$. (We still need an $O(1)$ term since the choice of an optimal prefix-stable function is arbitrary). Then consider the function

$$U_2(p, x) = [U_1(p)](x)$$

where $[q](x)$ denotes the output of a program q on input x. Then U_2 is a prefix-stable function from the definition of conditional prefix complexity, and

$$\mathrm{C}_{U_2}(y|x) \leqslant \mathrm{C}_{U_1}(q)$$

for any program q that maps x to y (i.e., $[q](x) = y$). This gives the inequality mentioned in the proposition. Now we have to show that this inequality is not an equality with $O(1)$-precision.

Note that $\mathrm{K}(x|n) \leqslant n + O(1)$ for every binary string x of length n. Indeed, a prefix-stable (or prefix-free) machine that gets n as input can copy n first bits of its program to the output. (The prefix-free machine should check that there are exactly n input bits.) In this way we get n-bit programs for all strings of length n.

Now assume that the two quantities coincide up to an $O(1)$ additive term. Then for every string x there exists a program q_x that maps $|x|$ to x and $\mathrm{K}(q_x) \leqslant |x| + c$ for all x and some c. Note that q_x may be equal to q_y for $x \neq y$, but this may happen only if x and y have different lengths. Consider now the set Q of all q_x for all strings x, and the series

$$\sum_{q \in Q} 2^{-\mathrm{K}(q)}. \qquad (**)$$

This sum does not exceed 1 (it is a part of a similar sum for all q that is at most 1, see above). On the other hand, we have at least 2^n different programs q_x for all n-bit strings x, and they correspond to different terms in $(**)$; each of these terms is at least 2^{-n-c}. We get a converging series that contains, for every n, at least 2^n terms of size at least 2^{-n-c}. It is easy to see that such a series does not exist. Indeed, each tail of this series should be at least 2^{-c-1} (consider these 2^n terms for large n when at least half of these terms are in the tail), and this is incompatible with convergence. ◀

Why do we get a bigger quantity when considering the prefix complexity of a program that maps y to x? The reason is that the prefix-freeness (or prefix-stability) requirement for the function $U(p, x)$ is formulated separately for each x: the decision where to stop reading

the program p *may depend on its input* x. This is not possible for a prefix-free description of a program that maps x to y. It is easy to overlook this problem when we informally describe prefix complexity $\mathrm{K}(x|y)$ as "the minimal length of a program, written in a self-delimiting language, that maps y to x", because the words "self-delimiting language" implicitly assume that we can determine where the program ends while reading the program text (and before we know its input), and this is a wrong assumption.

3.6 Historical digression

Let us comment a bit on the history of prefix complexity. It appeared first in 1971 in Levin's PhD thesis [6]; Kolmogorov was his thesis advisor. Levin used essentially the semimeasure definition (formulated a bit differently). This thesis was in Russian and remained unpublished for a very long time. In 1974 Gács' paper [4] appeared where the formula for the prefix complexity of a pair was proven. This paper mentioned prefix complexity as "introduced by Levin in [4], [5]" ([7] and [8] in our numbering). The first of these two papers does not say anything about prefix complexity explicitly, but defines the monotone complexity of sequences of natural numbers, and prefix complexity can be considered as a special case when the sequence has length 1 (this is equivalent to the prefix-stable definition of prefix complexity). The second paper has a comment "(to appear)" in Gács' paper. We discuss it later in this section.

Gács does not reproduce the definition of prefix complexity, saying only that it is "defined as the complexity of specifying x on a machine on which it is impossible to indicate the endpoint[2] of a master program: an infinite sequence of binary symbols enters the machine and the machine must itself decide how many binary symbols are required for its computation". This description is not completely clear, but it looks more like a prefix-free definition if we understand it in such a way that the program is written on a one-directional tape and the machine decides where to stop reading. Gács also notes that prefix complexity (he denotes it by $KP(x)$) "is equal to the [negative] base two logarithm of a universal semicomputable probability measure that can be defined on the countable set of all words".

Levin's 1974 paper [8] says that "the quantity $KP(x)$ has been investigated in details in [6,7]". Here [7] in Levin's numbering is Gács paper cited above ([4] is our numbering) and has the comment "in press", and [6] in Levin's numbering is cited as [Levin L.A., On different version of algorithmic complexity of finite objects, to appear]. Levin does not have a paper with exactly this title, but the closest approximation is his 1976 paper [9], where prefix complexity is defined as the logarithm of a maximal semimeasure. Except for these references, [8] describes the prefix complexity in terms of prefix-stable functions: "It differs from the Kolmogorov complexity measure $\langle\ldots\rangle$ in that the decoding algorithm A has the following "prefix" attribute: if $A(p_1)$ and $A(p_2)$ are defined and distinct, then p_1 cannot be a beginning fragment of p_2".

The prefix-free and a priori probability definitions were given independently by Chaitin in [3] (in different notation) together with the proof of their equivalence, so [3] was the first publication containing this (important) proof.

Now it seems that the most popular definition of prefix complexity is the prefix-free one, for example, it is given as the main definition in [11].

[2] The English translation says "halting" instead of "endpoint" but this is an obvious translation error.

4 Prefix complexity and information distance

4.1 Four versions of prefix information distance

Both the prefix-free and prefix-stable versions of prefix complexity have their counterparts for the information distance.

Let $U(p, x)$ be a partial computable prefix-free [respectively, prefix-stable] function of two string arguments having string values. Consider the function

$$E_U(x, y) = \min\{|p| \colon U(p, x) = y \text{ and } U(p, y) = x\}.$$

As before, one can easily prove that there exists a minimal (up to $O(1)$) function among all functions E_U of the class considered. It will be called *prefix-free* [respectively *prefix-stable*] *information distance* function. We clarify the difference between these variants.

Note that only the cases when $U(p, x) = y$ and also $U(p, y) = x$ matter for E_U. So we may assume without loss of generality that $U(p, x) = y \Leftrightarrow U(p, y) = x$ waiting until both equalities are true before finalizing the values of U. Then for every p we have some matching M_p on the set of all strings: an edge x–y is in M_p if $U(p, x) = y$ and $U(p, y) = x$. This is indeed a matching: for every x only $U(p, x)$ may be connected with x.

The set M_p is enumerable uniformly in p. In the prefix-free version the matchings M_p and M_q are disjoint (have no common vertices) for two compatible strings p and q (one is an extension of the other). For the prefix-stable version M_p increases when p increases (and remains a matching). It is easy to see that a family M_p that has these properties, always corresponds to some function U, and this statement holds both in the prefix-free and prefix-stable version.

There is another way in which this definition could be modified. As we have discussed for plain complexity, we may consider two different functions U and U' and consider the distance function

$$E_{U,U'}(x, y) = \min\{|p| \colon U(p, x) = y \text{ and } U'(p, y) = x\}.$$

Intuitively this means that we know the transformation direction in addition to the input string. This corresponds to matchings in a bipartite graph where both parts consist of all binary strings; the edge x–y is in the matching M_p if $U(p, x) = y$ and $U'(p, y) = x$. Again instead of the pair (U, U') we may consider the family of matchings that are disjoint (for compatible p, in the prefix-free version) or monotone (for the prefix-stable version). In this way we get two other versions of information distance that could be called *bipartite prefix-free* and *bipartite prefix-stable* information distances.

In [2] the information distance is defined as the prefix-free information distance with the same function U for both directions, not two different ones. The definition in section III considers the minimal function among all E_U. This minimal function is denoted by $E_0(x, y)$, while $\max(\mathrm{K}(x|y), \mathrm{K}(y|x))$ is denoted by $E_1(x, y)$, see section I of the same paper. The inequality $E_1 \leqslant E_0$ is obvious, and the reverse inequality with logarithmic precision is proven in [2] as Theorem 3.3.

Which of the four versions of prefix information distance is the most natural? Are they really different? It is easy to see that the prefix-stable version (bipartite or not) does not exceed the corresponding prefix-free version, since every prefix-free function has a prefix-stable extension. Also each bipartite version (prefix-free or prefix-stable) does not exceed the corresponding non-bipartite version for obvious reasons: one may take $U = U'$. It is hard to say which version is most natural, and the question whether some of them coincide or

all four are different, remains open. But as we will see in Theorem 13, the smallest of all four, the prefix-stable bipartite version, is still bigger than E_1 (the maximum of conditional complexities), and the difference is unbounded. Hence, for all four versions, including the prefix-free non-bipartite version used both in [2, 12, 13], the equality with $O(1)$-precision is not true, contrary to what is said in [13]. This was conjectured in [2, Section VII].

However, before going to this negative result, we prove some positive results about the definition of information distance that is a counterpart of the a priori probability definition of prefix complexity.

4.2 A priori probability of going back and forth

Fix some prefix-free function $U(p,x)$. The conditional a priori probability $m_U(y|x)$ is defined as

$$\Pr_{\pi}[U(\pi,x)=y],$$

where π is an infinite uniformly randomnly generated bitsequence, and $U(\pi,x)=y$ means that $U(p,x)=y$ for some p that is a prefix of π. As we discussed, there exists a maximal function among all m_U, and its negative logarithm equals the conditional prefix complexity $\mathrm{K}(y|x)$.

Now let us consider the counterpart of this construction for the information distance. The natural way to do this is to consider the function

$$e_U(x,y)=\Pr_{\pi}[U(\pi,x)=y \text{ and } U(\pi,y)=x].$$

Note that in this definition the prefixes of π used for both computations are not necessarily the same. It is easy to show, as usual, that there exists an *optimal* machine U that makes e_U maximal. Fixing some optimal U, we get some function $\mathsf{e}(x,y)$. Note that different optimal U lead to functions that differ only by $O(1)$-factor. The negative logarithm of this function coincides with E_1 from [2] with $O(1)$-precision, as the following result says.

▶ **Theorem 9.**

$$-\log \mathsf{e}(x,y)=\max(\mathrm{K}(x|y),\mathrm{K}(y|x))+O(1).$$

Proof. Rewriting the right-hand side in the exponential scale, we need to prove that

$$\mathsf{e}(x,y)=\min(\mathsf{m}(x|y),\mathsf{m}(y|x))$$

up to $O(1)$-factors. One direction is obvious: $\mathsf{e}(x,y)$ is smaller than $\mathsf{m}(x|y)$ since the set of π in the definition of e is a subset of the corresponding set for m, if we use the probabilistic definition of $\mathsf{m}=m_U$. The same is true for $\mathsf{m}(y|x)$.

The non-trivial part of the statement is the reverse inequality. Here we need to construct a machine U such that

$$e_U(x,y)\geqslant \min(\mathsf{m}(x|y),\mathsf{m}(y|x))$$

up to $O(1)$-factors.

Let us denote the right-hand side by $u(x,y)$. The function u is symmetric, lower semicomputable and $\sum_y u(x,y)\leqslant 1$ for all x (due to the symmetry, we do not need the other inequality where y is fixed). This is all we need to construct U with the desired properties; in fact $e_U(x,y)$ will be at least $0.5u(x,y)$, (and the factor 0.5 is important for the proof).

Every machine U has a "dual" representation: for every pair (x, y) one may consider the subset $U_{x,y}$ of the Cantor space that consists of all π such that $U(\pi, x) = y$ and $U(\pi, y) = x$. These sets are effectively open (i.e., are computably enumerable unions of intervals in the Cantor space) uniformly in x, y, are symmetric ($U_{x,y} = U_{y,x}$) and have the following property: for a fixed x, all sets $U_{x,y}$ for all y (including $y = x$) are disjoint.

What is important to us is that this correspondence works in both directions. If we have some family $U_{x,y}$ of uniformly effectively open sets that is symmetric and has the disjointness property mentioned above, there exists a prefix-free machine U that generates these sets as described above. This machine works as follows: given some x, it enumerates the intervals that form $U_{x,y}$ for all y (it is possible since the sets $U_{x,y}$ are effectively open uniformly in x, y). One may assume without loss of generality that all the intervals in the enumeration are disjoint. Indeed, every effectively open set can be represented as a union of a computable sequence of disjoint intervals (to make intervals disjoint, we represent the set difference between the last interval and previously generated intervals as a a finite union of intervals). Note also that for different values of y the sets $U_{x,y}$ are disjoint by the assumption. If the enumeration for $U_{x,y}$ contains the interval $[p]$ (the set of all extensions of some bit string p), then we let $U(p, x) = y$ and $U(p, y) = x$ (we assume that the same enumeration is used for $U_{x,y}$ and $U_{y,x}$). Since all intervals are disjoint, the function $U(p, x)$ is prefix-free.

Now it remains (and this is the main part of the proof) to construct the family $U_{x,y}$ with the required properties in such a way that the measure of $U_{x,y}$ is at least $0.5u(x, y)$. In our construction it will be *exactly* $0.5u(x, y)$. For that we use the same idea as in [2] but in the continuous setting. Since $u(x, y)$ is lower semicomputable, we may consider the increasing sequence $u'(x, y)$ of approximations from below (that increase with time, though we do not explicitly mention time in the notation) that converge to $u(x, y)$. We assume that at each step one of the values $u'(x, y)$ increases by a dyadic rational number r. In response to that increase, we add to $U_{x,y}$ one or several intervals that have total measure $r/2$ and do not intersect $U_{x,z}$ and $U_{z,y}$ for any z. For that we consider the unions of all already chosen parts of $U_{x,z}$ and of all chosen parts of $U_{z,y}$. The measure of the first union is bounded by $0.5 \sum_z u'(x, z)$ and the measure of the second union is bounded by $0.5 \sum_z u'(z, y)$ where u' is the lower bound for u before the r-increase. Since the sums remain bounded by 1 after the r-increase, we may select a subset of measure $r/2$ outside both unions. (We may even select a subset of measure r, but this will destroy the construction at the following steps, so we add only $r/2$ to $U_{x,y}$.) ◀

▶ Remark 10. As for the other settings, we may consider two functions U and U' and the probability of the event

$$e_{U,U'}(x, y) = \Pr_{\pi}[U(\pi, x) = y \text{ and } U'(\pi, y) = x]$$

for those U, U' that make this probability maximal. The equality of Theorem 9 remains valid for this version. Indeed, the easy part can be proven in the same way, and for the difficult direction we have proven a stronger statement with additional requirement $U = U'$.

One can also describe the function e as a maximal function in some class, therefore getting a quantitative definition of E_0. This is essentially the statement of theorem 4.2 in [2]. In terms of semimeasures it can be reformulated as follows.

▶ **Proposition 11.** *Consider the class of symmetric lower semicomputable functions $u(x, y)$ with string arguments and non-negative real values such that $\sum_y u(x, y) \leqslant 1$ for all x. This class has a maximal function that coincides with $\min(\mathsf{m}(x|y), \mathsf{m}(y|x))$ up to an $O(1)$ factor.*

Indeed, we have already seen that this minimum has the required properties; if some other function $u(x,y)$ in this class is given, we compare it with conditional semimeasures $\mathsf{m}(x|y)$ and $\mathsf{m}(y|x)$ and conclude that u does not exceed both of them.

In logarithmic scale this statement can be reformulated as follows: *the class of upper semicomputable symmetric functions $D(x,y)$ with string arguments and real values such that $\sum_y 2^{-D(x,y)} \leqslant 1$ for each x, has a minimal element that coincides with* $\max(\mathrm{K}(x|y),\mathrm{K}(y|x))$ *up to an $O(1)$ additive term.* Theorem 4.2 in [2] says the same with the additional condition for D: it should satisfy the triangle inequality. This restriction makes the class smaller and could increase the minimal element in the class, but this does not happen since the function

$$\max(\mathrm{K}(x|y),\mathrm{K}(y|x))+c$$

satisfies the triangle inequality for large enough c. This follows from the inequality $\mathrm{K}(x|z) \leqslant \mathrm{K}(x|y)+\mathrm{K}(y|z)+O(1)$ since the left hand size increases by c and the right hand size increases by $2c$ when K is increased by c.

▶ Remark 12. To be pedantic, we have to note that in [2] an additional condition $D(x,x)=0$ is required for the functions in the class; to make this possible, one has to exclude the term $2^{-D(x,x)}$ in the sum (now this term equals 1) and require that $\sum_{y\neq x} 2^{-D(x,y)} \leqslant 1$ (p. 1414, the last inequality). Note that the triangle inequality remains valid if we change D and let $D(x,x)=0$ for all x.

5 A counterexample

In this section we present the main negative (and most technically difficult) result of this paper that shows that none of the four prefix distances coincides with

$$E_1(x,y)=\max(\mathrm{K}(x|y),\mathrm{K}(y|x)).$$

▶ **Theorem 13.** *The bipartite prefix-stable information distance exceeds $E_1(x,y)$ more than by a constant: the difference is unbounded.*

As we have mentioned, the other three versions of the information distance are even bigger, so the same result is true for all of them. We will explain the proof for the non-bipartite prefix-stable version (it is a bit easier and less notation is needed) and then explain the changes needed for the bipartite prefix-stable version. Our proof also provides a lower bound in terms of the length: for strings of length n, the difference can be as large as

$$\log\log n - O(\log\log\log n).$$

The proof can be found in the ArXiv version of this paper [1]. It uses game approach: We first explain the game rules, then show that a computable winning strategy in the game implies that the difference is unbounded, and finally, present that computable winning strategy. Both the game and the winning strategy has similarities with the game in [5].

6 Equality if the distance is superlogarithmic

Given the previous result, all distances become equal for pairs of strings of equal length, provided their distance is not too small.

▶ **Theorem 14.** *If $|x|=|y|$ and $E_1(x,y) \geqslant 6\log|x|$, then all four prefix information distances are equal to $E_1(x,y)+O(1)$.*

This seems to be the first equality in information theory whose precision becomes smaller if the quantity becomes larger. The proof can be found in the ArXiv version of this paper [1].

References

1 Bruno Bauwens and Alexander Shen. Information distance revisited. *arXiv preprint*, 2018. arXiv:1807.11087.

2 C. H. Bennett, P. Gács, M. Li, P.M.B. Vitányi, and W. H. Zurek. Information distance. *IEEE Transactions on Information Theory*, 44(4), 1998.

3 G.J. Chaitin. A theory of program size formally identical to information theory. *J. Assoc. Comput. Mach.*, 22(3):329–340, 1975. doi:10.1145/321892.321894.

4 P. Gács. On the symmetry of algorithmic information. *Soviet Math. Dokl.*, 15(5):1477–1480, 1974.

5 P. Gács. On the relation between descriptional complexity and algorithmic probability. *Theor. Comput. Sci.*, 22:71–93, 1983.

6 Leonid A Levin. *Some theorems on the algorithmic approach to probability theory and information theory.* PhD thesis, BostonUniversity, MA, UnitedStates, 1971. Dissertation directed by A. N. Kolmogorov; turned down as required by the Soviet authorities despite unanimously positive reviews. Translated in English in [10].

7 Leonid A Levin. On the notion of a random sequence. *Soviet Mathematics-Doklady*, 14:1413–1416, 1973.

8 Leonid A Levin. Laws of information conservation (nongrowth) and aspects of the foundation of probability theory. *Problemy Peredachi Informatsii*, 10(3):30–35, 1974.

9 Leonid A Levin. Various measures of complexity for finite objects (axiomatic description). *Soviet Mathematics Doklady*, 17(2):522–526, 1976.

10 Leonid A Levin. Some theorems on the algorithmic approach to probability theory and information theory. *Annals of Pureand Applied Logic*, 162:224–235, 2010. 1971 dissertation directed by A. N. Kolmogorov; turned down as required by the Soviet authorities despite unanimously positive reviews.

11 Ming Li and Paul M.B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications, 4th edition.* Springer, 2019. 1 ed., 1993; 2 ed., 1997, 3 ed 2008,.

12 Chong Long, Xiaoyan Zhu, Ming Li, and Bin Ma. Information shared by many objects. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 1213–1220. ACM, 2008.

13 MM Hassan Mahmud. On universal transfer learning. *Theoretical Computer Science*, 410(19):1826–1846, April 2009.

14 Alexander Shen, Vladimir A Uspensky, and Nikolay Vereshchagin. *Kolmogorov complexity and algorithmic randomness*, volume 220. Mathematical Surveys and Monographs, volume 220, xviii+511 pages. American Mathematical Society American Mathematical Soc., 2017. Draft version: http://www.lirmm.fr/~ashen/kolmbook-eng.pdf.

15 Paul M.B. Vitányi. Exact expression for information distance. *IEEE Transactions on Information Theory*, 63:4725–4728, 2017. arXiv:1410.7328.

On Computing Multilinear Polynomials Using Multi-r-ic Depth Four Circuits

Suryajith Chillara
CRI, University of Haifa, Israel
http://cmi.ac.in/~suryajith/
suryajith@cmi.ac.in

Abstract

In this paper, we are interested in understanding the complexity of computing multilinear polynomials using depth four circuits in which polynomial computed at every node has a bound on the individual degree of r (referred to as multi-r-ic circuits). The goal of this study is to make progress towards proving superpolynomial lower bounds for general depth four circuits computing multilinear polynomials, by proving better and better bounds as the value of r increases.

Recently, Kayal, Saha and Tavenas (Theory of Computing, 2018) showed that any depth four arithmetic circuit of bounded individual degree r computing a multilinear polynomial on $n^{O(1)}$ variables and degree $d = o(n)$, must have size at least $\left(\frac{n}{r^{1.1}}\right)^{\Omega\left(\sqrt{\frac{d}{r}}\right)}$ when r is $o(d)$ and is strictly less than $n^{1.1}$. This bound however deteriorates with increasing r. It is a natural question to ask if we can prove a bound that does not deteriorate with increasing r or a bound that holds for a *larger* regime of r.

We here prove a lower bound which does not deteriorate with r, however for a specific instance of $d = d(n)$ but for a *wider* range of r. Formally, we show that there exists an explicit polynomial on $n^{O(1)}$ variables and degree $\Theta(\log^2 n)$ such that any depth four circuit of bounded individual degree $r < n^{0.2}$ must have size at least $\exp\left(\Omega\left(\log^2 n\right)\right)$. This *improvement* is obtained by suitably adapting the complexity measure of Kayal et al. (Theory of Computing, 2018). This adaptation of the measure is inspired by the complexity measure used by Kayal et al. (SIAM J. Computing, 2017).

2012 ACM Subject Classification Theory of computation → Circuit complexity; Theory of computation → Algebraic complexity theory

Keywords and phrases Lower Bounds, Multilinear, Multi-r-ic

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.47

Funding The author was supported by the Institute Post Doctoral Fellowship at IIT Bombay where a part of this work was done. The author is currently supported by a CHE-PBC (VATAT) fellowship at University of Haifa. The author is also affiliated to Caesarea Rothschild Institute at University of Haifa.

Acknowledgements The author is grateful to Nutan Limaye and Srikanth Srinivasan for their invaluable feedback.

1 Introduction

One of the major focal points in the area of algebraic complexity theory is to show that certain polynomials are hard to compute syntactically. Here, the hardness of computation is quantified by the number of arithmetic operations that are needed to compute the target polynomial. Instead of the standard Turing machine model, we consider arithmetic circuits and formulas as models of computation.

Arithmetic circuits are directed acyclic graphs such that the leaf nodes are labeled by variables or constants from the underlying field, and every non-leaf node is labeled either by a $+$ or $\times$. Every node computes a polynomial by operating on its inputs with the operand

given by its label. The flow of computation flows from the leaf to the output node. We refer the readers to the standard resources [30, 29] for more information on arithmetic formulas and arithmetic circuits.

Valiant conjectured that the Permanent does not have polynomial sized arithmetic circuits [33]. Working towards that conjecture, we aim to prove superpolynomial circuit size lower bounds. However, the best known circuit size lower bound is $\Omega(n \log n)$, for a power symmetric polynomial, due to Baur and Strassen [31, 3], and, the best known formula size lower bound is $\Omega(n^2)$, due to Kalorkoti [13]. Due to the slow progress towards proving general circuit/formula lower bounds, it is natural to study some restricted class of arithmetic circuits and formulas.

Since most of the polynomials of interest such as Determinant, Permanent, etc., are multilinear polynomials, it is natural to consider the restriction where every intermediate computation is in fact multilinear. Due to the phenomenal work in the last two decades [23, 25, 24, 27, 28, 12, 26, 2, 6, 4, 5], the complexity of multilinear formulas and circuits is better understood than that of general formulas and circuits.

Backed with this progress it is natural to try to extend these results to a circuit model where the individual degree of every variable in the polynomial computed at every node in the circuit is r. We refer to these circuits as multi-r-ic circuits. When $r = 1$, the circuit model is multilinear.

Kayal and Saha [18] first studied multi-r-ic circuits of depth three and proved exponential lower bounds. Kayal, Saha and Tavenas [20] have extended this and proved exponential lower bounds at depth three and depth four. These circuits that were considered were syntactically multi-r-ic . That is, at any product node, any variable appears in the support of at most r many operands, and the total of the individual degrees is also at most r. Henceforth, all the multi-r-ic depth four circuits that we talk about shall be syntactically multi-r-ic .

Recently, Kumar, Oliviera and Saptharishi [21] showed that there is a chasm[1] for multi-r-ic circuits too. Formally, they showed that any polynomial sized (say n^c for a fixed constant c) multi-r-ic circuit of arbitrary depth computing a polynomial on n variables can be depth reduced to syntactical multi-r-ic depth four circuits of size $\exp(O(\sqrt{rn \log n}))$. This provides us a motivation to study multi-r-ic depth four circuits and prove strong lower bounds against them.

Kayal, Saha and Tavenas [20] proved an exponential size lower bound against multi-r-ic depth four circuits computing a variant of the iterated matrix multiplication polynomial. They achieved this bound using a measure that is inspired by the method *Shifted Partial Derivatives* [14, 9] and the method of *Skew Partial Derivatives* [17]. They referred to this new technique as the method of *Shifted Skew Partial Derivatives*. Hegde and Saha [11] improved upon [20] and showed a *near-optimal* size lower bound. However, the *best known* lower bounds are for polynomials that are multi-r-ic.

Motivation for this work

Raz and Yehudayoff [28] showed a lower bound of $\exp(\Omega\left(\sqrt{d \log d}\right))$ against multilinear depth four circuits which compute a multilinear polynomial over n variables and degree $d \ll n$ (cf. [20, Footnote 9]). Kayal, Saha and Tavenas [20] have shown a lower bound of $\left(\frac{n}{r^{1.1}}\right)^{\Omega\left(\sqrt{\frac{d}{r}}\right)}$ for a multilinear polynomial over $n^{O(1)}$ variables and degree d that is computed

[1] Agrawal and Vinay [1], Koiran, and Tavenas [32] showed that any general circuit can be depth reduced to a depth four circuit of non-trivial size.

by a multi-r-ic depth four circuit. This lower bound degrades with the increasing value of r and is superpolynomial only when r is $o(d)$ and is strictly less than $n^{1.1}$. This raises a natural question if the dependence on r could be improved upon.

In this work, we show that for a certain regime of d, we can prove a lower bound that does not deteriorate with increasing values of r.

▶ **Theorem 1.** *Let n and r be integers such that $r < n^{0.2}$. There exists an explicit polynomial Q_n of degree $\Theta(\log^2 n)$, over $n^{O(1)}$ variables such that any syntactically multi-r-ic depth four circuit computing it must have size* $\exp\left(\Omega(\log^2 n)\right)$.

The explicit polynomial that we consider can be expressed as a p-projection of an Iterated Matrix Multiplication polynomial $\mathsf{IMM}_{\tilde{n},\tilde{d}}$ (where $\tilde{n} = n^{O(1)}$ and $\tilde{d} = \Theta(\log^2 n)$) and thus Theorem 1 implies a lower bound of $n^{\Omega(\log n)}$ for Iterated Matrix Multiplication polynomial as well. By substituting for $d = \Theta(\log^2 n)$ into the bound from [20], we get that their bound evaluates to $n^{\Omega\left(\frac{\log n}{\sqrt{r}}\right)}$. Note that this bound is superpolynomial only when $r = o(\log^2 n)$. Thus lower bound is quantitatively better in this regime of parameters. In particular, we show a lower bound in the regimes of parameters where [20] cannot.

If we can show superpolynomial size lower bounds against multi-r-ic depth four circuits for $r = n^c$ for any constant c, then we indeed have superpolynomial circuit size lower bounds against depth four circuits. We believe that by building on the work of [20, 11], Theorem 1 is a step towards that direction.

Proof overview

Analogous to the work of Fournier et al. [8], and Kumar and Saraf [22], we first consider multi-r-ic depth four circuits of low bottom support[2] and prove lower bounds against them.

Let $T_1, T_2, \ldots, T_s$ be the terms corresponding to the product gates feeding into the output sum gate. The output polynomial is obtained by adding the terms $T_1, T_2, \ldots, T_s$. Note that each of these T_i's is a product of low support polynomials $Q_{i,j}$, that is, every monomial in these $Q_{i,j}$'s is supported on a small set of variables (say μ many). One major observation at this point is to see that there can be at most $N \cdot r$ many factors in any of the T_i's.

Kayal et al. [20] observed that the measure of shifted partial derivates [19, 8] does not yield any non-trivial lower bound if the number of factors is much larger than the number of variables itself. They worked around this obstacle by defining a *hybrid* complexity measure (refered to as *Shifted Skew Partial Derivatives*) where they first split the variables into two disjoint and unequal sets Y and Z such that $|Y| \gg |Z|$, then considered the *low* order partial derivatives with respect to only the Y variables and subsequently set all the Y variables to zero in the partial derivatives obtained. This effectively reduces the number of factors in a partial derivative to at most $|Z| \cdot r$. They then shift these polynomials by monomials in Z variables and look at the dimension of the polynomials thus obtained.

This measure gave them a size lower bound of $\left(\frac{n}{r^{1.1}}\right)^{\Omega\left(\sqrt{\frac{d}{r}}\right)}$ against multi-r-ic depth four circuits computing an explicit polynomial on $n^{O(1)}$ variables and degree $d = o(n)$ when $r = o(d)$. To improve the dependence on r in the lower bound, we consider a variant of *Shifted Skew Partial Derivatives* that we call *Projected Shifted Skew Partial Derivatives*. Here, we project down the space of Shifted Skew Partials and only look at the multilinear terms. Since the polynomial of interest is multilinear, it makes sense to only look at the multilinear

[2] That is, all the product gates at the bottom are supported on small set of variables.

terms obtained after the shifts of the skew partial derivatives. This is analogous to the method employed by Kayal et al. [16] to prove exponential lower bounds for Homogeneous depth four circuits, through the measure of *Projected Shifted Partial Derivatives*.

We first show that the dimension of Projected Shifted Skew Partial derivatives is not too large for small multi-r-ic depth four circuits of low bottom support. We then show that there exists an explicit polynomial whose dimension of Projected Shifted Skew Partial derivatives is large and thus cannot be computed by small multi-r-ic depth four circuits. We then lift this result to multi-r-ic depth four circuits for suitable set of parameters.

2 Preliminaries

Notation

- For a polynomial f, we use $\partial^{=k}_{Y}(f)$ to refer to the space of partial derivatives of order k of f with respect to monomials of degree k in Y.
- We use $\mathbf{z}^{=\ell}$ and $\mathbf{z}^{\leq\ell}$ to refer to the set of all the monomials of degree equal to ℓ and at most ℓ, respectively, in Z variables.
- We use $\mathbf{z}^{\leq\ell}_{\mathrm{ML}}$ to refer to the set of all the multilinear monomials of degree at most ℓ in Z variables.
- We use $\mathbf{z}^{\leq\ell}_{\mathrm{NonML}}$ to refer to the set of all the non-multilinear monomials of degree at most ℓ in Z variables.
- For a monomial m we use $|\mathrm{MonSupp}(m)|$ to refer to the size of the set of variables that appear in it.
- For a polynomial f, we use $|\mathrm{MonSupp}(f)|$ to refer to the maximum $|\mathrm{MonSupp}(m)|$ over all monomials in f.

Depth four circuits

A depth four circuit (denoted by $\Sigma\Pi\Sigma\Pi$) over a field $\mathbb{F}$ and variables $\{x_1, x_2, \ldots, x_n\}$ computes polynomials which can be expressed in the form of sums of products of polynomials. That is, $\sum_{i=1}^{s}\prod_{j=1}^{d_i} Q_{i,j}(x_1, \ldots, x_n)$ for some d_i's. A depth four circuit is said to have a bottom support of t (denoted by $\Sigma\Pi\Sigma\Pi^{\{t\}}$) if it is a depth four circuit and all the monomials in each polynomial $Q_{i,j}$ are supported on at most t variables.

Multi-r-ic arithmetic circuits

▶ **Definition 2** (multi-**r**-ic circuits)**.** *Let* $\mathbf{r} = (r_1, r_2, \cdots, r_n)$*. An arithmetic circuit* Φ *is said to be a syntactically multi-*$\mathbf{r}$*-ic circuit if for all product gates* $u \in \Phi$ *and* $u = u_1 \times u_2 \times \cdots \times u_t$*, each variable* x_i *can appear in at most* r_i *many of the* u_i*'s (*$i \in [t]$*). Further the total degree with respect to every variable over the polynomials computed at* $u_1, u_2, \cdots, u_t$*, is bounded by* r*, i.e.* $\sum_{j\in[t]} \deg_{x_i}(f_{u_j}) \leq r$ *for all* $i \in [n]$*. If* $\mathbf{r} = (r, r, \cdots, r)$*, then we simply refer to them as multi-*r*-ic circuits.*

Complexity Measure

We shall now describe our complexity measure which we shall henceforth refer to as Dimension of Projected Shifted Skew Partial Derivatives. This is a natural extension of the Dimension of Shifted Skew Partial Derivatives as used by [20].

This is analogous to the work of [15] where they study a *shifted partials inspired* measure called *Shifted Projected Partial derivatives* and then [16] where they study *Projected Shifted Partial derivatives*.

Since the polynomial of interest is multilinear, it does make sense for us to only look at those shifts of the partial derivatives that maintain multilinearity. At the same time, since the individual degree of the intermediate computations in the multi-r-ic depth four circuit is large and non-multilinear terms *cancel* out to generate the multilinear polynomial, we can focus on the multilinear terms generated after the shifts by projecting our linear space of polynomials down to them. We describe this process formally, below.

Let the variable set X be partitioned into two fixed, disjoint sets Y and Z such that $|Y|$ is an *order* larger than $|Z|$. Let $\sigma_Y : \mathbb{F}[Y \sqcup Z] \mapsto \mathbb{F}[Z]$ be a map such that for any polynomial $f(Y, Z)$, $\sigma_Y(f) \in \mathbb{F}[Z]$ is obtained by setting every variable in Y to zero by it and leaving Z variables untouched. Let $\text{mult} : \mathbb{F}[Z] \mapsto \mathbb{F}[Z]$ be a map such that for any polynomial $f(Y, Z)$, $\text{mult}(f) \in \mathbb{F}[Z]$ is obtained by setting the coeficients of all the non-multilinear monomials in f to 0 and leaving the rest untouched. We use $\mathbf{z}^{\leq \ell} \cdot \sigma_Y(\partial_Y^{=k} f)$ to refer to the linear span of polynomials obtained by multiplying each polynomial in $\sigma_Y(\partial_Y^{=k} f)$ with monomials of degree at most ℓ in Z variables. We will now define our complexity measure, Dimension of Projected Shifted Skew Partial Derivatives (denoted by $\Gamma_{k,\ell}$) as follows.

$$\Gamma_{k,\ell}(f) = \dim\left(\mathbb{F}\text{-span}\left\{\text{mult}\left(\mathbf{z}^{\leq \ell} \cdot \sigma_Y\left(\partial_Y^{=k} f\right)\right)\right\}\right)$$

This is a natural generalization of Shifted Skew Partial Derivatives of [20]. The following proposition is easy to verify.

▶ **Proposition 3** (Sub-additivity). *Let k and ℓ be integers. Let the polynomials f, f_1, f_2 be such that $f = f_1 + f_2$. Then, $\Gamma_{k,\ell}(f) \leq \Gamma_{k,\ell}(f_1) + \Gamma_{k,\ell}(f_2)$.*

Monomial Distance

We recall the following definition of distance between monomials from [7].

▶ **Definition 4** (Definition 2.7, [7]). *Let m_1, m_2 be two monomials over a set of variables. Let S_1 and S_2 be the multisets of variables corresponding to the monomials m_1 and m_2 respectively. The distance* $\text{dist}(m_1, m_2)$ *between the monomials m_1 and m_2 is the* $\min\{|S_1| - |S_1 \cap S_2|, |S_2| - |S_1 \cap S_2|\}$ *where the cardinalities are the order of the multisets.*

For example, let $m_1 = x_1^2 x_2 x_3^2 x_4$ and $m_2 = x_1 x_2^2 x_3 x_5 x_6$. Then $S_1 = \{x_1, x_1, x_2, x_3, x_3, x_4\}$, $S_2 = \{x_1, x_2, x_2, x_3, x_5, x_6\}$, $|S_1| = 6$, $|S_2| = 6$ and $\text{dist}(m_1, m_2) = 3$. It is important to note that two distinct monomials could have distance 0 between them if one of them is a multiple of the other and hence the triangle inequality does not hold.

The following beautiful lemma (from [9]) is key to the asymptotic estimates required for the lower bound analyses.

▶ **Lemma 5** (Lemma 6, [9]). *Let $a(n), f(n), g(n) : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ be integer valued functions such that $(f + g) = o(a)$. Then,*

$$\ln \frac{(a+f)!}{(a-g)!} = (f+g)\ln a \pm O\left(\frac{(f+g)^2}{a}\right)$$

We use the following strengthening of the Principle of Inclusion and Exclusion in our proof.

▶ **Lemma 6** (Strong Inclusion-Exclusion [22]). *Let $W_1, W_2, \cdots, W_t$ be subsets of a finite set W. For a parameter $\lambda \geq 1$, let $\sum_{\substack{i,j\in[t]\\ i\neq j}} |W_i \cap W_j| \leq \lambda \sum_{i\in[t]} |W_i|$. Then, $\left|\cup_{i\in[t]} W_i\right| \geq \frac{1}{4\lambda} \sum_{i\in[t]} |W_i|$.*

Polynomial Families

Let n, α, k be positive integers. We define the polynomial family $\{P_{n,\alpha,k}\}_{n,\alpha,k\geq 0}$ as follows. Let the variable set $X = \{x_1, \ldots, x_{N_0}\}$ be partitioned into two fixed, and disjoint sets Y and Z. We first define the polynomial family $f_{n,\alpha,k}(Y, Z)$ as follows (as it was defined in [20]).

$$f_{n,\alpha,k} = \prod_{i=1}^{k} g_i(Y_i, Z_i) \text{ where } g_i(Y_i, Z_i) = \sum_{a,b\in[n]} y_{a,b}^{(i)} \prod_{c\in[\alpha]} z_{a,c}^{(i,1)} z_{c+\alpha,b}^{(i,2)}.$$

It is easy to see that $|Y|$ is n^2k and $|Z|$ is $2\alpha nk$. We shall henceforth use m to refer to $|Z|$. Thus, $N_0 = |X| = |Y| + |Z| = k(n^2 + 2\alpha n)$.

Let c be a fixed constant in $(0, 1)$. We shall now define another polynomial family $P_{n,\alpha,k}$ based on the definition of $f_{n,\alpha,k}$. Let $p = N_0^{-c}$. Let $\hat{X} = \{\hat{x}_{1,1}, \hat{x}_{1,2}, \ldots, \hat{x}_{1,t}, \ldots, \hat{x}_{N_0,1}, \hat{x}_{N_0,2}, \ldots, \hat{x}_{N_0,t}\}$ be a variable set distinct from X such that $t = \frac{N_0 \log N_0}{p}$. Let $\mathrm{Lin}_p : X \mapsto \hat{X}$ be a linear map such that $x_i \mapsto \sum_{j=1}^{t} \hat{x}_{i,j}$ for all $i \in [N_0]$. Then the polynomial $P_{n,\alpha,k}(\hat{X})$ is defined to be $f_{n,\alpha,k} \circ \mathrm{Lin}_p(\hat{X})$. That is,

$$P_{n,\alpha,k} = f_{n,\alpha,k}\left(\sum_{j=1}^{t} \hat{x}_{1,j}, \sum_{j=1}^{t} \hat{x}_{2,j}, \cdots, \sum_{j=1}^{t} \hat{x}_{N_0,j}\right) \quad \text{where } t = \frac{N_0 \log N_0}{p}.$$

Note that $P_{n,\alpha,k}$ is a polynomial on $N = N_0^{2+c} \log N_0$ many variables.

We will now recall the following lemma which in the mentioned form is due to Kumar and Saptharishi [29].

▶ **Lemma 7** (Analogous to Lemma 20.5, [29]). *Let ρ be a random restriction on the variable set $\hat{X}$ that sets each variable to zero with a probability of at least $(1 - p)$ where $p = N_0^{-c}$ for some constant $c \in (0, 1)$. Then $f_{n,\alpha,k}(X)$ is a projection of $\rho(P_{n,\alpha,k}(\hat{X}))$ with a probability of at least $(1 - 2^{-N_0})$.*

3 Multi-r-ic Depth Four Circuits of Low Bottom Support

For some carefully chosen parameters k and ℓ, we shall first show that if a multi-r-ic depth four circuit C of bottom support μ is *small* then $\Gamma_{k,\ell}(C)$ is not too large. We shall then show that $\Gamma_{k,\ell}(f_{n,\alpha,k})$ is large and thus it cannot be computed by C.

3.1 Upper Bound on $\Gamma_{k,\ell}(C)$

Recall that C is a sum of at most s many products of polynomials $T^{(1)} + \cdots + T^{(s)}$ where each T_i is a syntactically multi-r-ic product of polynomials of low monomial support.

We shall first prove a bound on $\Gamma_{k,\ell}(T_i)$ for an arbitrary T_i and derive a bound on $\Gamma_{k,\ell}(C)$ by using sub-additivity of the measure (cf. Proposition 3).

Let T be a syntactic multi-r-ic product of polynomials $\tilde{Q}_1(Y, Z) \cdot \tilde{Q}_2(Y, Z) \cdot \ldots \cdot \tilde{Q}_D(Y, Z) \cdot R(Y)$ such that $\left|\mathrm{MonSupp}(\tilde{Q}_i)\right| \leq \mu$. We will first argue that D is not too large since T is a syntactically multi-r-ic product. We shall first pre-process the product T by doing the following procedure.

Repeat this process until all but at most one of the factors in T (except R) have a Z-support of at least $\frac{\mu}{2}$.

1. Pick two factors $\tilde{Q}_{i_1}$ and $\tilde{Q}_{i,2}$ such that they have the smallest Z-support amongst $Q_1, \cdots, Q_D$.
2. If both of them have support strictly less than $\frac{\mu}{2}$, merge these factors to obtain a new factor. Else, stop.

In the afore mentioned procedure, it is important to note that the monomial support post merging will still be at most μ since the factors being merged are of support strictly less than $\frac{\mu}{2}$. Henceforth, W.L.O.G we shall consider that every product gate at the top, in any multi-r-ic depth four circuit to be in a processed form.

Let $T = Q_1(Y,Z) \cdot Q_2(Y,Z) \cdot \ldots \cdot Q_t(Y,Z) \cdot R(Y)$ be the product obtained after the preprocessing. Each of the Q_i has a Z-support of at least $\frac{\mu}{2}$. The total Z-support is at most $|Z|\, r = mr$ since T is a syntactically multi-r-ic product. Thus t could at most be $\frac{2mr}{\mu}$.

▶ **Lemma 8.** *Let n, k, r, ℓ and μ be positive integers such that $\ell + k\mu < \frac{m}{2}$. Let T be a processed syntactic multi-r-ic product of polynomials $Q_1(Y,Z) \cdot Q_2(Y,Z) \cdot \ldots \cdot Q_t(Y,Z) \cdot R(Y)$ such that $|\mathrm{MonSupp}(Q_i)| \leq \mu$. Then, $\Gamma_{k,\ell}(T)$ is at most $\binom{t}{k} \cdot \binom{m}{\ell+k\mu} \cdot (\ell + k\mu)$.*

Before proving Lemma 8, we shall first use it to show an upper bound on the dimension of the space of Projected Shifted Skew Partial derivatives of a depth four multi-r-ic circuit of low bottom support.

▶ **Lemma 9.** *Let n, k, r, ℓ and μ be positive integers such that $\ell + k\mu < \frac{m}{2}$. Let C be a processed syntactic multi-r-ic depth four circuit of bottom support μ and size s. Then, $\Gamma_{k,\ell}(C)$ is at most $s \cdot \binom{\frac{2mr}{\mu}}{k} \cdot \binom{m}{\ell+k\mu} \cdot (\ell + k\mu)$.*

Proof. W.L.O.G we can assume that C be expressed as $\sum_i^s T^{(i)}$ such that $T^{(i)}$ is a processed syntactically multi-r-ic product of bottom support polynomials at most μ. From Proposition 3, we get that $\Gamma_{k,\ell}(C) \leq \sum_{i=1}^s \Gamma_{k,\ell}(T^{(i)})$. From the afore mentioned discussion we know that the number of factors in $T^{(i)}$ with a non-zero Z-support is at most $\frac{2mr}{\mu}$. From Lemma 8, we get that $\Gamma_{k,\ell}(T^{(i)})$ is at most $\binom{\frac{2mr}{\mu}}{k} \cdot \binom{m}{\ell+k\mu} \cdot (\ell + k\mu)$. By putting all of this together, we get that

$$\Gamma_{k,\ell}(C) \leq s \cdot \binom{\frac{2mr}{\mu}}{k} \cdot \binom{m}{\ell + k\mu} \cdot (\ell + k\mu)\,. \qquad \blacktriangleleft$$

Proof of Lemma 8. We will first show by induction on k, the following.

$$\partial_Y^{=k} T \subseteq \mathbb{F}\text{-span}\left\{ \bigcup_{S \in \binom{t}{t-k}} \left\{ \prod_{i \in S} Q_i(Y,Z) \cdot \mathbf{z}_{\mathrm{ML}}^{\leq k\mu} \cdot \mathbb{F}[Y] \right\} \right\}$$
$$\bigcup \mathbb{F}\text{-span}\left\{ \bigcup_{S \in \binom{t}{t-k}} \left\{ \prod_{i \in S} Q_i(Y,Z) \cdot \mathbf{z}_{\mathrm{NonML}}^{\leq kr\mu} \cdot \mathbb{F}[Y] \right\} \right\}$$

The base case of induction for $k = 0$ is trivial as T is already in the required form. Let us assume the induction hypothesis for all derivatives of order $< k$. That is, $\partial_Y^{=k-1} T$ can be expressed as a linear combination of terms of the form

$$h(X,Y) = \prod_{i \in S} Q_i(Y,Z) \cdot h_1(Z) \cdot h_2(Y)$$

where S is a set of size $t-(k-1)$, $h_1(Z)$ is a polynomial in Z variables of degree at most $(k-1)r\mu$, and $h_2(Y)$ is some polynomial in Y variables. In fact, $h_1(Z)$ can be expressed as a linear combination of multilinear monomials of degree at most $(k-1)\mu$, and non-multilinear monomials of degree at most $(k-1)r\mu$.

For some $u \in [|Y|]$ and some fixed i_0 in S,

$$\begin{aligned}
\frac{\partial h(Y,Z)}{\partial y_u} &= \left(\sum_{j\in S} \prod_{\substack{i\in S\\ i\neq j}} Q_i(Y,Z) \cdot \frac{\partial Q_j(Y,Z)}{\partial y_u} \cdot h_1(Z) \cdot h_2(Y) \right) \\
&\qquad + \frac{\prod_{i\in S} Q_i}{Q_{i_0}} \cdot Q_{i_0}(Y,Z) \cdot h_1(Z) \cdot \frac{\partial h_2(Y)}{\partial y_k} \\
&\in \mathbb{F}\text{-span}\left\{ \prod_{\substack{i\in S\\ i\neq j}} Q_i(Y,Z) \cdot \frac{\partial Q_j(Y,Z)}{\partial y_u} \cdot h_1(Z) \cdot \mathbb{F}[Y] \mid j \in [S] \right\} \\
&\qquad \bigcup \mathbb{F}\text{-span}\left\{ \frac{\prod_{i\in S} Q_i}{Q_{i_0}} \cdot Q_{i_0}(Y,Z) \cdot h_1(Z) \cdot \mathbb{F}[Y] \right\} \\
&\subseteq \mathbb{F}\text{-span}\left\{ \bigcup_{T\in\binom{S}{|S|-1}} \left\{ \prod_{i\in T} Q_i(Y,Z) \cdot \mathbf{z}^{\mu}_{\mathrm{ML}} \cdot h_1(Z) \cdot \mathbb{F}[Y] \right\} \right\} \\
&\qquad \bigcup \left\{ \bigcup_{T\in\binom{S}{|S|-1}} \left\{ \prod_{i\in T} Q_i(Y,Z) \cdot \mathbf{z}^{\leq r\mu}_{\mathrm{NonML}} \cdot h_1(Z) \cdot \mathbb{F}[Y] \right\} \right\} \\
&\subseteq \mathbb{F}\text{-span}\left\{ \bigcup_{T\in\binom{t}{t-k}} \left\{ \prod_{i\in T} Q_i(Y,Z) \cdot \mathbf{z}^{\leq k\mu}_{\mathrm{ML}} \cdot \mathbb{F}[Y] \right\} \right\} \\
&\qquad \bigcup \mathbb{F}\text{-span}\left\{ \bigcup_{T\in\binom{t}{t-k}} \left\{ \prod_{i\in T} Q_i(Y,Z) \cdot \mathbf{z}^{\leq kr\mu}_{\mathrm{NonML}} \cdot \mathbb{F}[Y] \right\} \right\}
\end{aligned}$$

The last inclusion follows from the fact that $h_1(Z)$ is a linear combination of multilinear monomials of degree at most $(k-1)\mu$, and non-multilinear monomials of degree at most $(k-1)r\mu$. From the discussion above we know that any polynomial in $\partial^k_Y(T)$ can be expressed as a linear combination of polynomials of the form $\frac{\partial h}{\partial y_u}$. Further every polynomial of the form $\frac{\partial h}{\partial y_u}$ belongs to the set

$$\begin{aligned}
W = \mathbb{F}\text{-span}&\left\{ \bigcup_{T\in\binom{t}{t-k}} \left\{ \prod_{i\in T} Q_i(Y,Z) \cdot \mathbf{z}^{\leq k\mu}_{\mathrm{ML}} \cdot \mathbb{F}[Y] \right\} \right\} \\
&\bigcup \mathbb{F}\text{-span}\left\{ \bigcup_{T\in\binom{t}{t-k}} \left\{ \prod_{i\in T} Q_i(Y,Z) \cdot \mathbf{z}^{\leq kr\mu}_{\mathrm{NonML}} \cdot \mathbb{F}[Y] \right\} \right\}.
\end{aligned}$$

Thus, we get that $\partial^{=k}T$ is a subset of W. This completes the proof by induction.

From the afore mentioned discussion, we can now derive the following expressions.

$$\sigma_Y\left(\partial_Y^{=k}T\right) \subseteq \mathbb{F}\text{-span}\left\{\bigcup_{S\in\binom{[t]}{t-k}}\left\{\prod_{i\in S}\sigma_Y(Q_i)\cdot \mathbf{z}_{\mathrm{ML}}^{\leq k\mu}\right\}\right\} \bigcup \mathbb{F}\text{-span}\left\{\bigcup_{S\in\binom{[t]}{t-k}}\left\{\prod_{i\in S}\sigma_Y(Q_i)\cdot \mathbf{z}_{\mathrm{NonML}}^{\leq kr\mu}\right\}\right\}$$

$$\mathbf{z}^{\leq \ell}\sigma_Y\left(\partial_Y^{=k}T\right) \subseteq \mathbb{F}\text{-span}\left\{\bigcup_{S\in\binom{[t]}{t-k}}\left\{\prod_{i\in S}\sigma_Y(Q_i)\cdot \mathbf{z}_{\mathrm{ML}}^{\leq \ell+k\mu}\right\}\right\} \bigcup \mathbb{F}\text{-span}\left\{\bigcup_{S\in\binom{[t]}{t-k}}\left\{\prod_{i\in S}\sigma_Y(Q_i)\cdot \mathbf{z}_{\mathrm{NonML}}^{\leq \ell+kr\mu}\right\}\right\}$$

$$\implies \mathbb{F}\text{-span}\left\{\mathrm{mult}\left(\mathbf{z}^{\leq \ell}\cdot\sigma_Y\left(\partial_Y^{=k}T\right)\right)\right\} \subseteq \mathbb{F}\text{-span}\left\{\bigcup_{S\in\binom{[t]}{t-k}}\left\{\prod_{i\in S}\mathrm{mult}(\sigma_Y(Q_i))\cdot \mathbf{z}_{\mathrm{ML}}^{\leq k\mu+\ell}\right\}\right\}.$$

Thus we get that $\dim\left(\mathbb{F}\text{-span}\left\{\mathrm{mult}\left(\mathbf{z}^{\leq \ell}\cdot\sigma_Y(\partial_Y^{=k}T)\right)\right\}\right)$ is at most

$$\dim\left(\mathbb{F}\text{-span}\left\{\bigcup_{S\in\binom{[t]}{t-k}}\left\{\prod_{i\in S}\mathrm{mult}(\sigma_Y(Q_i))\cdot \mathbf{z}_{\mathrm{ML}}^{\leq k\mu+\ell}\right\}\right\}\right)$$

$$\leq \dim\left(\mathbb{F}\text{-span}\left\{\bigcup_{S\in\binom{t}{t-k}}\left\{\prod_{i\in S}\mathrm{mult}(\sigma_Y(Q_i))\right\}\right\}\right)\cdot\dim\left(\mathbb{F}\text{-span}\left\{\mathbf{z}_{\mathrm{ML}}^{\leq k\mu+\ell}\right\}\right)$$

$$\leq \binom{t}{t-k}\cdot\sum_{i=0}^{k\mu+\ell}\binom{m}{i}$$

$$\leq \binom{t}{k}\cdot\binom{m}{\ell+k\mu}\cdot(\ell+k\mu) \qquad \text{(Since } \ell+k\mu < m/2\text{).}$$

◀

3.2 Lower Bound on $\Gamma_{k,\ell}(f_{n,\alpha,k})$

First we recall the generalized Hamming bound [10, Section 1.7].

▷ Claim 10. Let the vectors $\mathbf{a}=(a_1,a_2,\ldots,a_k)$ and $\mathbf{b}=(b_1,b_2,\ldots,b_k)$ correspond to the indices of the Y-monomial $y^{(1)}_{a_1,b_1}y^{(2)}_{a_2,b_2}\cdots y^{(k)}_{a_k,b_k}$ that is used to derive $f_{n,\alpha,k}$ with. For every $\Delta_0<k$, there is a subset $\mathcal{P}_{\Delta_0}\subset[n]^{2k}$ of size $\frac{n^{2k-\Delta_0}}{\Delta_0\binom{2k}{\Delta_0}}$ such that for all $(\mathbf{a},\mathbf{b}),(\mathbf{a}',\mathbf{b}')\in\mathcal{P}$, $\mathrm{dist}((\mathbf{a},\mathbf{b}),(\mathbf{a}',\mathbf{b}'))\geq\Delta_0$.

▶ **Observation 1.** *It is important to note that* $\frac{\partial^k f_{n,\alpha,k}}{y^{(1)}_{a_1,b_1}y^{(2)}_{a_2,b_2}\cdots y^{(k)}_{a_k,b_k}}$ *for any choice of* $(\mathbf{a},\mathbf{b})\in[n]^{2k}$ *is a multilinear monomial over just the* Z *variables.*

▷ Claim 11. Let $(\mathbf{a},\mathbf{b})$ and $(\mathbf{a}',\mathbf{b}')$ be such that $\text{dist}((\mathbf{a},\mathbf{b}),(\mathbf{a}',\mathbf{b}')) \geq \Delta_0$. Then $\text{dist}\left(\partial^k_{(\mathbf{a},\mathbf{b})} f_{n,\alpha,k}, \partial^k_{(\mathbf{a}',\mathbf{b}')} f_{n,\alpha,k}\right) \geq \alpha\Delta_0$.

Let $m_1, m_2, \ldots, m_t$ be the monomials in the set $\mathcal{M}_0 (= \partial^{=k}_{\mathcal{P}_{\Delta_0}} f_{n,\alpha,k})$, over Z variables such that $\text{dist}(m_i, m_j) \geq \Delta \geq \alpha\Delta_0$ for all $i \neq j$. Further, $\sigma_Y(\mathcal{M}_0) = \mathcal{M}_0$. Let $\mathcal{M}$ be the set of mutlilinear monomials of the form $m_i m'$ over Z-variables for some $1 \leq i \leq t$ where m' is a monomial of length ℓ. It is important to note that the set $\mathcal{M}$ now corresponds to the set $\mathbf{z}^{=\ell} \cdot \sigma_Y\left(\partial^{=k}_{\mathcal{P}_{\Delta_0}} f_{n,\alpha,k}\right)$. We shall now show that the cardinality of the set $\mathcal{M}$ is large enough for a suitable setting of parameters α, Δ_0 and k.

▶ **Lemma 12.** *Let $m, k, d, r, \Delta_0, \Delta, \ell$ and μ be positive integers such that $\ell + k\mu < \frac{m}{2}$, $(d-k)^2 = o(m)$, $\Delta^2 = o(m)$, $\Delta_0 = \delta k$ and $\ell = \frac{m}{2}(1-\varepsilon)$ for some fixed constants δ and ε. Then, $|\mathcal{M}| \geq \frac{1}{2}\left(\frac{2}{1-\varepsilon}\right)^{\delta\alpha k} \cdot \binom{m-(d-k)}{\ell}$.*

Proof. For all $i \in [t]$, Let B_i be the set of multilinear monomials of the form $m_i m'$ where m' is a multilinear monomial of degree ℓ. From the previous discussion, it follows that $|\mathcal{M}| = |\cup_{i=1}^t B_i|$. Using the principle of Inclusion and Exclusion, we get that

$$\left|\cup_{i=1}^t B_i\right| \geq \sum_{i=1}^t |B_i| - \sum_{i \neq j \in [t]}^t |B_i \cap B_j| \ .$$

▷ Claim 13. For all $i \in [t]$, $|B_i| = \binom{m-(d-k)}{\ell}$.

Proof. Since $\deg(m_i)$ is equal to $d-k$, the cardinality of B_i is equal to $\binom{m-(d-k)}{\ell}$. ◁

▷ Claim 14. For all $i, j \in [t]$ such that $i \neq j$, $|B_i \cap B_j| \leq \binom{m-(d-k)-\Delta}{\ell-\Delta}$.

Proof. Consider any two monomials $\hat{m}_i$ and $\hat{m}_j$ from B_i and B_j respectively. For $\hat{m}_i$ and $\hat{m}_j$ to be identical, $\hat{m}_i$ should contain at least Δ variables from $\hat{m}_j \setminus \hat{m}_i$ and similarly $\hat{m}_j$ should contain at least Δ variables from $\hat{m}_i \setminus \hat{m}_j$. The rest of the at most $(\ell - \Delta)$ many variables should be the same both in $\hat{m}_i$ and $\hat{m}_j$. The number of such multilinear monomials over Z variables is at most $\binom{m-(d-k)-\Delta}{\ell-\Delta}$. ◁

Putting Claim 13 and Claim 14 together, we get the following.

$$|\mathcal{M}| = \left|\cup_{i=1}^t B_i\right| \geq t\binom{m-(d-k)}{\ell} - \frac{t^2}{2}\binom{m-(d-k)-\Delta}{\ell-\Delta}.$$

Let $T_1 = t\binom{m-(d-k)}{\ell}$ and $T_2 = \frac{t^2}{2}\binom{m-(d-k)-\Delta}{\ell-\Delta}$. Let us consider the case where $T_2 = \lambda T_1$ where $\lambda \geq 1$ for some setting of the parameters Δ, α, ℓ and k.

$$\begin{aligned}
\lambda = \frac{T_2}{T_1} &= \frac{\frac{t^2}{2}\binom{m-(d-k)-\Delta}{\ell-\Delta}}{t\binom{m-(d-k)}{\ell}} \\
&= \frac{t}{2} \cdot \frac{(m-(d-k)-\Delta)!}{(\ell-\Delta)!(m-\ell-(d-k))!} \cdot \frac{(m-\ell-(d-k))!\ell!}{(m-(d-k))!} \\
&= \frac{t}{2} \cdot \frac{(m-(d-k)-\Delta)!}{(m-(d-k))!} \cdot \frac{\ell!}{(\ell-\Delta)!} \\
&= \frac{t}{2} \cdot \frac{(m-(d-k)-\Delta)!}{m!} \cdot \frac{m!}{(m-(d-k))!} \cdot \frac{\ell!}{(\ell-\Delta)!} \\
&= \frac{t}{2} \cdot \frac{m^{(d-k)} \cdot \ell^{\Delta}}{m^{(d-k)+\Delta}} && \text{(Using Lemma 5)} \\
&= \frac{t}{2} \cdot \left(\frac{\ell}{m}\right)^{\Delta}.
\end{aligned}$$

The math block above crucially uses the fact that $\Delta^2 = o(\ell)$ and $(d-k)^2 = o(m)$ while invoking Lemma 5. Since $\lambda \geq 1$, we get that $\frac{t}{2} \cdot \left(\frac{\ell}{m}\right)^{\Delta} \geq 1$. For some suitably fixed constants δ and ε, let Δ_0 be set to δk and ℓ be set to $\frac{m}{2}(1-\varepsilon)$. Recall that for a fixed Δ_0, $t = \frac{n^{2k-\Delta_0}}{\Delta_0\binom{2k}{\Delta_0}}$ and $\Delta = \alpha\Delta_0 = \delta\alpha k$. Thus,

$$\begin{aligned}
\frac{n^{2k-\Delta_0}}{2\binom{2k}{\Delta_0}} \cdot \left(\frac{\ell}{m}\right)^{\Delta} &\geq 1 \\
n^{2k-\Delta_0} &\geq 2\Delta_0 \left(\frac{m}{\ell}\right)^{\Delta} \binom{2k}{\Delta_0} \\
n^{2k-\Delta_0} &\geq \left(\frac{2}{1-\varepsilon}\right)^{\Delta} \left(\frac{2k}{\Delta_0}\right)^{\Delta_0} \\
n^{(2-\delta)k} &\geq \left(\frac{2}{1-\varepsilon}\right)^{\alpha\delta k} \left(\frac{2k}{\Delta_0}\right)^{\delta k}
\end{aligned}$$

and hence,

$$\alpha \leq \frac{(2-\delta)\log n - \delta \log\left(\frac{2}{\delta}\right)}{\delta \log\left(\frac{2}{1-\varepsilon}\right)}.$$

Invoking Lemma 6 with the previous discussion, we get that

$$|\mathcal{M}| \geq \frac{T_1}{4\lambda} = \frac{t\binom{m-(d-k)}{\ell}}{4\lambda} = \frac{1}{2}\left(\frac{m}{\ell}\right)^{\Delta} \cdot \binom{m-(d-k)}{\ell} = \frac{1}{2}\left(\frac{2}{1-\varepsilon}\right)^{\delta\alpha k} \cdot \binom{m-(d-k)}{\ell}. \blacktriangleleft$$

▶ **Lemma 15.** *Let m, k, d, r, ℓ and μ be positive integers such that $\ell + k\mu < \frac{m}{2}$, $\Delta_0 = \delta k$ and $\ell = \frac{m}{2}(1-\varepsilon)$ for some fixed constants δ and ε. Then, $\Gamma_{k,\ell}(f_{n,\alpha,k}) \geq |\mathcal{M}| \geq \frac{1}{2}\left(\frac{2}{1-\varepsilon}\right)^{\delta\alpha k} \cdot \binom{m-(d-k)}{\ell}$.*

Proof. Recall that $\mathcal{M}$ corresponds to the set $\mathbf{z}^{=\ell} \cdot \sigma_Y\left(\partial^{=k}_{\mathcal{P}_{\Delta_0}} f_{n,\alpha,k}\right)$. Since $\mathcal{M}$ is a bag of multilinear monomials over just the Z variables

$$|\mathcal{M}| = \dim\left(\mathbb{F}\text{-span}\left\{\text{mult}\left(\mathbf{z}^{=\ell} \cdot \sigma_Y\left(\partial^{=k}_{\mathcal{P}_{\Delta_0}} f_{n,\alpha,k}\right)\right)\right\}\right)$$

Since $\partial^{=k}_{\mathcal{P}^{\Delta_0}} f_{n,\alpha,k} \subseteq \partial^{=k} f_{n,\alpha,k}$ and $\mathbf{z}^{=\ell} \subseteq \mathbf{z}^{\leq \ell}$, we get that

$$\dim\left(\mathbb{F}\text{-span}\left\{\text{mult}\left(\mathbf{z}^{=\ell}\cdot\sigma_Y\left(\partial^{=k}_{\mathcal{P}^{\Delta_0}} f_{n,\alpha,k}\right)\right)\right\}\right) \leq \dim\left(\mathbb{F}\text{-span}\left\{\text{mult}\left(\mathbf{z}^{\leq \ell}\cdot\sigma_Y\left(\partial^{=k}_{Y} f_{n,\alpha,k}\right)\right)\right\}\right)$$
$$= \Gamma_{k,\ell}(f_{n,\alpha,k})\,.$$

Thus, $\Gamma_{k,\ell}(f_{n,\alpha,k}) \geq |\mathcal{M}| \geq \frac{1}{2}\left(\frac{2}{1-\varepsilon}\right)^{\delta\alpha k}\cdot\binom{m-(d-k)}{\ell}$. ◀

We shall now a size lower bound on the depth four multi-r-ic circuits of low bottom support that compute $f_{n,\alpha,k}$.

▶ **Theorem 16.** *Let $\delta = 0.25$ and $\varepsilon = 0.8$. Let n, r, α, k and μ be positive integers such that $r \leq n^{0.2}$, $\mu \leq \frac{\log n}{50}$ and $\alpha = \frac{(2-\delta)\log n}{0.9\delta\log\frac{2}{1-\varepsilon}}$. Let C be a depth four multi-r-ic circuit of low bottom support μ and size s. If C computes the polynomial $f_{n,\alpha,k}$ then s must be $\exp(\Omega(k\log n))$.*

Proof. Recall that the polynomial $f_{n,\alpha,k}$ is defined on the variable sets Y and Z such that $|Z| = m = 2\alpha nk$. Let ℓ be an integer such that $\ell = \frac{m}{2}(1-\varepsilon)$ and $\ell + k\mu < \frac{m}{2}$. Let $\Delta_0 = \delta k$. Let us assume that the polynomial $f_{n,\alpha,k}$ is computed by a depth four multi-r-ic circuit C of low bottom support μ and size s. Then it must be the case that $\Gamma_{k,\ell}(f_{n,\alpha,k}) = \Gamma_{k,\ell}(C)$.

$$\begin{aligned}
s &\geq \frac{\frac{1}{2}\left(\frac{2}{1-\varepsilon}\right)^{\delta\alpha k}\cdot\binom{m-(d-k)}{\ell}}{\binom{\frac{2mr}{\mu}}{k}\cdot\binom{m}{\ell+k\mu}\cdot(\ell+k\mu)} \\
&\geq \frac{1}{2(\ell+k\mu)}\cdot\left(\frac{2}{1-\varepsilon}\right)^{\delta\alpha k}\cdot\left(\frac{k\mu}{2emr}\right)^{k}\cdot\frac{\binom{m-(d-k)}{\ell}}{\binom{m}{\ell+k\mu}} \\
&= \frac{1}{2(\ell+k\mu)}\cdot\left(\frac{2}{1-\varepsilon}\right)^{\delta\alpha k}\cdot\left(\frac{k\mu}{2emr}\right)^{k}\cdot\frac{(m-(d-k))!}{m!}\cdot\frac{(m-\ell-k\mu)!}{(m-\ell-(d-k))!}\cdot\frac{(\ell+k\mu)!}{\ell!} \\
&\geq \frac{1}{2(\ell+k\mu)}\cdot\left(\frac{2}{1-\varepsilon}\right)^{\delta\alpha k}\cdot\left(\frac{k\mu}{2emr}\right)^{k}\cdot\frac{\ell^{k\mu}}{m^{(d-k)}}\cdot(m-\ell)^{(d-k)-k\mu} \\
&= \frac{1}{2(\ell+k\mu)}\cdot\left(\frac{2}{1-\varepsilon}\right)^{\delta\alpha k}\cdot\left(\frac{k\mu}{2emr}\right)^{k}\cdot\left(\frac{\ell}{m-\ell}\right)^{k\mu}\cdot\left(\frac{m-\ell}{m}\right)^{d-k} \\
&= \frac{1}{2(\ell+k\mu)}\cdot\left(\frac{2}{1-\varepsilon}\right)^{\delta\alpha k}\cdot\left(\frac{\mu}{4e\alpha nr}\right)^{k}\cdot\left(\frac{1-\varepsilon}{1+\varepsilon}\right)^{k\mu}\cdot\left(\frac{1+\varepsilon}{2}\right)^{d-k} \\
&= \frac{1}{2(\ell+k\mu)}\cdot\left(\frac{2}{1-\varepsilon}\right)^{\delta\alpha k}\cdot\left(\frac{\mu}{4e\alpha nr}\right)^{k}\cdot\left(\frac{1-\varepsilon}{1+\varepsilon}\right)^{k\mu}\cdot\left(\frac{1+\varepsilon}{2}\right)^{2\alpha k} \\
&= \frac{1}{2(\ell+k\mu)}\cdot\left(\left(\frac{2}{1-\varepsilon}\right)^{\delta}\cdot\left(\frac{1+\varepsilon}{2}\right)^{2}\right)^{\alpha k}\cdot\left(\frac{\mu}{4e\alpha nr}\right)^{k}\cdot\left(\frac{1-\varepsilon}{1+\varepsilon}\right)^{k\mu} \\
&= \frac{\exp\left(\alpha k\log\left(\left(\frac{2}{1-\varepsilon}\right)^{\delta}\cdot\left(\frac{1+\varepsilon}{2}\right)^{2}\right) - k\log n - k\log r - k\log\frac{4e\alpha}{\mu} + k\mu\log\frac{1-\varepsilon}{1+\varepsilon}\right)}{2(\ell+k\mu)}\,.
\end{aligned}$$

In the above math block, we use Lemma 5 to simplify the terms along with the fact that $k^2\mu^2 = o(m-\ell)$, $(d-k)^2 = o(m)$ and $k^2\mu^2 = o(\ell)$. To get a meaningful lower bound, we need $\alpha\log\left(\left(\frac{2}{1-\varepsilon}\right)^{\delta}\cdot\left(\frac{1+\varepsilon}{2}\right)^{2}\right)$ to be strictly greater than $(\log n + \log r + \log\frac{4e\alpha}{\mu})$. Let us set α to $\frac{(2-\delta)\log n}{0.9\delta\log\frac{2}{1-\varepsilon}}$. This reduces to showing that there exist constants δ, ε and ν such that

$$\frac{(2-\delta)\cdot\log\left(\left(\frac{2}{1-\varepsilon}\right)^{\delta}\cdot\left(\frac{1+\varepsilon}{2}\right)^{2}\right)}{0.9\delta\log\frac{2}{1-\varepsilon}} - 1 \geq \nu. \tag{1}$$

Let us fix the constants as follows: $\varepsilon = 0.8$, $\delta = 0.25$ and $\nu = 0.23$. Through some calculations, it can be verified that Equation 1 gets satisfied. Thus,

$$s \geq \frac{\exp\left(k\left(0.23\log n - \log r - \log\frac{4e\alpha}{\mu} + \mu\log\frac{1-\varepsilon}{1+\varepsilon}\right)\right)}{2(\ell + k\mu)}$$

If $\mu \leq \frac{\log n}{50}$ and $r \leq n^{0.2}$, we get that $s \geq \exp\left(\Omega\left(k\log n\right)\right)$. ◀

4 Multi-r-ic Depth Four Circuits

To prove the main theorem, we also need the following lemma.

▶ **Lemma 17** (Analogous to Lemma 20.4, [29]). *Let P be a multi-r-ic polynomial that is computed by a syntactically multi-r-ic depth 4 circuit C of size $s \leq N^{\gamma\mu}$ for some $\gamma > 0$. Let ρ be a random restriction that sets each variable to zero independently with probability $(1 - N^{-2\gamma})$. Then with probability at least $(1 - N^{-\gamma\mu})$ the polynomial $\rho(P)$ is computed by a multi-r-ic depth four circuit C' of bottom support at most μ and size s.*

Proof of Theorem 1. Let n be a large positive integer. Let us set some relevant parameters in terms of n or otherwise as follows.

$$\mu = \frac{\log n}{50}, \qquad \varepsilon = 0.8,$$
$$\delta = 0.25, \qquad \alpha = \frac{(2-\delta)\log n}{0.9\delta\log\frac{2}{1-\varepsilon}},$$
$$k = \frac{c}{100}\log n \qquad N_0 = k(n^2 + 2\alpha n),$$
$$N = N_0^{2+c}\log N_0, \qquad \gamma \text{ is a small constant such that } N^{2\gamma} \approxeq N_0^c.$$

Let $\hat{X} = \{\hat{x}_{1,1}, \hat{x}_{1,2}, \ldots, \hat{x}_{1,t}, \ldots, \hat{x}_{N_0,1}, \hat{x}_{N_0,2}, \ldots, \hat{x}_{N_0,t}\}$ be a set of variables over which the polynomial $P_{n,\alpha,k}$ is defined. Let $\rho : \hat{X} \mapsto \{0, *\}$ be a random restriction such that a variable is set to zero with a probability of $(1 - N^{-2\gamma})$, and is left untouched otherwise. Let C be a syntactically multi-r-ic depth four circuit of size $s \leq N^{\gamma\mu}$ that computes $P_{n,\alpha,k}$. Lemma 17 tells us that $C' = \rho(C)$ is a multi-r-ic depth four circuit of size s and bottom support at most μ with a probability of at least $(1 - N^{-\gamma\mu})$. Conditioned on this probability, $\rho(P_{n,\alpha,k})$ has a multi-r-ic $\Sigma\Pi\Sigma\Pi^{\{\mu\}}$ size at most s.

Lemma 7 tells us that $f_{n,\alpha,k}$ is a p-projection of $\rho(P_{n,\alpha,k})$ with a probability of $(1 - 2^{-N_0})$ and hence $f_{n,\alpha,k}$ also has a multi-r-ic $\Sigma\Pi\Sigma\Pi^{\{\mu\}}$ of size at most s with a probability of at least $(1 - N^{-\gamma\mu} - 2^{-N_0})$. In other words, there is a random restriction σ such that $f_{n,\alpha,k}$ is a p-projection of $\rho(P_{n,\alpha,k})$ and $C' = \rho(C)$ is a multi-r-ic $\Sigma\Pi\Sigma\Pi^{\{\mu\}}$ circuit.

On the other hand, from Theorem 16 we know that any multi-r-ic $\Sigma\Pi\Sigma\Pi^{\{\mu\}}$ circuit that computes $f_{n,\alpha,k}$ must be of size $\exp(\Omega(k\log n))$. Thus, it must be that $\exp(\Omega(k\log n)) \leq s \leq N^{\gamma\mu}$. We can choose c to be a small enough constant such that the aforementioned expression is satisfied. Thus, s must at least be $\exp\left(\Omega(\log^2 n)\right)$. The explicit polynomial Q_n is $P_{n,\alpha,k}$ where $\alpha = \frac{(2-\delta)\log n}{0.9\delta\log\frac{2}{1-\varepsilon}}$ and $k = \frac{c}{100}\log n$. ◀

References

1. Manindra Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *proceedings of Foundations of Computer Science (FOCS)*, pages 67–75, 2008. doi:10.1109/FOCS.2008.32.
2. Noga Alon, Mrinal Kumar, and Ben Lee Volk. Unbalancing sets and an almost quadratic lower bound for syntactically multilinear arithmetic circuits. In *CCC*, volume 102 of *LIPIcs*, pages 11:1–11:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
3. Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theor. Comput. Sci.*, 22:317–330, 1983.
4. Suryajith Chillara, Christian Engels, Nutan Limaye, and Srikanth Srinivasan. A near-optimal depth-hierarchy theorem for small-depth multilinear circuits. In *FOCS*, pages 934–945. IEEE Computer Society, 2018.
5. Suryajith Chillara, Nutan Limaye, and Srikanth Srinivasan. A quadratic size-hierarchy theorem for small-depth multilinear formulas. In *ICALP*, volume 107 of *LIPIcs*, pages 36:1–36:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
6. Suryajith Chillara, Nutan Limaye, and Srikanth Srinivasan. Small-depth multilinear formula lower bounds for iterated matrix multiplication with applications. *SIAM J. Comput.*, 48(1):70–92, 2019.
7. Suryajith Chillara and Partha Mukhopadhyay. Depth-4 lower bounds, determinantal complexity: A unified approach. *computational complexity*, May 2019. doi:10.1007/s00037-019-00185-4.
8. Hervé Fournier, Nutan Limaye, Guillaume Malod, and Srikanth Srinivasan. Lower bounds for depth-4 formulas computing iterated matrix multiplication. *SIAM J. Comput.*, 44(5):1173–1201, 2015.
9. Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Approaching the chasm at depth four. *Journal of the ACM (JACM)*, 61(6):33, 2014.
10. Venkatesan Guruswami, Atri Rudra, and Madhu Sudan. Essential coding theory, 2019. URL: https://cse.buffalo.edu/faculty/atri/courses/coding-theory/book/.
11. Sumant Hegde and Chandan Saha. Improved lower bound for multi-r-ic depth four circuits as a function of the number of input variables. *Proceedings of the Indian National Science Academy*, 83(4):907–922, 2017.
12. Pavel Hrubeš and Amir Yehudayoff. Homogeneous formulas and symmetric polynomials. *Computational Complexity*, 20(3):559–578, 2011.
13. K. Kalorkoti. A lower bound for the formula size of rational functions. *SIAM J. Comput.*, 14(3):678–687, 1985.
14. Neeraj Kayal. An exponential lower bound for the sum of powers of bounded degree polynomials. *Electronic Colloquium on Computational Complexity (ECCC)*, 19:81, 2012.
15. Neeraj Kayal, Nutan Limaye, Chandan Saha, and Srikanth Srinivasan. Super-polynomial lower bounds for depth-4 homogeneous arithmetic formulas. In *STOC*, pages 119–127. ACM, 2014.
16. Neeraj Kayal, Nutan Limaye, Chandan Saha, and Srikanth Srinivasan. An exponential lower bound for homogeneous depth four arithmetic formulas. *SIAM J. Comput.*, 46(1):307–335, 2017.
17. Neeraj Kayal, Vineet Nair, and Chandan Saha. Separation between read-once oblivious algebraic branching programs (ROABPs) and multilinear depth three circuits. In *proceedings of Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 46:1–46:15, 2016. doi:10.4230/LIPIcs.STACS.2016.46.
18. Neeraj Kayal and Chandan Saha. Multi-k-ic depth three circuit lower bound. *Theory Comput. Syst.*, 61(4):1237–1251, 2017.
19. Neeraj Kayal, Chandan Saha, and Ramprasad Saptharishi. A super-polynomial lower bound for regular arithmetic formulas. In *STOC*, pages 146–153. ACM, 2014.
20. Neeraj Kayal, Chandan Saha, and Sébastien Tavenas. On the size of homogeneous and of depth-four formulas with low individual degree. *Theory of Computing*, 14(16):1–46, 2018. doi:10.4086/toc.2018.v014a016.

21 Mrinal Kumar, Rafael Mendes de Oliveira, and Ramprasad Saptharishi. Towards optimal depth reductions for syntactically multilinear circuits. In *ICALP*, volume 132 of *LIPIcs*, pages 78:1–78:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019.

22 Mrinal Kumar and Shubhangi Saraf. On the power of homogeneous depth 4 arithmetic circuits. *SIAM J. Comput.*, 46(1):336–387, 2017.

23 Noam Nisan and Avi Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Computational Complexity*, 6(3):217–234, 1997. doi:10.1007/BF01294256.

24 Ran Raz. Multilinear-$\mathsf{NC}^2 \neq$ multilinear-NC^1. In *proceedings of Foundations of Computer Science (FOCS)*, pages 344–351, 2004. doi:10.1109/FOCS.2004.42.

25 Ran Raz. Separation of multilinear circuit and formula size. *Theory of Computing*, 2(1):121–135, 2006. doi:10.4086/toc.2006.v002a006.

26 Ran Raz, Amir Shpilka, and Amir Yehudayoff. A lower bound for the size of syntactically multilinear arithmetic circuits. *SIAM Journal of Computing*, 38(4):1624–1647, 2008. doi:10.1137/070707932.

27 Ran Raz and Amir Yehudayoff. Balancing syntactically multilinear arithmetic circuits. *Computational Complexity*, 17(4):515–535, 2008. doi:10.1007/s00037-008-0254-0.

28 Ran Raz and Amir Yehudayoff. Lower bounds and separations for constant depth multilinear circuits. *Computational Complexity*, 18(2):171–207, 2009. doi:10.1007/s00037-009-0270-8.

29 Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity version 8.0.4. Github survey, 2019. URL: https://github.com/dasarpmar/lowerbounds-survey/releases/.

30 Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5:207–388, March 2010. doi:10.1561/0400000039.

31 Volker Strassen. Berechnungen in partiellen algebren endlichen typs. *Computing*, 11(3):181–196, 1973.

32 Sébastien Tavenas. Improved bounds for reduction to depth 4 and depth 3. *Information and Computation*, 240:2–11, 2015. doi:10.1016/j.ic.2014.09.004.

33 Leslie G. Valiant. Completeness classes in algebra. In *STOC*, pages 249–261. ACM, 1979.

A Missing Proofs

Proof of Claim 10. There are n^{2k} elements in $\mathcal{P}$. It is easy to see that the volume of a Hamming Ball of radius Δ_0 for vectors of length $2k$ is at most $\sum_{i=0}^{\Delta_0} \binom{2k}{i} \cdot n^i \leq \Delta_0 \binom{2k}{\Delta_0} n^{\Delta_0}$ and thus there are at most $\binom{2k}{\Delta_0} n^{\Delta_0}$ many vectors $(\mathbf{a}, \mathbf{b})$ in that Hamming ball. Thus, there exists a packing of these Hamming balls in $\mathcal{P}$ with at least $\frac{n^{2k-\Delta_0}}{\Delta_0 \binom{2k}{\Delta_0}}$ many balls. ◁

Proof of Claim 11. For a vector $(\mathbf{a}, \mathbf{b}) \in [n]^{2k}$, $\frac{\partial^k f_{n,\alpha,k}}{y^{(1)}_{a_1,b_1} y^{(2)}_{a_2,b_2} \cdots y^{(k)}_{a_k,b_k}} = \prod_{i=1}^{k} \prod_{v \in [\alpha]} z^{i,1}_{a_i,v} \cdot z^{i,1}_{v+\alpha,b_i}$. For all $i \in [k]$, let $B_i(\mathbf{a}, \mathbf{b}) = \prod_{v \in [\alpha]} z^{i,1}_{a_i,v} \cdot z^{i,1}_{v+\alpha,b_i}$. Note that for some $i \in [k]$, if $a_i \neq a'_i$, $\mathrm{dist}(B_i(\mathbf{a}, \mathbf{b}), (\mathbf{a}', \mathbf{b}'))$ is at least α. Similar is the case when $b_i \neq b'_i$. Thus, if $\mathrm{dist}((\mathbf{a}, \mathbf{b}), (\mathbf{a}', \mathbf{b}')) \geq \Delta_0)$, there are at least Δ_0 many locations where either $a_i \neq a'_i$ or $b_i \neq b'_i$ and hence $\mathrm{dist}\left(\partial^k_{(\mathbf{a},\mathbf{b})} f_{n,\alpha,k}, \partial^k_{(\mathbf{a}',\mathbf{b}')} f_{n,\alpha,k}\right) \geq \alpha\Delta_0$. ◁

The following proofs are a step by step adaptation, rather a replication of proofs Lemma 20.4 and Lemma 20.5 respectively in [29].

Proof of Lemma 17. Let C be a multi-r-ic depth four circuit of size s. Let $m_1, m_2, \cdots, m_t$ be the set of monomials computed at the lower product gate of C, that have at least μ distinct variables in their support. Note that t is at most s. For all $i \in [t]$, $\Pr[\rho(m_i) \neq 0] \leq N^{-2\gamma\mu}$. By taking an union bound, the probability that there exists in a monomial amongst

$m_1, m_2, \cdots, m_t$ which is not set to 0 by ρ is at most $t \cdot N^{-2\gamma\mu} \leq s \cdot N^{-2\gamma\mu} \leq N^{-\gamma\mu}$. Thus with a probability of at least $(1 - N^{-\gamma\mu})$, all the monomials at the bottom product gate have at most μ distinct variables in their support. ◀

Proof of Lemma 7. For all $i \in [N_0]$,

$$\Pr[\rho(\hat{x}_{i,1}) = \rho(\hat{x}_{i,2}) = \cdots \rho(\hat{x}_{i,t}) = 0] = (1 - N_0^{-c})^t \approx \frac{1}{N_0 2^{N_0}}.$$

By union bound, probability that there exists an $i \in [N_0]$ such that all the variables of the form $\hat{x}_{i,j}$ for $j \in [t]$ are set to zero is at most $\frac{1}{2^{N^0}}$. Thus, with a probability of at least $(1 - 2^{-N_0})$, for each i, there exists at least one j such that $\rho(\hat{x}_{i,j}) \neq 0$.It is easy to see that the polynomial $f_{n,\alpha,k}$ can be written as a p-projection of $\rho(P_{n,\alpha,k})$ in such a case. ◀

Observation and Distinction. Representing Information in Infinite Games

Dietmar Berwanger
LSV, CNRS & ENS Paris-Saclay, France
dwb@lsv.fr

Laurent Doyen
LSV, CNRS & ENS Paris-Saclay, France
doyen@lsv.fr

Abstract

We compare two approaches for modelling imperfect information in infinite games by using finite-state automata. The first, more standard approach views information as the result of an observation process driven by a sequential Mealy machine. In contrast, the second approach features indistinguishability relations described by synchronous two-tape automata.

The indistinguishability-relation model turns out to be strictly more expressive than the one based on observations. We present a characterisation of the indistinguishability relations that admit a representation as a finite-state observation function. We show that the characterisation is decidable, and give a procedure to construct a corresponding Mealy machine whenever one exists.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation → Representations of games and their complexity; Computing methodologies → Reasoning about belief and knowledge; Computing methodologies → Planning under uncertainty

Keywords and phrases Infinite Games on Finite Graphs, Imperfect Information, Automatic Structures

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.48

Related Version A full version of the paper is available at [2], `https://arxiv.org/abs/1809.05978`.

1 Introduction

Uncertainty is a main concern in strategic interaction. Decisions of agents are based on their knowledge about the system state, and that is often limited. The challenge grows in dynamical systems, where the state changes over time, and it becomes severe, when the dynamics unravels over infinitely many stages. In this context, one fundamental question is how to model knowledge and the way it changes as information is acquired along the stages of the system run.

Finite-state automata offer a solid framework for the analysis of systems with infinite runs. They allow to reason about infinite state spaces in terms of finite ones – of course, with a certain loss. The connection has proved to be extraordinarily successful in the study of infinite games on finite graphs, in the particular setting of *perfect information* assuming that players are informed about every move in the play history, which determines the actual state of the system. One key insight is that winning strategies, in this setting, can be synthesized effectively [6, 23]: for every game described by finite automata, one can describe the set of winning strategies by an automaton (over infinite trees) and, moreover, construct an automaton (a finite-state Moore machine) that implements a winning strategy.

In this paper, we discuss two approaches for modelling *imperfect information*, where, in contrast to the perfect-information setting, it is no longer assumed that the decision maker is informed about the moves that occurred previously in the play history.

The first, more standard approach corresponds to viewing information as a result of an observation *process* that may be imperfect in the sense that different moves can yield the same observation in a stage of the game. Here, we propose a second approach, which corresponds to representing information as a *state* of knowledge, by describing which histories are indistinguishable to the decision maker.

Concretely, we assume a setting of synchronous games with perfect recall in a partitional information model. Plays proceed in infinitely many stages, each of which results in one move from a finite range. Histories and plays are thus determined as finite or infinite sequences of moves, respectively.

To represent information partitions, we consider two models based on finite-state automata. In the observation-based model, which corresponds to the standard approach in computing science and non-cooperative game theory, the automaton is a sequential Mealy machine that inputs moves and outputs observations from a finite alphabet. The machine thus describes an observation function, which maps any history of moves to a sequence of observations that represents its information set. In the indistinguishability-based model, we use two-tape automata to describe which pairs of histories belong to the same information set.

As an immediate insight, we point out that, in the finite-state setting, the standard model based on observation functions is less expressive than the one based on indistinguishability relations. Intuitively, this is because observation functions can only yield a bounded amount of information in each round – limited by the size of the observation alphabet, whereas indistinguishability relations can describe situations where the amount of information received per round grows unboundedly as the play proceeds.

We investigate the question whether an information partition represented as (an indistinguishability relation given by) a two-tape automaton admits a representation as (an observation function given by) a Mealy machine. We show that this question is decidable, using results from the theory of word-automatic structures. We also present a procedure for constructing a Mealy machine that represents a given indistinguishability relation as an observation function, whenever this is possible.

2 Basic Notions

2.1 Finite automata

To represent components of infinite games as finite objects, finite-state automata offer a versatile framework (see [13], for a survey). Here, we use automata of two different types, which we introduce following the notation of [22, Chapter 2].

As a common underlying model, a *semi-automaton* is a tuple $\mathcal{A} = (Q, \Gamma, q_\varepsilon, \delta)$ consisting of a finite set Q of *states*, a finite *input alphabet* Γ, a designated *initial state* $q_\varepsilon \in Q$, and a *transition function* $\delta\colon Q \times \Gamma \to Q$. We define the size $|\mathcal{A}|$ of $\mathcal{A}$ to be the number of its transitions, that is $|Q| \cdot |\Gamma|$. To describe the internal behaviour of the semi-automaton we extend the transition function from letters to input words: the extended transition function $\delta\colon Q \times \Gamma^* \to Q$ is defined by setting, for every state $q \in Q$,

- $\delta(q, \varepsilon) := q$ for the empty word ε, and
- $\delta(q, \tau c) := \delta(\delta(q, \tau), c)$, for any word obtained by the concatenation of a word $\tau \in \Gamma^*$ and a letter $c \in \Gamma$.

On the one hand, we use automata as acceptors of finite words. A *deterministic finite automaton* (for short, DFA) is a tuple $\mathcal{A} = (Q, \Gamma, q_\varepsilon, \delta, F)$ expanding a semi-automaton by a designated subset $F \subseteq Q$ of *accepting states*. We say that a finite input word $\tau \in \Gamma^*$ is *accepted* by $\mathcal{A}$ from a state q if $\delta(q, \tau) \in F$. The set of words in Γ^* that are accepted by $\mathcal{A}$ from the initial state q_ε forms its *language*, denoted $L(\mathcal{A}) \subseteq \Gamma^*$.

Thus, a DFA recognises a set of words. By considering input alphabets over pairs of letters from a basis alphabet Γ, the model can be used to recognise synchronous relations over Γ, that is, relations between words of the same length. We refer to a DFA over an input alphabet $\Gamma \times \Gamma$ as a *two-tape* DFA. The relation recognised by such an automaton consists of all pairs of words $c_1 c_2 \dots c_\ell, c'_1 c'_2 \dots c'_\ell \in \Gamma^*$ such that $(c_1, c'_1)(c_2, c'_2) \dots (c_\ell, c'_\ell) \in L(\mathcal{A})$. With a slight abuse of notation, we also denote this relation by $L(\mathcal{A})$. We say that a synchronous relation is regular if it is recognised by a DFA.

On the other hand, we consider automata with output. A *Mealy* automaton is a tuple $(Q, \Gamma, \Sigma, q_\varepsilon, \delta, \lambda)$ where $(Q, \Gamma, q_\varepsilon, \delta)$ is a semi-automaton, Σ is a finite *output alphabet*, and $\lambda \colon Q \times \Gamma \to \Sigma$ is an output function. To describe the external behaviour of such an automaton, we define the extended output function $\lambda \colon \Gamma^* \times \Gamma \to \Sigma$ by setting $\lambda(\tau, c) := \lambda(\delta(q_\varepsilon, \tau), c)$ for every word $\tau \in \Gamma^*$ and every letter $c \in \Gamma$. Thus, the external behaviour of a Mealy automaton defines a function from the set $\Gamma^+ := \Gamma^* \setminus \{\varepsilon\}$ of nonempty histories to Σ. We say that a function on Γ^+ is *regular*, if there exists a Mealy automaton that defines it.

2.2 Repeated games with imperfect information

In our general setup, we consider games played in an infinite sequence of stages. In each stage, every player chooses an action from a given set of alternatives, independently and simultaneously. As a consequence, this determines a move that is recorded in the play history. Then, the game proceeds to the next stage. The outcome of the play is thus an infinite sequence of moves.

Decisions of a player are based on the available information, which we model by a partition of the set of play histories into information sets: at the beginning of each stage game, the player is informed of the information set to which the actual play history belongs (in the partition associated to the player). Accordingly, a strategy for a player is a function from information sets to actions. Every strategy profile (that is, a collection of strategies, one for each player) determines a play.

Basic questions in this setup concern strategies of an individual player to enforce an outcome in a designated set of winning plays or to maximise the value of a given payoff function, regardless of the strategy of other players. More advanced issues target joint strategies of coalitions among players towards coordinating on a common objective, or equilibrium profiles. Scenarios where the available actions depend on the history, or where the play might end after finitely many stages, can be captured by adjusting the information partition together with the payoff or winning condition.

For our formal treatment of information structures, we use the model of abstract infinite games as introduced by Thomas in his seminal paper on strategy synthesis [26]; the relevant questions for more elaborate settings, such as infinite games on finite graphs or concurrent game structures can be reduced easily to this abstraction. The underlying model is consistent with the classical definition of extensive games with information partitions and perfect recall due to von Neumann and Morgenstern [28], in the formulation of Kuhn [15]. For a more detailed account on partitional information, we refer to Bacharach [1] and Geanakoplos [11].

Our formalisation captures the information structures of repeated games with imperfect monitoring as studied in non-cooperative game theory (see the survey of Gossner and Tomala [12]), and of infinite games with partial observation on finite-state systems as studied in computing science (see Reif [25], Lin and Wonham [18], van der Meyden and Wilke [27], Chatterjee et al. [7], Berwanger et al. [3]). For background on the modelling of knowledge, and the notion of synchronous perfect recall we refer to Chapter 8 in the book of Fagin et al. [9].

2.2.1 Move and information structure

As a basic object for describing a game, we fix a finite set Γ of *moves*. A *play* is an infinite sequence of moves $\pi = c_1 c_2 \ldots \in \Gamma^\omega$. A *history* (of length ℓ) is a finite prefix $\tau = c_1 c_2 \ldots c_\ell \in \Gamma^*$ of a play; the empty history ε has length zero. The *move structure* of the game is the set Γ^* of histories equipped with the successor relation, which consists of all pairs $(\tau, \tau c)$ for $\tau \in \Gamma^*$ and $c \in \Gamma$. For convenience, we denote the move structure of a game on Γ simply by Γ^* omitting the (implicitly defined) successor relation.

The information available to a player is modeled abstractly by a partition $\mathcal{U}$ of the set Γ^* of histories; the parts of $\mathcal{U}$ are called *information sets* (of the player). The intended meaning is that if the actual history belongs to an information set U, then the player considers every history in U possible. The particular case where all information sets in the partition are singletons characterises the setting of *perfect information*.

The *information structure* (of the player) is the quotient $\Gamma^*/_{\mathcal{U}}$ of the move structure by the information partition. That is, the first-order structure on the domain consisting of the information sets, with a binary relation connecting two information sets (U, U') whenever there exists a history $\tau \in U$ with a successor history $\tau c \in U'$. Generally, we assume the perspective of just one player, so we simply refer to the information structure of the game.

Our information model is *synchronous*, which means, intuitively, that the player always knows how many stages have been played. Formally, this amounts to asserting that all histories in an information set have the same length; in particular the empty history forms a singleton information set. Further, we assume that the player has *perfect recall* – he never forgets what he knew previously. Formally, if an information set contains nonempty histories τc and $\tau' c'$, then the predecessor history τ is in the same information set as τ'. In different terms, an information partition satisfies synchronous perfect recall if whenever a pair of histories $c_1 \ldots c_\ell$ and $c'_1 \ldots c'_\ell$ belongs to an information set, then for every stage $t \leq \ell$, the prefix histories $c_1 \ldots c_t$ and $c'_1 \ldots c'_t$ belong to the same information set. As a direct consequence, the information structures that arise from such partitions are indeed trees.

▶ **Lemma 1.** *For every information partition $\mathcal{U}$ of perfect synchronous recall, the information structure $\Gamma^*/_{\mathcal{U}}$ is a directed tree.*

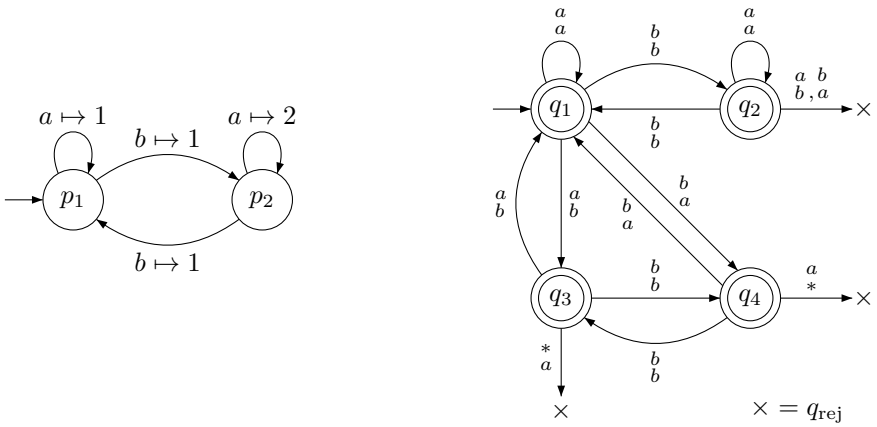
We will use the term *information tree* when referring to the information structure associated with an information partition with synchronous perfect recall.

In the following, we discuss two alternative representations of information partitions.

2.2.2 Observation

The first alternative consists in describing the information received by the player in each stage. To do so, we specify a set Σ of *observation symbols* and an *observation function* $\beta \colon \Gamma^+ \to \Sigma$. Intuitively, the player observes at every nonempty history τ the symbol $\beta(\tau)$; under the assumption of perfect recall, the information available to the player at history $\tau = c_1 c_2 \ldots c_\ell$ is thus represented by the sequence of observations $\beta(c_1)\beta(c_1 c_2) \ldots \beta(c_1 \ldots c_\ell)$, which we call *observation history* (at τ); let us denote by $\hat{\beta} \colon \Gamma^* \to \Sigma^*$ the function that returns, for each play history, the corresponding observation history.

The information partition $\mathcal{U}_\beta$ represented by an observation function β is the collection of sets $U_\eta := \{\tau \in \Gamma^* \mid \hat{\beta}(\tau) = \eta\}$ indexed by observation histories $\eta \in \hat{\beta}(\Gamma^*)$. Clearly, information partitions described in this way verify the conditions of synchronous perfect recall: each information set U_η consists of histories of the same length (as η), and for every pair τ, τ' of histories with different observations $\hat{\beta}(\tau) \neq \hat{\beta}(\tau')$, and every pair of moves $c, c' \in \Gamma$, the observation history of the successors τc and $\tau' c'$ will also differ $\hat{\beta}(\tau c) \neq \hat{\beta}(\tau' c')$.



(a) Mealy automaton. **(b)** Two-tape DFA.

Figure 1 A Mealy automaton and a two-tape DFA over alphabet $\Gamma = \{a, b\}$ describing the same information partition (the symbol $*$ stands for $\{a, b\}$).

To describe observation functions by a finite-state automaton, we fix a *finite* set Σ of observations and specify a Mealy automaton $\mathcal{M} = (Q, \Gamma, \Sigma, q_\varepsilon, \delta, \lambda)$, with moves from Γ as input and observations from Σ as output. Then, we consider the extended output function of $\mathcal{M}$ as an observation function $\beta_{\mathcal{M}} \colon \Gamma^+ \to \Sigma$.

To illustrate, Figure 1a shows a Mealy automaton defining an observation function. The input alphabet is the set $\Gamma = \{a, b\}$ of moves, and the output alphabet is the set $\{1, 2\}$ of observations. For example, the histories abb and bba map to the same observation sequence, namely 111, thus they belong to the same information set; the information partition on histories of length 2 is $\{aa, ab, bb\}, \{ba\}$.

This formalism captures the standard approach for describing information in finite-state systems (see, e.g., Reif [25], Lin and Wonham [18], Kupferman and Vardi [16], van der Meyden and Wilke [27]).

2.2.3 Indistinguishability

As a second alternative, we represent information partitions as equivalence relations between histories, such that the equivalence classes correspond to information sets. Intuitively, a player cannot distinguish between equivalent histories.

We say that an equivalence relation is an *indistinguishability* relation if the represented information partition satisfies the conditions of synchronous perfect recall. The following characterisation simply rephrases the relevant conditions for partitions in terms of equivalence relations.

▶ **Lemma 2.** *An equivalence relation $R \subseteq \Gamma^* \times \Gamma^*$ is an indistinguishability relation if, and only if, it satisfies the following properties:*

(1) *For every pair $(\tau, \tau') \in R$, the histories τ, τ' are of the same length.*

(2) *For every pair of histories $\tau, \tau' \in R$ of length ℓ, every pair (ρ, ρ') of histories of length $t \leq \ell$ that occur as prefixes of τ, τ', respectively, is also related by $(\rho, \rho') \in R$.*

As a finite-state representation, we will consider indistinguishability relations recognised by two-tape automata. To illustrate, Figure 1b shows a two-tape automaton that defines the same information partition as the Mealy automaton of Figure 1a. Here and throughout

the paper, the state q_{rej} represents a rejecting sink state. For example, the pair of words τ_1, τ_2 where $\tau_1 = abb$ and $\tau_2 = bba$ is accepted by the automaton (the state q_1 is accepting), meaning that the two words are indistinguishable.

Given a two-tape automaton $\mathcal{A} = (Q, \Gamma \times \Gamma, q_\varepsilon, \delta, F)$, the recognised relation $L(\mathcal{A})$ is, by definition, synchronous and hence satisfies condition (1) of Lemma 2. To decide whether $\mathcal{A}$ indeed represents an indistinguishability relation, we can use standard automata-theoretic techniques to verify that $L(\mathcal{A})$ is an equivalence relation, and that it satisfies the perfect-recall condition (2) of Lemma 2.

▶ **Lemma 3.** *The question whether a given two-tape automaton recognises an indistinguishability relation with perfect recall is decidable in polynomial (actually, cubic) time.*

The idea of using finite-state automata to describe information constraints of players in infinite games has been advanced in a series of work by Maubert and different coauthors [20, 21, 5, 8], with the aim of extending the classical framework of temporal logic and automata for perfect-information games to more expressive structures. In the general setup, the formalism features binary relations between histories that can be asynchronous and may not satisfy perfect recall. The setting of synchronous perfect recall is adressed as a particular case described by a one-state automaton that compares observation sequences rather than move histories. This allows to capture indistinguishability relations that actually correspond to regular observation functions in our setup.

Another approach of relating game histories via automata has been proposed recently by Fournier and Lhote [10]. The authors extend our framework to arbitrary synchronous relations, which are not necessarily prefix closed – and thus do not satisfy perfect recall.

2.2.4 Equivalent representations

In general, any partition of a set X can be represented either as an equivalence relation on X – equating the elements of each part – or as a (complete) invariant function, that is a function $f\colon X \to Z$ such that $f(x) = f(y)$ if, and only if, x, y belong to the same part. Thus equivalence relations and invariant functions represent different faces of the same mathematical object. The correspondence is witnessed by the following canonical maps.

For every function $f\colon X \to Z$, the *kernel* relation $\ker f := \{(x, y) \in X \times X \mid f(x) = f(y)\}$ is an equivalence. Given an equivalence relation $\sim \subseteq X \times X$, the *quotient map* $[\,\cdot\,]_\sim\colon X \to 2^X$, which sends each element $x \in X$ to its equivalence class $[x]_\sim := \{y \in X \mid y \sim x\}$, is a complete invariant function for $\sim$. Notice that the kernel of the quotient map is just $\sim$.

For the case of information partitions with synchronous perfect recall, the above correspondence relates indistinguishability relations and observation-history functions.

▶ **Lemma 4.** *If $\beta\colon \Gamma^* \to \Sigma$ is an observation function, then $\ker \hat{\beta}$ is an indistinguishability relation that describes the same information partition. Conversely, if $\sim$ is an indistinguishability relation, then the quotient map is an observation function that describes the same information partition.*

Accordingly, every information partition given by an indistinguishability relation can be alternatively represented by an observation function, and vice versa. However, if we restrict to finite-state representations, the correspondence might not be preserved. In particular, as the quotient map of any indistinguishability relation on Γ^* has infinite range (histories of different length are always distinguishable), it is not definable by a Mealy automaton, which has finite output alphabet.

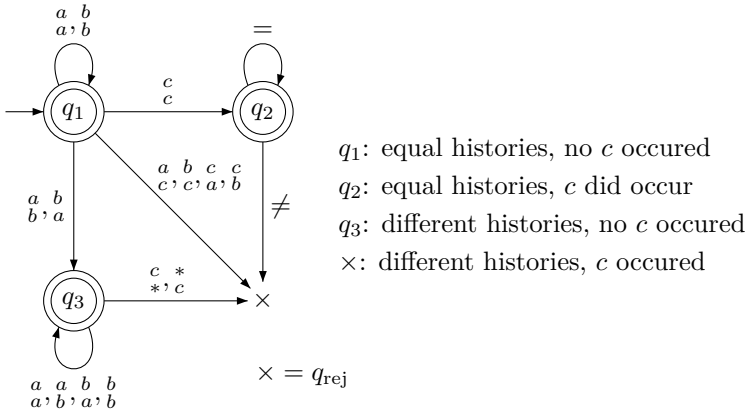


Figure 2 A two-tape DFA defining an indistinguishability relation that does not correspond to any regular observation function (the symbol $=$ stands for $\{\binom{a}{a}, \binom{b}{b}, \binom{c}{c}\}$, the symbol $\neq$ stands for $\{\binom{x}{y} \in \Gamma \times \Gamma \mid x \neq y\}$, and the symbol $*$ stands for $\{a, b, c\}$).

3 Observation is Weaker than Distinction

Firstly, we shall see that for every regular observation function the corresponding indistinguishability relation is also regular.

▶ **Proposition 5.** *For every observation function β given by a Mealy automaton of size m, we can construct a two-tape DFA of size $O(m^2)$ that defines the corresponding indistinguishability relation* $\ker \hat{\beta}$.

Proof. To construct such a two-tape automaton, we run the given Mealy automaton on the two input tapes simultaneously, and send it into a rejecting sink state whenever the observation output on the first tape differs from the output on the second tape. Accordingly, the automaton accepts a pair $(\tau, \tau') \in (\Gamma \times \Gamma)^*$ of histories, if and only if, their observation histories agree $\hat{\beta}(\tau) = \hat{\beta}(\tau')$. ◀

The statement of Proposition 5 is illustrated in Figure 1 where the structure of the two-tape DFA of Figure 1b is obtained as a product of two copies of the Mealy automaton in Figure 1a, where $q_1 = (p_1, p_1)$, $q_2 = (p_2, p_2)$, $q_3 = (p_1, p_2)$, and $q_4 = (p_2, p_1)$.

For the converse direction, however, the model of imperfect information described by regular indistinguishability relations is strictly more expressive than the one based on regular observation functions.

▶ **Lemma 6.** *There exists a regular indistinguishability relation that does not correspond to any regular observation function.*

Proof. As a simple example, consider a move alphabet with three letters $\Gamma := \{a, b, c\}$, and let $\sim \in \Gamma^* \times \Gamma^*$ relate two histories τ, τ' whenever they are equal or none of them contains the letter c. This is an indistinguishability relation, and it is recognised by the two-tape automaton of Figure 2.

We argue that the induced information tree has unbounded branching. All histories of the same length n that do not contain c are indistinguishable, hence $U_n = \{a, b\}^n$ is an information set. However, for every history $w \in U_n$ the history wc forms a singleton information set. Therefore U_n has at least 2^n successors, for every n.

However, for any observation function, the degree of the induced information tree is bounded by the size of the observation alphabet. Hence, the information partition described by $\sim$ cannot be represented by an observation function of finite range and so, a fortiori, not by any regular observation function. ◀

4 Which Distinctions Correspond to Observations

We have just seen, as a necessary condition for an indistinguishability relation to be representable by a regular observation function, that the information tree needs to be of bounded branching. In the following, we show that this condition is actually sufficient.

▶ **Theorem 7.** *Let Γ be a finite set of moves. A regular indistinguishability relation $\sim$ admits a representation as a regular observation function if, and only if, the information tree $\Gamma^*/_\sim$ is of bounded branching.*

Proof. The *only-if*-direction is immediate. If for an indistinguishability relation $\sim$, there exists an observation function $\beta\colon \Gamma^+ \to \Sigma$ with finite range (not necessarily regular) such that $\sim = \ker \hat{\beta}$, then the maximal degree of the information tree $\Gamma^*/_\sim$ is at most $|\Sigma|$. Indeed, the observation-history function $\hat{\beta}$ is a strong homomorphism from the move tree Γ^* to the tree of observation histories $\hat{\beta}(\Gamma^*) \subseteq \Sigma^*$: it maps every pair $(\tau, \tau c)$ of successive move histories to the pair of successive observation histories $(\hat{\beta}(\tau), \hat{\beta}(\tau)\beta(\tau c))$, and conversely, for every pair of successive observation histories, there exists a pair of successive move histories that map to it. By the Homomorphism Theorem (in the general formulation of Mal'cev [19]), it follows that the information tree $\Gamma^*/_\sim = \Gamma^*/_{\ker \hat{\beta}}$ is isomorphic to the image $\hat{\beta}(\Gamma^*)$, which, as a subtree Σ^*, has degree at most $|\Sigma|$.

To verify the *if*-direction, consider an indistinguishability relation $\sim$ over Γ^*, given by a DFA $\mathcal{R}$, such that the information tree $\Gamma^*/_\sim$ has branching degree at most $n \in \mathbb{N}$.

Let us fix an arbitrary linear ordering $\preceq$ of Γ. First, we pick as a representative for each information set, its least element with respect to the lexicographical order $<_{\text{lex}}$ induced by $\preceq$. Then, we order the information sets in $\Gamma^*/_\sim$ according to the lexicographical order of their representatives. Next, we define the *rank* of any nonempty history $\tau c \in \Gamma^*$ to be the index of its information set $[\tau c]_\sim$ in this order, restricted to successors of $[\tau]_\sim$ – this index is bounded by n. Let us consider the observation function β that associates to every history its rank. We claim that (1) it describes the same information partition as $\sim$ and (2) it is a regular function.

To prove the first claim, we show that whenever two histories are indistinguishable $\tau \sim \tau'$, they yield the same observation sequence $\hat{\beta}(\tau) = \hat{\beta}(\tau')$. The rank of a history is determined by its information set. Since $\tau \sim \tau'$, every pair (ρ, ρ') of prefix histories of the same length are also indistinguishable, and therefore yield the same rank $\beta(\rho) = \beta(\rho')$. By definition of $\hat{\beta}$, it follows that $\hat{\beta}(\tau) = \hat{\beta}(\tau')$. Conversely, to verify that $\hat{\beta}(\tau) = \hat{\beta}(\tau')$ implies $\tau \sim \tau'$, we proceed by induction on the length of histories. The basis concerns only the empty history and thus holds trivially. For the induction step, suppose $\hat{\beta}(\tau c) = \hat{\beta}(\tau' c')$. By definition of $\hat{\beta}$, we have in particular $\hat{\beta}(\tau) = \hat{\beta}(\tau')$, which by induction hypothesis implies $\tau \sim \tau'$. Hence, the information sets of the continuations τc and $\tau' c'$ are successors of the same information set $[\tau]_\sim = [\tau']_\sim$ in the information tree $\Gamma^*/\sim$. As we assumed that the histories τc and $\tau' c'$ have the same rank, it follows that they indeed belong to the same information set, that is $\tau c \sim \tau' c'$.

To verify the second claim on the regularity of the observation function β, we first notice that the following languages are regular:

- the (synchronous) lexicographical order $\{(\tau, \tau') \in (\Gamma \times \Gamma)^* \mid \tau \leq_{\text{lex}} \tau'\}$,
- the set of representatives $\{\tau \in \Gamma^* \mid \tau \leq_{\text{lex}} \tau' \text{ for all } \tau' \sim \tau\}$, and
- the representation relation $\{(\tau, \tau') \in \sim \mid \tau' \text{ is a representative}\}$.

Given automata recognising these languages, we can then construct, for each $k \leq n$, an automaton $\mathcal{A}_k$ that recognises the set of histories of rank at least k: together with the representative of the input history, guess the $k-1$ representatives that are below in the lexicographical order. Finally, we take the synchronous product of the automata $\mathcal{A}_1 \dots \mathcal{A}_k$ and equip it with an output function as follows: for every transition in the product automaton all components of the target state, up to some index k, are accepting – we define the output of the transition to be just this index k. This yields a Mealy automaton that outputs the rank of the input history, as desired. ◀

For further use, we estimate the size of the Mealy automaton defining the rank function as outlined in the proof. Suppose that an indistinguishability relation $\sim \subseteq (\Gamma \times \Gamma)^*$ given by a two-tape DFA $\mathcal{R}$ of size m gives rise to an information tree $\Gamma^*/_{L(\mathcal{R})}$ of degree n. The lexicographical order is recognisable by a two-tape DFA of size $O(|\Gamma|^2)$, bounded by $O(m)$; to recognise the set of representatives we take the product of this automaton with $\mathcal{R}$, and apply a projection and a complementation, obtaining a DFA of size bounded by $2^{O(m^2)}$); for the representation relation, we take a product of this automaton with $\mathcal{R}$ and obtain a two-tape DFA of size still bounded by $2^{O(m^2)}$. For every index $k \leq n$, the automaton $\mathcal{A}_k$ can be constructed via projection from a product of n such automata, hence its size bounded is by $2^{2^{O(nm^2)}}$. The Mealy automaton for defining the rank runs all these n automata synchronously, so it is of the same order of magnitude $2^{2^{O(nm^2)}}$.

To decide whether the information tree represented by a regular indistinguishability relation has bounded degree, we use a result from the theory of word-automatic structures [14, 4]. For the purpose of our presentation, we define an automatic presentation of a tree $T = (V, E)$ as a triple $(\mathcal{A}_V, \mathcal{A}_=, \mathcal{A}_E)$ of automata with input alphabet Γ, together with a surjective naming map $h\colon L \to V$ defined on a set of words $L \subseteq \Gamma^*$ such that

- $L(\mathcal{A}_V) = L$,
- $L(\mathcal{A}_=) = \ker h$, and
- $L(\mathcal{A}_E) = \{(u, v) \in L \times L \mid (h(u), h(v)) \in E\}$.

In this case, h is an isomorphism between $T = (V, E)$ and the quotient $(L, L(\mathcal{A}_E))/_{L(\mathcal{A}_=)}$. The size of such an automatic presentation is the added size of the three component automata. A tree is automatic if it has an automatic presentation.

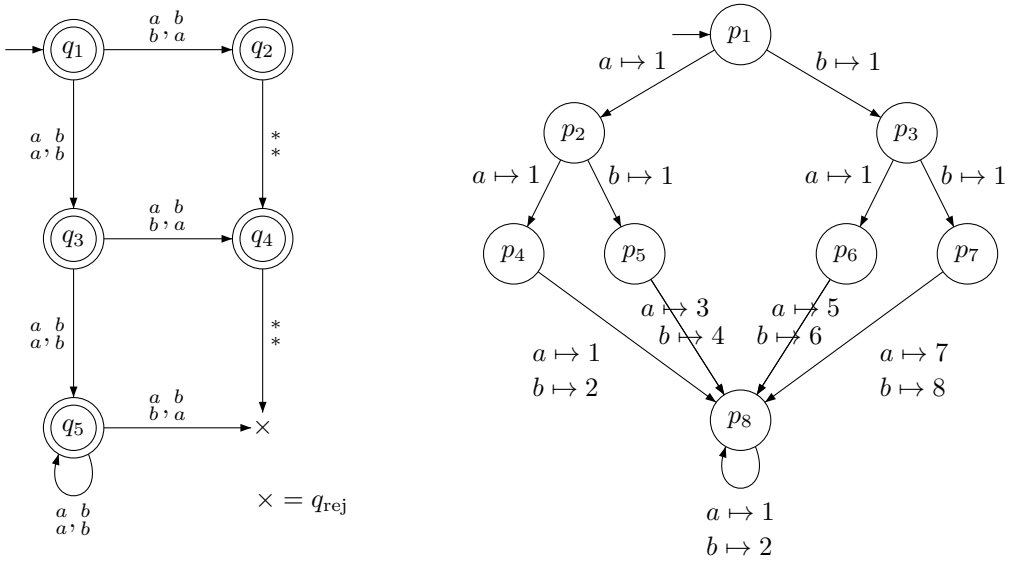
For an information partition given by a indistinguishability relation $\sim$ defined by a two-tape-DFA $\mathcal{R}$ on a move alphabet Γ, the information tree $\Gamma^*/_\sim$ admits an automatic presentation with the naming map that sends every history τ to its information set $[\tau]_\sim$, and

- as domain automaton A_V, the one-state automaton accepting all of Γ^* (of size Γ);
- as the equality automaton $\mathcal{A}_=$, the two-tape DFA $\mathcal{R}$, and
- for the edge relation, a two-tape DFA $\mathcal{A}_E$ that recognises the relation

$$\{(\tau, \tau' c) \in \Gamma^* \times \Gamma^* \mid (\tau, \tau') \in L(\mathcal{R})\}.$$

The latter automaton is obtained from $\mathcal{R}$ by adding transitions from each accepting state, with any move symbol on the first tape and the padding symbol on the second tape, to a unique fresh accepting state from which all outgoing transitions lead to the rejecting sink q_{rej}. Overall, the size of the presentation will thus be bounded by $O(|\mathcal{R}|)$.

Now, we can apply the following result of Kuske and Lohrey.



(a) Two-tape DFA. **(b)** Mealy automaton.

Figure 3 A synchronous two-tape automaton with $2k$ states (here $k = 3$) for which an equivalent observation Mealy automaton requires exponential number of states (2^k).

▶ **Proposition 8** ([17, Propositions 2.14–2.15]). *The question whether an automatic structure has bounded degree is decidable in exponential time. If the degree of an automatic structure is bounded, then it is bounded by* $2^{2^{m^{O(1)}}}$ *in the size* m *of the presentation.*

This allows to conclude that the criterion of Theorem 7 characterising regular indistinguishability relations that are representable by regular observation functions is effectively decidable. By following the construction for the rank function outlined in the proof of the theorem, we obtain a fourfold exponential upper bound for the size of a Mealy automaton defining an observation function.

▶ **Theorem 9.**

(i) *The question whether an indistinguishability relation given as a two-tape* DFA *admits a representation as a regular observation function is decidable in exponential time (with respect to the size of the* DFA*).*

(ii) *Whenever this is the case, we can construct a Mealy automaton of fourfold-exponential size and with at most doubly exponentially many output symbols that defines a corresponding observation function.*

5 Improving the Construction of Observation Automata

Theorem 9 establishes only a crude upper bound on the size of a Mealy automaton corresponding to a given indistinguishability DFA. In this section, we present a more detailed analysis that allows to improve the construction by one exponential.

Firstly, we point out that an exponential blowup is generally unavoidable, for the size of the automaton and for its observation alphabet.

▶ **Example 10.** Figure 3a shows a two-tape DFA that compares histories over a move alphabet $\{a, b\}$ with an embargo period of length k. Every pair of histories of length less than k is accepted, whereas history pairs of length k and onwards are rejected if, and only if, they are different. (The picture illustrates the case for $k = 3$). A Mealy automaton that describes this indistinguishability relation needs to produce, for every different prefix of length k, a different observation symbol. To do so, it has to store the first k symbols, which requires 2^k states and 2^k observation symbols (see Figure 3b). ◀

5.1 Structural properties of regular indistinguishability relations

For the following, let us fix a move alphabet Γ and a two-tape DFA $\mathcal{R} = (Q, \Gamma \times \Gamma, q_\varepsilon, \delta, F)$ defining an indistinguishability relation $L(\mathcal{R}) = \sim$. We assume that $\mathcal{R}$ is a minimal automaton in the usual sense that all states are reachable from the initial state, and the languages accepted from two different states are different. Let m be the size of $\mathcal{R}$. We usually write $\delta(q_\varepsilon, \frac{\tau}{\tau'})$ for $\delta(q, (\tau, \tau'))$.

First, we classify the states according to the behaviour of the automaton when reading the same input words on both tapes. On the one hand, we consider the states reachable from the initial state on such inputs, which we call *reflexive* states:

$$\mathsf{Ref} = \{q \in Q \mid \exists \tau \in \Gamma^* : \delta(q_\varepsilon, \tfrac{\tau}{\tau}) = q\}.$$

On the other hand, we consider the states from which it is possible to reach the rejecting sink by reading the same input word on both tapes, which we call *ambiguous* states,

$$\mathsf{Amb} = \{q \in Q \mid \exists \tau \in \Gamma^* : \delta(q, \tfrac{\tau}{\tau}) = q_{\mathrm{rej}}\}.$$

For instance, in the running example of Figure 1, the reflexive states are $\mathsf{Ref} = \{q_1, q_2\}$ and the ambiguous states are $\mathsf{Amb} = \{q_3, q_4, q_{\mathrm{rej}}\}$.

Since indistinguishability relations are reflexive, all the reflexive states are accepting and by reading any pair of identical words from a reflexive state, we always reach an accepting state. Therefore, a reflexive state cannot be ambiguous. Perhaps less obviously, the converse also holds: a non-reflexive state must be ambiguous.

▶ **Lemma 11** (Partition Lemma). $Q \setminus \mathsf{Ref} = \mathsf{Amb}$.

Proof. The inclusion $\mathsf{Amb} \subseteq Q \setminus \mathsf{Ref}$ (or, equivalently, that Amb and Ref are disjoint) follows from the definitions and the fact that $\sim$ is a reflexive relation, and thus $\delta(q_\varepsilon, \frac{\tau}{\tau}) \neq q_{\mathrm{rej}}$ for all histories τ.

To show that $Q \setminus \mathsf{Ref} \subseteq \mathsf{Amb}$, let us consider an arbitrary state $q \in Q \setminus \mathsf{Ref}$. By minimality of R, the state q is reachable from q_ε: there exist histories τ, τ' such that $\delta(q_\varepsilon, \frac{\tau}{\tau'}) = q$. Let $q_\tau = \delta(q_\varepsilon, \frac{\tau}{\tau})$ be the state reached after reading $\frac{\tau}{\tau}$ (see figure). Thus, $q_\tau \in \mathsf{Ref}$ and in particular $q_\tau \neq q$. Again by minimality of R, the languages accepted from q and q_τ are different. Hence, there exist histories π, π' such that $\frac{\pi}{\pi'}$ is accepted from q and rejected from q_τ, or the other way round. In the former case, we have that $\tau\pi \sim \tau'\pi'$ and $\tau\pi \not\sim \tau\pi'$, which by transitivity of $\sim$, implies $\tau\pi' \not\sim \tau'\pi'$. This means that from state q reading $\frac{\pi'}{\pi'}$ leads to q_{rej}, showing that $q \in \mathsf{Amb}$, which we wanted to prove. In the latter case, the argument is analogous. ◀

We say that a pair of histories accepted by $\mathcal{R}$ is *ambiguous*, if, upon reading them, the automaton $\mathcal{R}$ reaches an ambiguous state other than q_{rej}. Histories τ, τ' that form an ambiguous pair are thus indistinguishable, so they must map to the same observation.

However, there exists a suffix π such that the extensions $\tau \cdot \pi$ and $\tau' \cdot \pi$ become distinguishable. Therefore, any observation automaton for $\mathcal{R}$ has to reach two different states after reading τ and τ' since otherwise, the extensions by the suffix π would produce the same observation sequence, making $\tau \cdot \pi$ and $\tau' \cdot \pi$ wrongly indistinguishable. The argument generalises immediately to collections of more than two histories. We call a set of histories that are pairwise ambiguous an *ambiguous clique*.

We shall see later, in the proof of Lemma 15, that if the size of ambiguous cliques is unbounded, then the information tree $\Gamma^*/_{L(\mathcal{R})}$ has unbounded branching, and therefore there exists no Mealy automaton corresponding to $\mathcal{R}$. Now, we show conversely that whenever the size of the ambiguous cliques is bounded, we can construct such a Mealy automaton.

We say that two histories $\tau, \tau' \in \Gamma^*$ of the same length are *interchangeable*, denoted by $\tau \approx \tau'$, if $\delta(q_\varepsilon, \frac{\tau}{\pi}) = \delta(q_\varepsilon, \frac{\tau'}{\pi})$, for all $\pi \in \Gamma^*$. Note that $\approx$ is an equivalence relation and that $\tau \approx \tau'$ implies $\delta(q_\varepsilon, \frac{\tau}{\tau'}) \in \mathsf{Ref}$. The converse also holds.

▶ **Lemma 12.** *For all histories $\tau, \tau' \in \Gamma^*$, we have $\delta(q_\varepsilon, \frac{\tau}{\tau'}) \in \mathsf{Ref}$ if, and only if, $\tau \approx \tau'$.*

Proof. One direction, that $\tau \approx \tau'$ implies $\delta(q_\varepsilon, \frac{\tau}{\tau'}) \in \mathsf{Ref}$), follows immediately from the definitions (take $\pi = \tau'$ in the definition of interchangeable histories).

For the reverse direction, let us suppose that $\delta(q_\varepsilon, \frac{\tau}{\tau'}) \in \mathsf{Ref}$. We will show that, for all histories τ'', the states $q_1 = \delta(q_\varepsilon, \frac{\tau}{\tau''})$ and $q_2 = \delta(q_\varepsilon, \frac{\tau'}{\tau''})$ accept the same language. Towards this, let π_1, π_2 be an arbitrary pair of histories such that $\frac{\pi_1}{\pi_2}$ is accepted from q_1. Then,

- $\tau\pi_1 \sim \tau'\pi_1$, because $\delta(q_\varepsilon, \frac{\tau}{\tau'}) \in \mathsf{Ref}$, and from a reflexive state reading $\frac{\pi_1}{\pi_1}$ does not lead to q_{rej} (by Lemma 11).
- $\tau\pi_1 \sim \tau''\pi_2$, because $\delta(q_\varepsilon, \frac{\tau}{\tau''}) = q_1$ and $\frac{\pi_1}{\pi_2}$ is accepted from q_1.

By transitivity of $\sim$, it follows that $\tau'\pi_1 \sim \tau''\pi_2$, hence $\frac{\pi_1}{\pi_2}$ is accepted from $q_2 = \delta(q_\varepsilon, \frac{\tau'}{\tau''})$. Accordingly, the language accepted from q_1 is included in the language accepted from q_2; the converse inclusion holds by a symmetric argument. Since the states q_1 and q_2 accept the same languages, and because the automaton $\mathcal{R}$ is minimal, it follows that $q_1 = q_2$, which means that τ and τ' are interchangeable. ◀

According to Lemma 12 and because $q_{\mathrm{rej}} \notin \mathsf{Ref}$, all pairs of interchangeable histories are also indistinguishable. In other words, the interchangeability relation $\approx$ refines the indistinguishability relation $\sim$, and thus $[\tau]_\approx \subseteq [\tau]_\sim$ for all histories $\tau \in \Gamma^*$. In the running example (Figure 1), the sets $\{aa, ab, bb\}$ and $\{ba\}$ are $\sim$-equivalence classes, and the sets $\{aa, bb\}$, $\{ab\}$, and $\{ba\}$ are $\approx$-equivalence classes.

Let us lift the lexicographical order $\leq_{\mathsf{lex}}$ to sets of histories of the same length by comparing the smallest word of each set: we write $S \leq S'$ if $\min S \leq_{\mathsf{lex}} \min S'$. This allows us to rank the $\approx$-equivalence classes contained in a $\sim$-equivalence class, in increasing order. In the running example, if we consider the $\sim$-equivalence class $\{aa, ab, bb\}$, $\{aa, bb\}$ gets rank 1, and $\{ab\}$ gets rank 2 because $\{aa, bb\} \leq \{ab\}$. On the other hand, the $\sim$-equivalence class $\{ba\}$, as a singleton, gets rank 1.

Now, we denote by $\mathsf{idx}(\tau)$ the rank of the $\approx$-equivalence class containing τ. For example, $\mathsf{idx}(bb) = 1$ and $\mathsf{idx}(ab) = 2$. Further, we denote by $\mathsf{mat}(\tau)$ the square matrix of dimension $n = \max_{\tau' \in [\tau]_\sim} \mathsf{idx}(\tau')$ where we associate to each coordinate $i = 1, \ldots, n$ the i-th $\approx$-equivalence class C_i contained in $[\tau]_\sim$. The (i, j)-entry of $\mathsf{mat}(\tau)$ is the state $q_{ij} = \delta(q_\varepsilon, \frac{\tau_i}{\tau_j})$ where $\tau_i \in C_i$ and $\tau_j \in C_j$. Thanks to interchangeability, the state q_{ij} is well defined being independent of the choice of τ_i and τ_j.

It is easy to see that diagonal entries in such matrices are reflexive states (Lemma 12). We can show conversely that non-diagonal entries are ambiguous states.

▶ **Lemma 13.** *For all histories τ, the non-diagonal entries in* $\mathsf{mat}(\tau)$ *are ambiguous states.*

Proof. Non-diagonal entries in $\mathsf{mat}(\tau)$ correspond to pair of histories that are not $\approx$-equivalent, therefore those entries are not reflexive states (Lemma 12), hence they must be ambiguous states (Lemma 11). ◀

Finally, we can define a successor operation on matrix-index pairs and moves to obtain a homomorphic image of Γ^*.

▶ **Lemma 14.** *For every move $c \in \Gamma$, we can define a function* succ_c *such that for all histories $\tau \in \Gamma^*$, if $(M, i) = (\mathsf{mat}(\tau), \mathsf{idx}(\tau))$, then $\mathsf{succ}_c(M, i) = (\mathsf{mat}(\tau c), \mathsf{idx}(\tau c))$.*

5.2 Construction

For the remainder of the paper, let us assume that the branching degree of the information tree $\Gamma^*/_{L(\mathcal{R})}$ is bounded.

We define a Mealy automaton $\mathcal{F} = (P, \Gamma, \Sigma, p_\varepsilon, \delta, \lambda)$ over the input alphabet Γ and an output alphabet Σ in two phases: first, we define the semi-automaton $\mathcal{F}_0 = (P, \Gamma, p_\varepsilon, \delta)$ and then we construct the output alphabet Σ and the output function λ. To define the semi-automaton $\mathcal{F}_0$, we set:

- $P := \{(M, i) \mid M = \mathsf{mat}(\tau)$ and $i = \mathsf{idx}(\tau)$ for some history $\tau\}$,
- $p_\varepsilon := (q_\varepsilon, 1)$,
- for every state $(M, i) \in P$ and every move $c \in \Gamma$, let $\delta((M, i), c) = \mathsf{succ}_c(M, i)$.

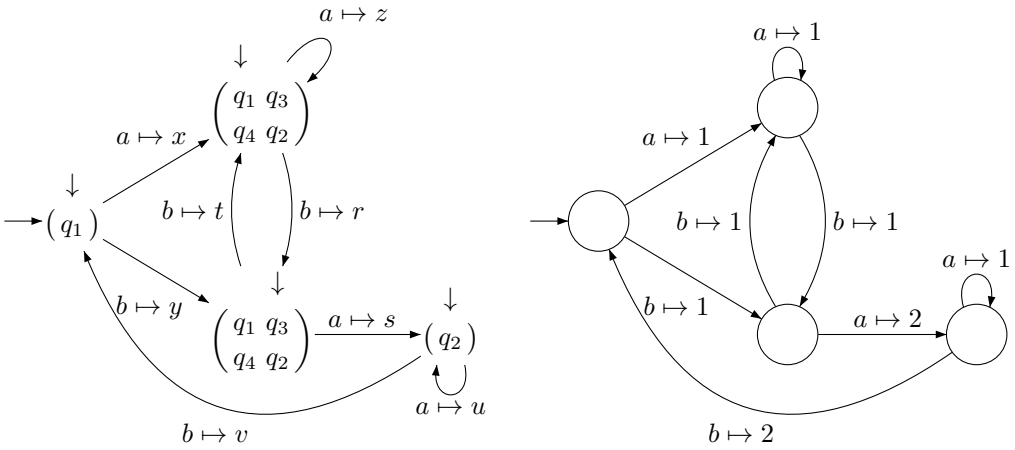
The construction of the Mealy automaton for the two-tape DFA of Figure 1b is shown in Figure 4a. The variables x, y, z, r, s, t, u, v represent the observation values of the output function. We determine the value of the variables by considering pairs of histories in the automaton, and in the Mealy automaton. For example, for $\tau = a$ and $\tau' = b$, we have $\tau \sim \tau'$ (according to the DFA), and therefore we derive the constraint $x = y$ in the Mealy automaton. We can show that the constraints are satisfiable and that every satisfying assignment describes an output function $\lambda\colon P \times \Gamma \to \Sigma$ such that $(P, \Gamma, \Sigma, p_\varepsilon, \delta, \lambda)$ is an observation automaton equivalent to the DFA (see Figure 4b for the running example).

According to Lemma 14, the state space P is the closure of $\{p_\varepsilon\}$ under the c-successor operation, for all $c \in \Gamma$. It remains to show that P is finite. The key is to bound the dimension of the largest matrix in P, which is the size of the largest ambiguous clique.

▶ **Lemma 15.** *If the branching degree of the information tree $\Gamma^*/_{L(\mathcal{R})}$ is bounded, then the largest ambiguous clique contains at most a doubly-exponential number of histories (with respect to the size of $\mathcal{R}$).*

Proof. First we show by contradiction that the size of the ambiguous cliques is bounded. Since the number of ambiguous states in $\mathcal{R}$ is finite, if there exists an arbitrarily large ambiguous clique, then by Ramsey's theorem [24], there exists an arbitrarily large set $\{\tau_1, \tau_2, \ldots, \tau_k\}$ of histories and a state $q \in \mathsf{Amb} \setminus \{q_{\mathrm{rej}}\}$ such that $\delta(q_\varepsilon, \frac{\tau_i}{\tau_j}) = q$ for all $1 \leq i < j \leq k$. By definition of Amb, there exists a nonempty history τc such that $\delta(q, \frac{\tau c}{\tau c}) = q_{\mathrm{rej}}$. Consider such a history τc of minimal length. The histories $\tau_i \tau$ $(i = 1, \ldots, k)$ are in the same $\sim$-equivalence class, but the equivalence classes $[\tau_i \tau c]_\sim$ are pairwise distinct. Therefore, the number of successors of $[\tau_i \tau]_\sim$ is at least k, thus arbitrarily large, in contradiction with the assumption that the branching degree the information tree $\Gamma^*/_{L(\mathcal{R})}$ is bounded.

Note that the size of the largest ambiguous clique corresponds to the maximum number of $\approx$-equivalence classes contained in an $\sim$-equivalence class (Lemma 13). We show that this number is at most doubly-exponential. Similarly to the proof of Theorem 7, we notice that the



(a) Transition structure. **(b)** Instantiated observations.

Figure 4 Construction of the Mealy automaton from the two-tape DFA of Figure 1b.

set of $\approx$-*representatives* defined by $\{\tau \in \Gamma^* \mid \tau \leq_{\text{lex}} \tau' \text{ for all } \tau' \approx \tau\}$ is regular, and therefore the representation relation $\{(\tau, \tau') \in \sim \mid \tau' \text{ is a } \approx\text{-representative}\}$ is also regular. Using a result of Weber [29, Theorem 2.1], there is a bound on the number of $\approx$-representatives that a history can have that is exponential in the size ℓ of the two-tape DFA recognising the representation relation, namely $O(\ell)^\ell$, and ℓ is bounded by $2^{O(m^2)}$ by the same argument as in the proof of Theorem 7 (where m is the size of $\mathcal{R}$). This provides a doubly-exponential bound $2^{2^{O(m^2)}}$ on the size of the ambiguous cliques. ◀

According to Lemma 15, the dimension k of the largest matrix in P is at most doubly exponential in $|\mathcal{R}|$. The number of matrices of a fixed dimension d is at most $|Q|^{d^2}$. Overall the number of matrices that appear in P is therefore bounded by $k \cdot |Q|^{k^2}$, and as the index is at most k, it follows that the number of states in P is bounded by $k^2 \cdot |Q|^{k^2}$, that is exponential in k and triply exponential in the size of $\mathcal{R}$.

▶ **Theorem 16.** *For every indistinguishability relation given by a two-tape DFA $\mathcal{R}$ such that the information tree $\Gamma^*/_{L(\mathcal{R})}$ is of bounded branching, we can construct a Mealy automaton of size triply exponential (with respect to the size of $\mathcal{R}$) that defines a corresponding observation function.*

6 Conclusion

The question of how to model information in infinite games is fundamental to defining their strategy space. As the decisions of each player are based on the available information, strategies are functions from information sets to actions. Accordingly, the information structure of a player in a game defines the support of her strategy space.

The assumption of synchronous perfect recall gives rise to trees as information structures (Lemma 1). In the case of observation functions with a finite range Σ, these trees are subtrees of the complete $|\Sigma|$-branching tree Σ^* – on which ω-tree automata can work (see [26, 13] for surveys on such techniques). Concretely, every strategy based on observations can be represented as a labelling of the tree Σ^* with actions; the set of all strategies for a given game forms a regular (that is, automata-recognisable) set of trees. Moreover, when considering winning conditions that are also regular, Rabin's Theorem [23] allows to conclude that winning

strategies also form a regular set. Indeed, we can construct effectively a tree automaton that recognises the set of strategies – for an individual player – that enforce a regular condition and, if this set is non-empty, we can also synthesise a Mealy automaton that defines one of these strategies. In summary, the interpretation of strategies as observation-directed trees allows us to search the set of all strategies systematically for winning ones using tree-automatic methods.

In contrast, when setting out with indistinguishability relations, we obtain more complicated tree structures that do not offer a direct grip to classical tree-automata techniques. As the example of Lemma 6 shows, there are cases where the information tree of a game is not regular, and so the set of all strategies is not recognisable by a tree automaton. Accordingly, the automata-theoretic approach to strategy synthesis via Rabin's Theorem cannot be applied to solve, for instance, the basic problem of constructing a finite-state strategy for one player to enforce a given regular winning condition.

On the other hand, modelling information with indistinguishability relations allows for significantly more expressiveness than observation functions. This covers notably settings where a player can receive an unbounded amount of information in one round. For instance, models with causal memory where one player may communicate his entire observation history to another player in one round can be captured with regular indistinguishability relation, but not with observation functions of any finite range. Even when an information partition that can be represented by finite-state observation functions, the representation by an indistinguishability relation may be considerably more succinct. For instance, a player that observes the move history perfectly, but with a delay of d rounds can be described by a two-tape DFA with $O(d)$ many states, whereas any Mealy automaton would require exponentially more states to define the corresponding observation function.

At the bottom line, as a finite-state model of information, indistinguishability relations are strictly more expressive and can be (at least exponentially) more succinct than observation functions. In exchange, the observation-based model is directly accessible to automata-theoretic methods, whereas the indistinguishability-based model is not. Our result in Theorem 9 allows to identify effectively the instances of indistinguishability relations for which this gap can be bridged. That is, we may take advantage of the expressiveness and succinctness of indistinguishability relations to describe a game problem and use the procedure to obtain, whenever possible, a reformulation in terms of observation functions towards solving the initial problem with automata-theoretic methods.

This initial study opens several exciting research directions. One immediate question is whether the fundamental finite-state methods on strategy synthesis for games with imperfect information can be extended from the observation-based model to the one based on indistinguishability relations. Is it decidable, given a game for one player with a regular winning condition against Nature, whether there exist a winning strategy? Can the set of all winning strategies be described by finite-state automata? In case this set is non-empty, does it contain a strategy defined by a finite-state automaton?

Another, more technical, question concerns the automata-theoretic foundations of games. The standard models are laid out for representations of games and strategies as trees of a fixed branching degree. How can these automata models be extended to trees with unbounded branching towards capturing strategies constrained by indistinguishability relations? Likewise, the automatic structures that arise as information quotients of indistinguishability relations form a particular class of trees, where both the successor and the descendant relation (that is, the transitive closure) are regular. On the one hand, this particularity may allow to decide properties about games (viz. their information trees) that are undecidable when considering general automatic trees, notably regarding bisimulation or other forms of game equivalence.

Finally, in a more application-oriented perspective, it will be worthwhile to explore indistinguishability relations as a model for games where players can communicate via messages of arbitrary length. In particular this will allow to extend the framework of infinite games on finite graphs to systems with causal memory considered in the area of distributed computing.

References

1 Michael Bacharach. Some extensions of a claim of Aumann in an axiomatic model of knowledge. *Journal of Economic Theory*, 37(1):167–190, 1985. doi:10.1016/0022-0531(85)90035-3.

2 Dietmar Berwanger and Laurent Doyen. Observation and distinction. Representing information in infinite games. *CoRR*, abs/1809.05978, 2018. arXiv:1809.05978.

3 Dietmar Berwanger, Lukasz Kaiser, and Bernd Puchala. A perfect-information construction for coordination in games. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011)*, volume 13 of *LIPIcs*, pages 387–398. Leibniz-Zentrum fuer Informatik, 2011. doi:10.4230/LIPIcs.FSTTCS.2011.387.

4 Achim Blumensath and Erich Grädel. Automatic structures. In *Logic in Computer Science (LICS 2000)*, pages 51–62. IEEE Comput. Soc, 2000. doi:10.1109/LICS.2000.855755.

5 Laura Bozzelli, Bastien Maubert, and Sophie Pinchinat. Uniform strategies, rational relations and jumping automata. *Information and Computation*, 242:80–107, June 2015. doi:10.1016/j.ic.2015.03.012.

6 Julius R. Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969. doi:10.2307/1994916.

7 Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, and Jean-Francois Raskin. Algorithms for omega-regular games with imperfect information. *Logical Methods in Computer Science*, Volume 3, Issue 3, 2007. doi:10.2168/LMCS-3(3:4)2007.

8 Catalin Dima, Bastien Maubert, and Sophie Pinchinat. Relating Paths in Transition Systems: The Fall of the Modal Mu-Calculus. *ACM Transactions on Computational Logic (TOCL)*, 19(3):23:1–23:33, September 2018. doi:10.1145/3231596.

9 Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about knowledge*. MIT Press, Cambridge, Mass., 2003.

10 Paulin Fournier and Nathan Lhote. Equivalence kernels of sequential functions and sequential observation synthesis. *CoRR*, abs/1910.06019, 2019. arXiv:1910.06019.

11 John Geanakoplos. Common Knowledge. *Journal of Economic Perspectives*, 6(4):53–82, 1992. doi:10.1257/jep.6.4.53.

12 Olivier Gossner and Tristan Tomala. Repeated games with complete information. In Robert A. Meyers, editor, *Encyclopedia of Complexity and Systems Science*, pages 7616–7630. Springer New York, New York, NY, 2009. doi:10.1007/978-0-387-30440-3_451.

13 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, logics, and infinite games*. Number 2500 in Lecture notes in computer science. Springer, 2002.

14 Bakhadyr Khoussainov and Anil Nerode. Automatic presentations of structures. In Gerhard Goos, Juris Hartmanis, Jan Leeuwen, and Daniel Leivant, editors, *Logic and Computational Complexity*, volume 960, pages 367–392. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995. doi:10.1007/3-540-60178-3_93.

15 Harold W. Kuhn. Extensive games and the problem of information, Contributions to the theory of games II. *Annals of Mathematics Studies*, 28:193–216, 1953.

16 Orna Kupferman and Moshe Y. Vardi. Synthesis with Incomplete Informatio. In Howard Barringer, Michael Fisher, Dov Gabbay, and Graham Gough, editors, *Advances in Temporal Logic*, Applied Logic Series, pages 109–127. Springer Netherlands, Dordrecht, 2000. doi:10.1007/978-94-015-9586-5_6.

17 Dietrich Kuske and Markus Lohrey. Automatic structures of bounded degree revisited. *The Journal of Symbolic Logic*, 76(04):1352–1380, 2011. doi:10.2178/jsl/1318338854.

18 F. Lin and W. M. Wonham. On observability of discrete-event systems. *Information Sciences*, 44(3):173–198, April 1988. doi:10.1016/0020-0255(88)90001-1.

19 Anatoly I. Mal'cev. *Algebraic Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1973. doi:10.1007/978-3-642-65374-2.

20 Bastien Maubert. *Logical foundations of games with imperfect information : uniform strategies. (Fondations logiques des jeux à information imparfaite : stratégies uniformes)*. PhD thesis, University of Rennes 1, France, 2014. URL: https://tel.archives-ouvertes.fr/tel-00980490.

21 Bastien Maubert and Sophie Pinchinat. A General Notion of Uniform Strategies. *International Game Theory Review*, 16(01):1440004, March 2014. doi:10.1142/S0219198914400040.

22 Boleslaw Mikolajczak. *Algebraic and structural automata theory*. Annals of Discrete Mathematics. North-Holland, Amsterdam, 1991.

23 Michael Oser Rabin. *Automata on Infinite Objects and Church's Problem*. American Mathematical Society, Boston, MA, USA, 1972.

24 Frank P. Ramsey. On a problem in formal logic. *Proc. London Math. Soc.*, 30:264–286, 1930.

25 John H. Reif. The complexity of two-player games of incomplete information. *Journal of Computer and System Sciences*, 29(2):274–301, 1984. doi:10.1016/0022-0000(84)90034-5.

26 Wolfgang Thomas. On the synthesis of strategies in infinite games. In *Symposium on Theoretical Aspects of Computer Science (STACS 1995)*, volume 900, pages 1–13. Springer, 1995. doi:10.1007/3-540-59042-0_57.

27 Ron van der Meyden and Thomas Wilke. Synthesis of Distributed Systems from Knowledge-Based Specifications. In Martín Abadi and Luca de Alfaro, editors, *CONCUR 2005 – Concurrency Theory*, volume 3653, pages 562–576, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. doi:10.1007/11539452_42.

28 John von Neumann and Oskar Morgenstern. *Theory of games and economic behavior*. Princeton University Press, 1944.

29 Andreas Weber. On the valuedness of finite transducers. *Acta Inf.*, 27(8):749–780, 1990. doi:10.1007/BF00264285.

How Fast Can You Escape a Compact Polytope?

Julian D'Costa
Indian Institute of Science, Bangalore, India
Max Planck Institute for Software Systems, Saarland Informatics Campus, Germany
julianrdcosta@gmail.com

Engel Lefaucheux
Max Planck Institute for Software Systems, Saarland Informatics Campus, Germany
elefauch@mpi-sws.org

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

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

Abstract

The Continuous Polytope Escape Problem (CPEP) asks whether every trajectory of a linear differential equation initialised within a convex polytope eventually escapes the polytope. We provide a polynomial-time algorithm to decide CPEP for compact polytopes. We also establish a quantitative uniform upper bound on the time required for every trajectory to escape the given polytope. In addition, we establish iteration bounds for termination of discrete linear loops via reduction to the continuous case.

2012 ACM Subject Classification Theory of computation → Timed and hybrid models

Keywords and phrases Continuous linear dynamical systems, termination

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.49

Related Version A full version of the paper is available on arXiv, `https://arxiv.org/abs/2001.08200`.

Funding *Joël Ouaknine*: ERC grant AVS-ISS (648701) and DFG grant 389792660 as part of TRR 248 (see `https://perspicuous-computing.science`).
James Worrell: EPSRC Fellowship EP/N008197/1.

1 Introduction

In ambient space $\mathbb{R}^d$, a *continuous linear dynamical system* is a trajectory $\mathbf{x}(t)$, where t ranges over the non-negative reals, defined by a differential equation $\dot{\mathbf{x}}(t) = f(\mathbf{x}(t))$ in which the function f is *affine* or *linear*. If the initial point $\mathbf{x}(0)$ is given, the differential equation uniquely defines the entire trajectory. (Linear) dynamical systems have been extensively studied in Mathematics, Physics, and Engineering, and more recently have played an increasingly important role in Computer Science, notably in the modelling and analysis of cyber-physical systems; two recent and authoritative textbooks on the subject are [1, 12].

In the study of dynamical systems, particularly from the perspective of control theory, considerable attention has been given to the study of *invariant sets*, i.e., subsets of $\mathbb{R}^d$ from which no trajectory can escape; see, *e.g.*, [7, 4, 2, 13]. Our focus in the present paper is on sets with the dual property that *no trajectory remains trapped*. Such sets play a key role in analysing *liveness* properties in cyber-physical systems (see, for instance, [1]): discrete

progress is ensured by guaranteeing that all trajectories (i.e., from any initial starting point) must eventually reach a point at which they "escape" (temporarily or permanently) the set in question, thereby forcing a discrete transition to take place.

More precisely, given an affine function $f : \mathbb{R}^d \to \mathbb{R}^d$ and a convex polytope $\mathcal{P} \subseteq \mathbb{R}^d$, both specified using rational coefficients encoded in binary, we consider the *Continuous Polytope Escape Problem (CPEP)* which asks whether, for all starting points $\mathbf{x}_0$ in $\mathcal{P}$, the corresponding trajectory of the solution to the differential equation

$$\begin{cases} \dot{\mathbf{x}}(t) = f(\mathbf{x}(t)) \\ \mathbf{x}(0) = \mathbf{x}_0 \end{cases}$$

eventually escapes $\mathcal{P}$.[1]

CPEP was shown to be decidable in [11], in which an algorithm having complexity between **NP** and **PSPACE** was exhibited. It is worth noting that, when the polytope $\mathcal{P}$ is unbounded in space, the time taken for a given trajectory to escape may be unboundedly large. For example, consider the unbounded one-dimensional polytope $\mathcal{P} = \{x \in \mathbb{R} \mid x \geq 1\}$ and differential equation $\dot{x}(t) = -x(t)$. For any starting point x_0, the trajectory $x(t) = e^{-t}x_0$ converges to 0 and thus all trajectories eventually escape. However, the escape time is at least $\log(x_0)$ and hence is not bounded over all initial points in $\mathcal{P}$. Even if the polytope is bounded, there still need not be a uniform bound on the escape time. For example, consider the polytope $\mathcal{P} = (0, 1]$ and the equation $\dot{x}(t) = x(t)$. Given an initial point x_0, the trajectory $x(t) = e^t x_0$ necessarily escapes $\mathcal{P}$: but the escape time is at least $\log(1/x_0)$, which again is not bounded over $\mathcal{P}$.

Main contributions. We show that, for *compact* (i.e., closed and bounded) polytopes, CPEP is decidable in polynomial time. Moreover, we show how to calculate uniform escape-time upper bounds; these bounds are exponential in the bit size of the descriptions of the differential equation and of the polytope, and doubly exponential in the ambient dimension. In the case of differential equations specified by invertible or diagonalisable matrices, we have singly exponential bounds.

In comparing the above with the results from [11], we note both a substantial improvement in complexity (from **PSPACE** to **PTIME**) as well as the production of explicit uniform bounds on escape times. It is worth pointing out that the mathematical approach pursued in [11] is non-effective, and therefore does not appear capable of yielding any quantitative escape-time bounds. The new constructive techniques used in the present paper, which originate mainly from linear algebra and algebraic number theory, are applicable owing to the fact that we focus our attention on *compact* polytopes. In practice, of course, this is usually not a burdensome restriction; in most cyber-physical systems applications, for instance, all relevant polytopes will be compact (see, *e.g.*, [1]).

Another interesting observation is that the seemingly closely related question of whether a given single trajectory of a linear dynamical system escapes a compact polytope appears to be vastly more challenging and is not known to be decidable; see, in particular, [3, 8, 9]. However, whether a given trajectory eventually hits a given single point is known as the *Continuous Orbit Problem* and can be decided in polynomial time [10].

[1] By "escaping" $\mathcal{P}$, we simply mean venturing outside of $\mathcal{P}$ – we are unconcerned whether the trajectory might re-enter $\mathcal{P}$ at a later time or not.

Finally, we also consider in the present paper a discrete analogue of CPEP for discrete-time linear dynamical systems, namely the *Discrete Polytope Escape Problem (DPEP)*. This consists in deciding, given an affine function $f : \mathbb{R}^d \to \mathbb{R}^d$ and a convex polytope $\mathcal{P} \subseteq \mathbb{R}^d$, whether for all initial points $\mathbf{x}_0 \in \mathcal{P}$, the sequence $(\mathbf{x}_n)_{n\in\mathbb{N}}$ defined by the initial point and the recurrence $\mathbf{x}_{n+1} = f(\mathbf{x}_n)$ eventually escapes $\mathcal{P}$. This problem – phrased as "termination of linear programs" over the reals and the rationals respectively – was already studied and shown decidable in the seminal papers [5, 15], albeit with no complexity bounds nor upper bounds on the number of iterations required to escape. By leveraging our results on CPEP, we are able to show that, for *compact* polytopes, DPEP is decidable in polynomial time, and moreover we derive upper bounds on the number of iterations that are singly exponential in the bit size of the problem description and doubly exponential in the ambient dimension.

2 Preliminaries

2.1 The Continuous Polytope Escape Problem

As noted in the previous section, the Continuous Polytope Escape Problem (CPEP) for continuous linear dynamical systems consists in deciding, given an affine function $f : \mathbb{R}^d \to \mathbb{R}^d$ and a convex polytope $\mathcal{P} \subseteq \mathbb{R}^d$, whether there exists an initial point $\mathbf{x}_0 \in \mathcal{P}$ for which the trajectory of the unique solution of the differential equation $\dot{\mathbf{x}}(t) = f(\mathbf{x}(t)), \mathbf{x}(0) = \mathbf{x}_0, t \geq 0$, is entirely contained in $\mathcal{P}$. For $T \in \mathbb{R} \cup \{\infty\}$, we denote by $X(T)$ the set $\{\mathbf{x}(t) \mid t \in \mathbb{R}_{\geq 0}, t \leq T\}$. A starting point $\mathbf{x}_0 \in \mathcal{P}$ is said to be a *fixed point* if for all $t \geq 0$, $\mathbf{x}(t) = \mathbf{x}_0$, and it is *trapped* if the trajectory of $\mathbf{x}(t)$ is contained in $\mathcal{P}$ (i.e., $X(\infty) \subseteq \mathcal{P}$); thus solving the CPEP amounts to deciding whether there is a trapped point.

We will represent a d-dimensional instance of the CPEP by a triple $(A, B, \mathbf{c})$, where $A \in \mathbb{R}^{d\times d}$ represents the linear function $f_A : \mathbf{x} \mapsto A\mathbf{x}$ [2] and $B \in \mathbb{R}^{n\times d}$, $\mathbf{c} \in \mathbb{R}^n$ represent the polytope $\mathcal{P}_{B,\mathbf{c}} = \{\mathbf{x} \in \mathbb{R}^d \mid B\mathbf{x} \leq \mathbf{c}\}$. Given such an instance and an initial point $\mathbf{x}_0$, the solution of the differential equation is $\mathbf{x}(t) = \exp(At)\mathbf{x}_0 \in \mathbb{R}^d$. For the computation of bounds, we assume that all the coefficients of A, B and $\mathbf{c}$ are rational and encoded in binary. The decidability results and escape bounds computed in this paper can be adapted to the case of algebraic coefficients, but we don't pursue this here.

Decidability of the CPEP was shown in [11]. In this paper we are interested in the following problem: given a positive instance of CPEP (i.e., one in which every trajectory escapes), compute an upper bound on the time to escape that holds uniformly over all initial points in the polytope. In other words, we wish to compute $T \in \mathbb{R}_{\geq 0}$ such that for all points $\mathbf{x}_0 \in \mathcal{P}$ there exists $t_0 \in \mathbb{R}$ such that $t_0 \leq T$ and $\mathbf{x}(t_0) \notin \mathcal{P}$. We call such a T an *escape-time bound*.

As noted in the Introduction, such an escape-time bound need not exist in general. In the remainder of this paper, we therefore restrict our attention to *compact* polytopes.

2.2 Jordan Normal Forms

Let $A \in \mathbb{Q}^{d\times d}$ be a square matrix with rational entries. The *minimal polynomial* of A is the unique monic polynomial $m(x) \in \mathbb{Q}[x]$ of least degree such that $m(A) = 0$. By the Cayley-Hamilton Theorem, the degree of m is at most the dimension of A. The set $\sigma(A)$ of eigenvalues of A is the set of roots of m. The *index* of an eigenvalue λ, denoted by $\nu(\lambda)$, is defined as its multiplicity as a root of m.

[2] We remark that by increasing the dimension by one, the general CPEP can be reduced to the homogeneous case, in which the function f is linear.

For each eigenvalue λ of A we denote by $\mathcal{V}_\lambda$ the subspace of $\mathbb{C}^d$ spanned by the set of generalised eigenvectors associated with λ. We also denote by $\mathcal{V}^r$ the subspace of $\mathbb{C}^d$ spanned by the set of generalised eigenvectors associated with some real eigenvalue; we likewise denote by $\mathcal{V}^c$ the subspace of $\mathbb{C}^d$ spanned by the set of generalised eigenvectors associated with some non-real eigenvalue.

It is well known that each vector $\mathbf{v} \in \mathbb{C}^d$ can be written uniquely as $\mathbf{v} = \sum_{\lambda \in \sigma(A)} \mathbf{v}_\lambda$, where $\mathbf{v}_\lambda \in \mathcal{V}_\lambda$. It follows that $\mathbf{v}$ can also be uniquely written as $\mathbf{v} = \mathbf{v}^r + \mathbf{v}^c$, where $\mathbf{v}^r \in \mathcal{V}^r$ and $\mathbf{v}^c \in \mathcal{V}^c$. Moreover, we can write any matrix A as $A = Q^{-1}JQ$ for some invertible matrix Q and block diagonal Jordan matrix $J = \operatorname{diag}(J_1, \ldots, J_N)$, with each block J_i, associated to the eigenvalue λ_i having the following form:

$$\begin{pmatrix} \lambda_i & 1 & 0 & \cdots & 0 \\ 0 & \lambda_i & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & \lambda_i \end{pmatrix}.$$

Given a rational matrix A, its Jordan Normal Form $J = QAQ^{-1}$ can be computed in polynomial time, as shown in [6]. Note that each vector $\mathbf{v}$ appearing as a column of the matrix Q^{-1} is a generalised eigenvector. We also note that the index $\nu(\lambda)$ of some eigenvalue λ corresponds to the dimension of the largest Jordan block associated with it. Given J_i, a Jordan block of size k associated with some eigenvalue λ, the closed-form expression for its exponential is

$$\exp(J_i t) = \exp(\lambda t) \begin{pmatrix} 1 & t & \cdots & \frac{t^{k-1}}{(k-1)!} \\ 0 & 1 & \cdots & \frac{t^{k-2}}{(k-2)!} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & t \\ 0 & 0 & \cdots & 1 \end{pmatrix}.$$

Using this, for all $j \leq d$, the closed form of the j-th component of a trajectory is, $x^{(j)}(t) = \sum_{\lambda \in \sigma(A)} p_\lambda(t) \exp(\lambda t)$ where for all $\lambda \in \sigma(A)$, p_λ is a polynomial of degree at most $\nu(\lambda) - 1$.

2.3 The Discrete Polytope Escape Problem

We shall also consider the Discrete Polytope Escape Problem (DPEP). The DPEP consists in deciding, given an affine function $f : \mathbb{R}^d \to \mathbb{R}^d$ and a convex polytope $\mathcal{P} \subseteq \mathbb{R}^d$, whether there exists an initial point $\mathbf{x}_0 \in \mathcal{P}$ for which the sequence $(\mathbf{x}_n)_{n \in \mathbb{N}}$ defined by the initial point and the recurrence $\mathbf{x}_{n+1} = f(\mathbf{x}_n)$ is entirely contained in $\mathcal{P}$. The definitions of fixed and trapped points are immediately transposed to the discrete setting by considering the sequence instead of the trajectory.

As with the CPEP, a d-dimensional instance of the DPEP is represented by a triple $(A, B, \mathbf{c})$, where $A \in \mathbb{R}^{d \times d}$ represents the function $f_A : \mathbf{x} \in \mathbb{R}^d \mapsto A\mathbf{x} \in \mathbb{R}^d$ and $B \in \mathbb{R}^{n \times d}$ and $\mathbf{c} \in \mathbb{R}^n$ represent the polytope $\mathcal{P}_{B,\mathbf{c}} = \{\mathbf{x} \in \mathbb{R}^d \mid B\mathbf{x} \leq \mathbf{c}\}$. Using the Jordan Normal form, one can see that the general form of the j-th component of the sequence $(\mathbf{x}_n)_{n \in \mathbb{N}}$ is $\mathbf{x}_n^{(j)} = \sum_{\lambda \in \sigma(A)} p_\lambda(n) \lambda^n$, where for all $\lambda \in \sigma(A)$, p_λ is a polynomial of degree at most $\nu(\lambda) - 1$. We assume that all the coefficients of A, B and $\mathbf{c}$ are rational.

The examples showing one cannot build a bound when the polytope is open or unbounded for the CPEP can easily be carried over to the DPEP. Thus, when considering the DPEP, we also only consider compact polytopes.

3 Deciding the Polytope Escape Problem for Compact Polytopes

While the result of [11] allows us to decide the existence of a trapped point for continuous linear dynamical systems, the method is quite involved. When restricting ourselves to compact polytopes, however, we can use the following proposition, which shows that the existence of a trapped point is equivalent to the existence of a fixed point.

▶ **Theorem 1.** *Given a CPEP instance $(A, B, \mathbf{c})$, the polytope $\mathcal{P}_{B,\mathbf{c}}$ contains a trapped point iff it contains a fixed point.*

Proof. For the "if" direction, observe that a fixed point $\mathbf{x}_0 \in \mathcal{P}_{B,\mathbf{c}}$ is necessarily trapped.

Conversely, assume that there exists a trapped point $\mathbf{x}_0 \in \mathcal{P}_{B,\mathbf{c}}$. Let H be the closure of the convex hull of $X(\infty) = \{\mathbf{x}(t) \mid t \in \mathbb{R}_{\geq 0}\}$. Then H is convex, compact, and is contained in $\mathcal{P}_{B,\mathbf{c}}$. For each $n \in \mathbb{N}$ we define a function $s_n : H \to H$ by $s_n(\mathbf{x}) = e^{A2^{-n}}\mathbf{x}$. Note that this function is well-defined: clearly $X(\infty)$ is invariant under s_n; moreover, since s_n is linear, the convex hull of $X(\infty)$ is also invariant under s_n; finally, since s_n is continuous, the closure of the convex hull of $X(\infty)$ (i.e., H) is invariant under s_n.

For all $n \in \mathbb{N}$, as the function s_n is continuous, by Brouwer's fixed-point theorem s_n admits at least one fixed point on H. Let F_n be the non-empty set of fixed points of s_n in H. Since $s_n = s_{n+1} \circ s_{n+1}$ we have that $F_{n+1} \subseteq F_n$ for all $n \in \mathbb{N}$. Moreover, by continuity of the function f_A, F_n is a closed set for all $n \in \mathbb{N}$. Therefore, the intersection $F_\infty = \bigcap_{n\in\mathbb{N}} F_n$ is non-empty. By continuity of f_A, any point $\mathbf{y} \in F_\infty$ satisfies $f_A(\mathbf{y}) = \mathbf{0}$. Therefore, the CPEP instance admits at least one fixed point within $\mathcal{P}_{B,\mathbf{c}}$, which concludes the proof. ◀

Since the set $F = \{\mathbf{x} \mid A\mathbf{x} = \mathbf{0}\}$ of fixed points is easy to calculate, we simply need to check whether its intersection with the polytope is empty in order to decide CPEP. Since the latter can be formulated as a linear program, we can decide CPEP for compact polytopes in polynomial time.

The proof of Theorem 1 carries over with very small changes (considering the function f_A directly, instead of the family $(s_n)_{n\in\mathbb{N}}$) to prove an analogous result for DPEP:

▶ **Theorem 2.** *Given a DPEP instance $(A, B, \mathbf{c})$, $\mathcal{P}_{B,\mathbf{c}}$ has a trapped point iff it contains a fixed point.*

4 Bounding the Escape Time for a Positive CPEP Instance

The goal of this section is to establish a uniform bound on the escape time of a positive CPEP instance. The main result is as follows:

▶ **Theorem 3.** *Given a d-dimensional positive instance of the CPEP, described by a tuple of bit size b, the time to escape the polytope is bounded by*

$$T = 4\exp\left(640bd^{4d+10}\right) = e^{bd^{O(d)}}.$$

We prove this bound in four steps. First, in Subsection 4.1, we show that one can ignore the component of the initial vector lying in the complex eigenspace $\mathcal{V}^c$ after a certain amount of time. Intuitively speaking, this stems from the fact that a convex polytope that contains

a spiral must contain the centre of that spiral. Thus whenever we have a complex eigenvalue we can ignore the effects of the rotation by focusing on the axis of the helix formed by the trajectory.

We could then try to find a bound on escape time by looking at positivity of expressions of the form $\mathbf{b}^T \exp(At)y_0$, where $\mathbf{b}$ is the normal to a hyperplane supporting a face of the polytope. Unfortunately, these expressions contain terms corresponding to many different eigenvalues, which significantly complicates the analysis. We get around this problem in Subsection 4.2 by bounding the distance of the polytope to the origin and to the set of fixed points of the differential equation using hypercubes in the Jordan basis. This allows us to disentangle the effects of the different eigenvalues. We prove that the trajectories of the system escape the enclosing hypercube, and use the escape time of the hypercube as an upper bound on the escape time of the polytope.

Our next step is then, in Subsection 4.3, to compute a uniform escape bound for our hypercube. Finally, Subsection 4.4 combines the results from the previous sections to get the desired bound on the escape time of the original polytope.

4.1 Removing the Complex Eigenvalues

Let $(A, B, \mathbf{c})$, be a positive CPEP instance. Assume for now that A is given in Jordan normal form. This assumption is not without cost as we will see in the next subsection. In this subsection, we consider a single block J_i of A corresponding to a non-real eigenvalue λ_i. Considering only the dimensions associated to the Jordan block J_i (i.e., the space $\mathcal{V}_\lambda$) and writing $k = \nu(\lambda_i)$, we have that given an initial point $\mathbf{x}_0 = [x^{(1)}, \dots, x^{(k)}]$, the components of the trajectory $\mathbf{x}(t)$ are

$$\begin{bmatrix} x^{(1)}(t) \\ x^{(2)}(t) \\ \vdots \\ x^{(k)}(t) \end{bmatrix} = \exp(\lambda_i t) \begin{bmatrix} x^{(1)} + x^{(2)}t + x^{(3)}t^2/2 + \cdots + x^{(k)}t^{k-1}/(k-1)! \\ x^{(2)} + x^{(3)}t + \cdots + x^{(k)}t^{k-2}/(k-2)! \\ \vdots \\ x^{(k)} \end{bmatrix}.$$

In order to compute the escape times in the presence of non-real eigenvalues we use the fact that if a convex set contains a spiralling or helical trajectory, it must contain the axis of that trajectory. A trajectory starting on this axis is not affected by the eigenvalue that generates the rotation, moreover, if the trajectory starting in the axis escapes, then the original trajectory also escapes (albeit, potentially a bit later). This allows us to reduce to the case where we only have real eigenvalues. The following lemma formalizes this intuition.

▶ **Lemma 4** (Zero in convex hull). *Let*

$$\mathbf{x}(t) = (p_{1,0}(t)e^{\lambda_1 t}, \dots, p_{1,\nu(\lambda_1)-1}(t)e^{\lambda_1 t}, \dots, p_{r,0}(t)e^{\lambda_r t}, \dots, p_{r,\nu(\lambda_r)-1}(t)e^{\lambda_r t})^T$$

be a trajectory where, for all j, $\lambda_j = \eta_j + i\theta_j$, θ_j is non-zero, and $p_{j,k}$ is the Taylor polynomial corresponding to the factor $e^{\lambda_j t}$ of degree k. Then there exists a time T such that $\mathrm{Conv}(X(T))$ *contains the origin (where* Conv *represents the convex hull). In particular, this T satisfies*

$$T \leq \sum_{j=1}^{r} \nu(\lambda_j)\frac{\pi}{\theta_j}.$$

Proof Sketch. The basic idea is to take an initial point parametrized by t, travel along the trajectory to the point of opposite phase for a particular component, and create a new point where this component is equal to 0 by adding together a suitable convex combination of

the opposite-phase point and the initial one. Since both these points were parametrized by t, we can take the trajectory starting in the newly created point (which lies in the convex hull of the original trajectory) and repeat for the other dimensions until every component corresponding to the $\mathcal{V}^c$ subspace is equal to 0. ◀

4.2 Replacing the Polytopes with Hypercubes

Let $(A, B, \mathbf{c})$ be a d-dimensional positive CPEP instance, $J \in \mathbb{R}^{d\times d}$ a matrix in Jordan normal form, and $Q \in \mathbb{R}^{d\times d}$ be such that $A = Q^{-1}JQ$.

Let us assume that all eigenvalues of A are real. Our approach is to work in the Jordan basis. To this end we note that the trajectory $\mathbf{x}(t) = \exp(At)\mathbf{x}_0$ escapes the polytope $\mathcal{P}_{B,\mathbf{c}}$ for all $\mathbf{x}_0 \in \mathbb{R}^d$ if and only if the trajectory $\mathbf{y}(t) = \exp(Jt)\mathbf{y}_0$ escapes the polytope $\mathcal{P}_{BQ^{-1},\mathbf{c}}$ for all $\mathbf{y}_0 \in \mathbb{R}^d$. (Note that all entries of Q^{-1} are real algebraic.) Below we analyse the latter version of CPEP, i.e., with a matrix J in Jordan form with real algebraic entries.

The key intuition is that for every initial vector $\mathbf{y}_0 \in \mathbb{R}^d$ the trajectory $\mathbf{y}(t) = \exp(Jt)\mathbf{y}_0$ will either converge to a fixed point of the system or otherwise will diverge to infinity in some component. In either case the trajectory must exit the polytope since the polytope is bounded and does not meet the set $F := \{\mathbf{y} \in \mathbb{R}^d \mid J\mathbf{y} = \mathbf{0}\}$ of fixed points. We are thus led to define constants $C, \varepsilon > 0$ such that every trajectory $\mathbf{y}(t) = \exp(Jt)\mathbf{y}_0$ that either exits the hypercube $[-C, C]^d$ or comes within distance ε of the set F of fixed points will necessarily have left the polytope $\mathcal{P}_{BQ^{-1},\mathbf{c}}$. More precisely, we seek $C > 0$ and $\varepsilon > 0$ such that:

1. $\mathcal{P}_{BQ^{-1},\mathbf{c}} \subseteq [-C, C]^d$,
2. For all $\mathbf{y} \in F$ the hypercube $\{\boldsymbol{y} + \boldsymbol{x} \mid \boldsymbol{x} \in [-\varepsilon, \varepsilon]^n\}$ does not meet $\mathcal{P}_{BQ^{-1},\mathbf{c}}$.

Note that such a positive ε must exist since, $\mathcal{P}_{BQ^{-1},\mathbf{c}} \cap F = \emptyset$, $\mathcal{P}_{BQ^{-1},\mathbf{c}}$ is compact, and F is closed. Having computed C and ε, we obtain the escape bound for the polytope $\mathcal{P}_{BQ^{-1},\mathbf{c}}$ by computing the time to either exit the hypercube in Item 1 or enter one of the hypercubes mentioned in Item 2.

In order to compute the escape bound, we only need the upper bound on the ratio C/ε given in the following lemma.

▶ **Lemma 5.** *Let $(A, B, \mathbf{c})$, be a d-dimensional positive CPEP instance involving rationals, each of at most $b \in \mathbb{N}$ bits. One can select $C \in \mathbb{R}$ and $\varepsilon > 0$ satisfying Conditions 1 and 2, above, and such that*

$$\frac{C}{\varepsilon} \leq \exp\left(640bd^{3d+8}\right).$$

Sketch of proof. The proof relies on Liouville's inequality, which states that the size of an algebraic number can be upper- and lower-bounded in terms of the degree and height (coefficient size) of its minimal integer polynomial, and an arithmetic complexity lemma which bounds the logarithmic height of the output of an arithmetic circuit in terms of the heights of the inputs. We apply these bounds to the vertices of the polytope in the Jordan basis (which are computed using the entries of B, $\mathbf{c}$ and Q^{-1}). ◀

Let us illustrate how the change of basis can lead to an exponential size polytope. Consider the matrix

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1.01 \end{bmatrix},$$

its associated Jordan decomposition

$$A = Q^{-1}JQ = \begin{bmatrix} 1 & 0 & 10000 \\ 0 & 1 & 100 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1.01 \end{bmatrix} \begin{bmatrix} 1 & 0 & -10000 \\ 0 & 1 & -100 \\ 0 & 0 & 1 \end{bmatrix}$$

and the polytope $\mathcal{P} = \{(0, 1, x_3) \in \mathbb{R}^3 \mid 0 \leq x_3 \leq 1\}$. This polytope is contained in the hypercube of size $C = 1$ and every point is at least at distance $\varepsilon = 1$ from any fixed point. However, in the Jordan basis, this polytope becomes equal to the set $\{(-10000x_3, 1 - 100x_3, x_3) \in \mathbb{R}^3 \mid (0, 1, x_3) \in \mathcal{P}\}$, which forces a choice of C and ε such that $\frac{C}{\varepsilon} \geq 10000$.

In general, using the same reasoning on the matrix of dimension d

$$A = \begin{bmatrix} 1 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 1 + 1/2^b \end{bmatrix},$$

leads to a blowup in the value for C/ε of $2^{b(d-1)}$, thus exponential in the dimension.

The bound obtained in Lemma 5 is however doubly exponential in the dimension. Analysing the proof of the lemma, in order to obtain an example for which the bound is tight, one would need to build a family of polynomials with splitting fields of degree exponential in the degree of the polynomial. Such polynomials unfortunately seem hard to find.

4.3 Computing an Upper Bound on the Escape Time for each Eigenspace

Consider a real eigenvalue λ of the Jordan matrix J associated with a Jordan block of size k. Let $\mathbf{x}_0 = (x^{(1)}, x^{(2)}, \ldots, x^{(k)})$ be a point in the polytope. By construction of C, we know that $\forall i \leq k, x^{(i)} \leq C$. The trajectory $\mathbf{x}(t)$, in that generalized eigenspace is

$$\begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(k)} \end{bmatrix}(t) = \exp(\lambda t) \begin{bmatrix} x^{(1)} + x^{(2)}t + \frac{x^{(3)}t^2}{2} + \cdots + \frac{x^{(k)}t^{k-1}}{(k-1)!} \\ x^{(2)} + x^{(3)}t + \cdots + \frac{x^{(k-1)}t^{k-2}}{(k-2)!} \\ \vdots \\ x^{(k)} \end{bmatrix}.$$

The trajectory, limited to this Jordan block, will either escape the hypercube $[-C, C]^d$ that encloses $\mathcal{P}_{BQ^{-1},\mathbf{c}}$, or will become so small that it will be at distance less than ε from the fixed point $\mathbf{0}$. We therefore consider three cases: $\lambda = 0$ and $\lambda > 0$ for which the trajectory will grow, and $\lambda < 0$ which decreases the coefficients. Once we have an escape bound for each eigenvalue, we will deduce a uniform bound for the entire trajectory.

Note that escaping the hypercube or converging to a fixed point do not give symmetric results: If we find a single component that grows larger than C, this is enough to escape the polytope, but *all* dimensions need to become smaller than ε in order to escape via entering the ε-region around the fixed point.

Case $\lambda < 0$. For all $j \leq k$, $x^{(j)}(t) = \exp(\lambda t) \sum_{i=j}^{k} x^{(i)} \frac{t^{i-j}}{(i-j)!}$. Using the bounds on the coefficients, we thus have when $t > 1$

$$|x^{(j)}(t)| = |\exp(\lambda t) \sum_{i=j}^{k} x^{(i)} \frac{t^{i-j}}{(i-j)!}| \leq \exp(\lambda t) k C t^k \quad \text{for } j \in \{1, \ldots, k\}$$

In order to have $|x^{(j)}(t)| < \varepsilon$, it is enough to have $\exp(\lambda t)kCt^k < \varepsilon$, which is equivalent to $\frac{kCt^k}{\varepsilon} < \exp(-\lambda t)$, and $t > \frac{1}{-\lambda}\log\left(\frac{kC}{\varepsilon}\right) + \frac{k}{-\lambda}\log t$

Here we need a small technical lemma.

▶ **Lemma 6** (Lemma A.1 and A.2 from [14]). *Suppose $a \geq 1$ and $b > 0$, then $t \geq a\log t + b$ if $t \geq 4a\log(2a) + 2b$.*

Applying this lemma with $a = \max\{1, \frac{k}{-\lambda}\}$ (we assume $\frac{k}{-\lambda} > 1$ in the following in order not to overload the formulas) and $b = \frac{1}{-\lambda}\log\left(\frac{kC}{\varepsilon}\right)$, we get a bound T_λ such that for all $j \leq k$, $x^{(j)}(T) < \varepsilon$, namely

$$\boxed{T_\lambda \leq \frac{4k}{-\lambda}\log\left(\frac{2k}{-\lambda}\right) + \frac{2}{-\lambda}\log\left(\frac{kC}{\varepsilon}\right).}$$

Case $\lambda = 0$. In this case, the trajectory restricted to this eigenspace is

$$x^{(j)}(t) = \sum_{i=j}^{k} x^{(i)}\frac{t^{i-j}}{(i-j)!} \quad \text{for } j \in \{1,\dots,k\}.$$

Assume that there exists $j \geq 2$ such that $|x^{(j)}| > \varepsilon$. This holds because by the definition of ε a point of the polytope is at distance at least ε from a fixed point. In particular, the line $\{x_j = 0 \mid j \neq 1\}$ is a line of fixed points of the differential equation. Now we require a time T_λ such that at least one of these components is larger in magnitude than $|C|$. We construct an upper bound on this time iteratively, using the fact that at least one coefficient $x^{(j)}$ is greater than ε, and all of them are less than C, giving the following bound on T_λ:

$$\boxed{T_\lambda \leq \frac{1}{k}\left(\frac{k^2C}{\varepsilon}\right)^{2^{k-1}}.}$$

Case $\lambda > 0$. This case proceeds similarly to the $\lambda = 0$ case, although the presence of an exponential factor gives us a much better bound T_λ:

$$\boxed{T_\lambda \leq \frac{2^{k-1}}{\lambda}\log\left(\frac{kC}{\varepsilon}\right).}$$

4.4 Constructing a Uniform Bound

We can now combine the results of the previous sections to get a uniform escape bound, considering all eigenvalues (real or not) simultaneously. Let the complex eigenvalues of A be $\{\eta_1 + i\theta_1, \eta_1 - i\theta_1 \dots, \eta_r + i\theta_r, \eta_r - i\theta_r\}$ and the real eigenvalues be $\{\lambda_1,\dots,\lambda_s\}$. Consider an arbitrary trajectory $\mathbf{x}(t)$ satisfying the differential equation $\dot{\mathbf{x}}(t) = A\mathbf{x}(t)$. By Lemma 4 we know that for $T_c := \sum_{j=1}^{r}\nu(\eta_j + i\theta_j)\frac{\pi}{\theta_j}$ there exists a point in the convex hull of $\{\mathbf{x}(t) \mid 0 \leq t \leq T_c\}$ that lies in the real eigenspace of A. This allows us to derive a bound on the escape time of the polytope $\mathcal{P}$ from a bound on the escape time of $\mathcal{P} \cap \mathcal{V}^r$. Indeed, let T_r be such that every "real" trajectory escapes the polytope in time T_r. Then any "complex" trajectory of duration $T_c + T_r$ contains in its convex hull a "real" trajectory of duration T_r which thus must have escaped the polytope. As the polytope is convex, this means that the complex trajectory itself escaped.

As for the subspace $\mathcal{V}^r$, we can derive from the escape bounds T_λ on each eigenspace computed in Subsection 4.3 a time bound beyond which every real point has escaped the polytope.

▶ **Lemma 7** (Real Time Bound). *Given an initial point $\mathbf{x}_0 \in \mathbb{R}^n$ with zero components in $\mathcal{V}^c$, the trajectory $\mathbf{x}(t)$ escapes within time $T_r = 2\max_\lambda T_\lambda$.*

Proof. Within a time $T_r/2 = \max_\lambda T_\lambda$, thanks to the analysis of subsection 4.3, there are three possibilities:

- the trajectory escapes the hypercube of size C, this occurs if there was a coefficient associated to a non-negative eigenvalue that was larger than ε;
- all coefficients are now smaller than ε, entering the hypercube of size ε and escaping the polytope since all the purely imaginary coefficients are zero;
- some component corresponding to a positive or zero eigenvalue originally less than ε has become greater than ε. In this case, waiting another $T_r/2$ amount of time puts the trajectory in the first case, ensuring it escapes.

Thus in all cases the trajectory has escaped by time T_r. ◀

From the above, we can deduce that every trajectory escapes within time $T_r + T_c$. We finally obtain Theorem 3 by analysing the complexity of this time bound in terms of the number of bits of the instance and its dimension.

The magnitude of the resulting escape bound is singly exponential in the bit size of the matrix entries and doubly exponential in the dimension of the matrix. However, if the matrix is diagonalizable or invertible, we can ignore the case where the eigenvalue is zero. Then the bound becomes $O(4^{bd^2})$ which is singly exponential in the bit size and dimension.

In Subsection 4.2 we showed how the change of basis explained the exponential factor in the number of dimensions. It is clear that the escape time can also be exponential in the bit size of the matrix.

For a very simple example, consider a 1-dimensional case where the polytope is the interval $[1, 2]$ and the differential equation is $\dot{x}(t) = 2^{-b}x(t)$ (which obviously can be written using constants of bit size at most b). Then the initial point $x_0 = 1$ yields a trajectory $x(t) = \exp(2^{-b}t)x_0$ whose escape time is $2^b \log 2$, which is exponential in b.

5 The Discrete Case

Tiwari [15] and Braverman [5] have shown decidability for the DPEP over the rationals and reals. In general, even if every trajectory is known to be escaping, it is not possible to place a uniform bound on the number of steps. However if the polytope is compact, we can use techniques similar to those used for the CPEP in order to provide a bound.

▶ **Theorem 8.** *Given a d-dimensional positive DPEP instance $(A, B, \mathbf{c})$ where the rational numbers use at most $b \in \mathbb{N}$ bits and an initial point $\mathbf{x}_0$, then for $N = e^{bd^{O(d)}}$, we have $\mathbf{x}_N \notin \mathcal{P}_{B,\mathbf{c}}$.*

Sketch of proof. We reduce the problem to the continuous case. Assuming every eigenvalue is positive, the matrix logarithm G of A is well defined. The trajectory of a continuous linear dynamical system generated by G is of the form $\mathbf{x}(t) = \exp(Gt)\mathbf{x}(0)$. In particular, for an initial point x_0 and $n \in \mathbb{N}$, we have

$$\mathbf{x}(n) = \exp(Gn)\mathbf{x}_0 = \exp(G)^n\mathbf{x}_0 = A^n\mathbf{x}_0 = \mathbf{x}_n$$

Therefore, we can relate the escape time of the CPEP instance $(G, B, \mathbf{c})$ to the escape time of the DPEP instance $(A, B, \mathbf{c})$.

The eigenvalues that are not positive are dealt with using a variant of the convex hull Lemma 4. ◀

References

1 R. Alur. *Principles of Cyber-Physical Systems*. MIT Press, 2015.

2 A. Bacciotti and L. Mazzi. Stability of dynamical polysystems via families of Lyapunov functions. *Jour. Nonlin. Analysis*, 67:2167–2179, 2007.

3 P. C. Bell, J.-C. Delvenne, R. M. Jungers, and V. D. Blondel. The continuous Skolem-Pisot problem. *Theor. Comput. Sci.*, 411(40-42):3625–3634, 2010.

4 V. Blondel and J. Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 36(9):1249–1274, 2000. doi:10.1016/S0005-1098(00)00050-9.

5 M. Braverman. Termination of integer linear programs. In *Proc. Intern. Conf. on Computer Aided Verification (CAV)*, volume 4144 of *LNCS*. Springer, 2006.

6 J.-Y. Cai. Computing Jordan normal forms exactly for commuting matrices in polynomial time. *Int. J. Found. Comput. Sci.*, 5(3/4):293–302, 1994. doi:10.1142/S0129054194000165.

7 E. B. Castelan and J.-C. Hennet. On invariant polyhedra of continuous-time linear systems. *IEEE Transactions on Automatic Control*, 38(11):1680–85, 1993.

8 Ventsislav Chonev, Joël Ouaknine, and James Worrell. On recurrent reachability for continuous linear dynamical systems. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 515–524, 2016.

9 Ventsislav Chonev, Joël Ouaknine, and James Worrell. On the Skolem Problem for continuous linear dynamical systems. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 100:1–100:13, 2016.

10 E. Hainry. Reachability in linear dynamical systems. In *Logic and Theory of Algorithms, 4th Conference on Computability in Europe, CiE 2008, Athens, Greece, June 15-20, 2008, Proceedings*, pages 241–250, 2008.

11 J. Ouaknine, J. Sousa Pinto, and J. Worrell. On the polytope escape problem for continuous linear dynamical systems. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control, HSCC 2017, Pittsburgh, PA, USA, April 18-20, 2017*, pages 11–17, 2017. doi:10.1145/3049797.3049798.

12 André Platzer. *Logical Foundations of Cyber-Physical Systems*. Springer, 2018.

13 S. Sankaranarayanan, T. Dang, and F. Ivancic. A policy iteration technique for time elapse over template polyhedra. In *Proceedings of HSCC*, volume 4981 of *LNCS*. Springer, 2008.

14 Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

15 A. Tiwari. Termination of linear programs. In *Proc. Intern. Conf. on Comp. Aided Verif. (CAV)*, volume 3114 of *LNCS*. Springer, 2004.

The SDP Value for Random Two-Eigenvalue CSPs

Sidhanth Mohanty
EECS Department, University of California, Berkeley, CA, USA
sidhanthm@berkeley.edu

Ryan O'Donnell
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
odonnell@cs.cmu.edu

Pedro Paredes
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
preisben@cs.cmu.edu

Abstract

We precisely determine the SDP value (equivalently, quantum value) of large random instances of certain kinds of constraint satisfaction problems, "two-eigenvalue 2CSPs". We show this SDP value coincides with the spectral relaxation value, possibly indicating a computational threshold. Our analysis extends the previously resolved cases of random regular 2XOR and NAE-3SAT, and includes new cases such as random Sort_4 (equivalently, CHSH) and Forrelation CSPs. Our techniques include new generalizations of the nonbacktracking operator, the Ihara–Bass Formula, and the Friedman/Bordenave proof of Alon's Conjecture.

2012 ACM Subject Classification Theory of computation → Semidefinite programming

Keywords and phrases Semidefinite programming, constraint satisfaction problems

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.50

Related Version A full version of the paper is available at `https://arxiv.org/abs/1906.06732`.

Funding *Sidhanth Mohanty*: Supported by NSF grant CCF-1718695.
Ryan O'Donnell: Supported by NSF grant CCF-1717606. This material is based upon work supported by the National Science Foundation under grant numbers listed above. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation (NSF).
Pedro Paredes: Supported by NSF grant CCF-1717606.

Acknowledgements We thank Yuval Peled for emphasizing the bipartite graph view of additive lifts, and Tselil Schramm for helpful discussions surrounding the trace method on graphs. S.M. would like to thank Jess Banks and Prasad Raghavendra for plenty of helpful discussions on orthogonal polynomials and nonbacktracking walks. Finally, we are grateful to Xinyu Wu for bringing the relevance of [13] to our attention and helping us to understand the issues discussed in Section A.

1 Introduction

This work is concerned with the average-case complexity of constraint satisfaction problems (CSPs). In the theory of algorithms and complexity, the most difficult instances of a given CSP are arguably random (sparse) instances. Indeed, the assumed intractability of random CSPs underlies various cryptographic proposals for one-way functions [31, 35], pseudorandom generators [11], public key encryption [6], and indistinguishability obfuscation [39], as well as hardness results for learning [21] and optimization [27]. Random CSPs also provide a rich testbed for algorithmic and lower-bound techniques based on statistical physics [44] and convex relaxation hierarchies [36, 52].

For a random, say, Max-Cut instance average degree d, its optimum value is with high probability (whp) concentrated around a certain function of d. Similarly, given a random 3SAT instance where each variable participates in an average of d clauses, the satisfiability status is whp determined by d. However explicitly working out the optimum/satisfiability as a function of d is usually enormously difficult; see, for example, Ding–Sly–Sun's landmark verification [25] of the kSAT threshold for sufficiently large k, or Talagrand's proof [55] of the Parisi formula for the Sherrington–Kirkpatrick model (Max-Cut with random Gaussian edge weights). The latter was consequently used by Dembo–Montanari–Sen [23] (see also [54]) to determine that the Max-Cut value in a random d-regular graph is a $\frac{1}{2} + \frac{P^*}{\sqrt{d}}(1 \pm o_d(1))$ fraction of edges (whp), where $P^* \approx .7632$ is an analytic constant arising from Parisi's formula.

Computational gaps for certification. Turning to computational issues, there are two main algorithmic tasks associated with an n-variable CSP: *searching* for an assignment achieving large value (hopefully near to the optimum), and *certifying* (as, e.g., convex relaxations do) that no assignment achieves some larger value. Let's take again the example of random d-regular Max-Cut, where whp we have $\mathrm{OPT} \approx \frac{1}{2} + \frac{P^*}{\sqrt{d}}$. It follows from [41] there is an efficient algorithm that whp finds a cut of value at least $\frac{1}{2} + \frac{2/\pi}{\sqrt{d}}$. One might say that this provides a $\frac{2}{\pi P^*}$-approximation for the search problem, where $\frac{2}{\pi P^*} \approx .83$. On the other side, the Max-Cut in a d-regular graph G is always at most $\frac{1}{2} + \frac{-\lambda_{\min}(G)}{2d}$, and Friedman's proof of Alon's Conjecture [29] shows that $-\lambda_{\min}(G) \leqslant 2\sqrt{d-1} + o_n(1)$ whp; thus computing the smallest eigenvalue efficiently certifies $\mathrm{OPT} \lessapprox \frac{1}{2} + \frac{1}{\sqrt{d}}$. One might say that this efficient spectral algorithm provides a $\frac{1}{P^*}$-approximation for the certification problem, where $\frac{1}{P^*} \approx 1.31$.

It is a very interesting question whether either of these approximation algorithms can be improved. On one hand, it would seem desirable to have efficient algorithms that come arbitrarily close to matching the "true" answer on random inputs. On the other hand, the nonexistence of such algorithms would be useful for cryptography and hardness-of-approximation and -learning results.

Speaking broadly, efficient algorithms for the search problem seem to do better than efficient algorithms for the certification problem. For example, given a random 3SAT instance with clause density slightly below the satisfiability threshold of ≈ 4.2667, there are algorithms [42] that seem to efficiently find satisfying assignments whp. On the other hand, the longstanding Feige Hypothesis [27] is that efficient algorithms cannot certify unsatisfiability at any large constant clause density, and indeed there is no efficient algorithm that is known to work at density $o(\sqrt{n})$. Similarly, for the Sherrington–Kirkpatrick model, Montanari [47] has recently given an efficient PTAS for the search problem[1], whereas the best known efficient algorithm for the certification problem is again only a $1/P^*$-approximation. These kinds of gaps seem to be closely related to "information-computation gaps" and Kesten–Stigum thresholds for information recovery and planted-CSP problems.

In this work we focus on potential computational thresholds for random CSP certification/refutation problems in the sparse setting, and in particular how these thresholds depend on the "type" of the CSP. For CSPs with a predicate supporting a pairwise-uniform distribution – such as kSAT or kXOR, $k \geqslant 3$ – there is solid evidence that the computational threshold for efficient certification of unsatisfiability is very far from the actual unsatisfiability threshold. Such CSPs are whp unsatisfiable at constant constraint density, but any polynomial-time algorithm using the powerful Sum-of-Squares (SoS) algorithm fails to

[1] Modulo a widely believed analytic assumption.

refute unless the density is $\Omega(\sqrt{n/\log n})$ [36]. But outside the pairwise-supporting case, and especially for "2XOR-like" CSPs such as Max-Cut and NAE-3SAT (Not-All-Equal 3SAT), the situation is much more subtle. For one, the potential gaps are much more narrow; e.g., in random NAE-3SAT, even a simple spectral algorithm efficiently refutes satisfiability at constant constraint density. Thus one must look into the actual *constants* to determine if there may be an "information-computation" gap. Another concern is that evidence for computational hardness in the form of SoS lower bounds (degree 4 or higher) seems very hard to come by (see, e.g., [46]).

Prior work. Let us describe two prior efforts towards computational thresholds for upper-bound-certification in "2XOR-like" random CSPs. Montanari and Sen [48] (see also [8]) investigated the Max-Cut problem in random d-regular graphs, where the optimum value is $\frac{1}{2} + \frac{P^*}{\sqrt{d}}$ whp (ignoring $1 \pm o_d(1)$ factors). Friedman's Theorem implies that the basic eigenvalue bound efficiently certifies the value is at most $\frac{1}{2} + \frac{1}{\sqrt{d}}$. By using a variant of the Gaussian Wave [26, 20, 34] construction for the infinite d-ary tree, Montanari and Sen were able to show that even the Goemans–Williamson semidefinite programming (SDP) relaxation [22, 30] is still just $\frac{1}{2} + \frac{1}{\sqrt{d}}$ whp. This may be considered evidence that *no* polynomial-time algorithm can certify upper bounds better than $\frac{1}{2} + \frac{1}{\sqrt{d}}$, as Goemans–Williamson has seemed to be the optimal polynomial-time Max-Cut algorithm in all previous circumstances. Of course it would be more satisfactory to see higher-degree SoS lower bounds, but as mentioned these seem very difficult to come by.

Recently, Deshpande et al. [24] have given similar results for random "c-constraint-regular" NAE-3SAT CSPs; i.e., random instances where each variable participates in exactly c NAE-3SAT constraints.[2] Random c-constraint-regular instances of NAE-3SAT are easily shown to be unsatisfiable (whp) for $c \geqslant 8$. Deshpande et al. identified an exact threshold result for when the natural SDP algorithm is able to certify unsatisfiability: it succeeds (whp) if $c > 13.5$ and fails (whp) if $c < 13.5$. Indeed, since c is always an integer, they show that for $c \geqslant 14$ even the basic spectral algorithm certifies unsatisfiability, whereas for $c \leqslant 13$ even the SDP augmented with "triangle inequalities" fails to certify unsatisfiability. Again, this gives evidence for a gap between the threshold for unsatisfiability and the threshold for computationally efficient refutation. The techniques used by Deshpande et al. are similar to those of Montanari–Sen, except with random (b, c)-biregular graphs replacing random c-regular graphs. (The reason is that the primal graph of a random c-constraint-regular NAE-3SAT instance resembles the square of a random $(3, c)$-biregular graph.)

In fact, the Deshpande et al. result is more refined, being concerned not just with satisfiability of random NAE-3SAT instances, but their optimal value as maximization problems. Letting $f(c) = \frac{9}{8} - \frac{3}{8} \cdot \frac{(\sqrt{c-1}-\sqrt{2})^2}{c}$ for $c \geqslant 3$, they determined that in a random c-constraint-regular NAE-3SAT instance, the SDP value is whp $f(c) \pm o(1)$; and furthermore, this is also the basic eigenvalue bound and the SDP-with-triangle-inequalities bound. (Note that $f(13.5) = 1$.) Again, this may suggest that in these instances, computationally efficient algorithms can only certify that at most an $f(c) + o(1)$ fraction of constraints are simultaneously satisfiable.

[2] We have changed terminology to avoid a potential future confusion; we will be associating NAE-3SAT constraints with triangle graphs, so c-constraint-regular NAE-3SAT instances will be associated to $2c$-regular graphs.

1.1 Our results

The goal of the present work is to generalize the preceding Montanari–Sen and Deshpande et al. results to a broader class of sparse random 2CSPs and 2XOR-like optimization problems, obtaining precise values for their SDP values. Along the way, we need to come to a deeper understanding of the combinatorial and analytic tools used (nonbacktracking walks, Ihara–Bass formulas, eigenvalues of random graphs and infinite graphs) and we need to extend these tools to graphs that do *not* locally resemble trees (as in Montanari–Sen and Deshpande et al.). We view this aspect of our work as a main contribution, beyond the mere statement of SDP values for specific CSPs. We defer to Section 1.2.1 more detailed discussions of the technical conditions under which we can obtain Ihara–Bass and Friedman-, and Gaussian Wave-type theorems. But roughly speaking, we are able to analyze the SDP value for random regular instances of optimization problems where each "constraint" (not necessarily a predicate) is an *edge-signed graph with two eigenvalues.* Such constraints include: a single edge (corresponding to random regular Max-Cut or 2XOR as in Montanari–Sen); a complete graph (studied by Deshpande et al., with the K_3 case corresponding to random regular NAE-3SAT); the Sort_4 (a.k.a. CHSH) predicate; and, $\mathsf{Forrelation}_k$ constraints. These last two have motivation from quantum mechanics, and in fact the SDP value of the associated CSPs is precisely their "quantum value". We discuss quantum connections further in Section B.1.

We state here two theorems that our new techniques allow us to prove. Recall the Sort_4 predicate, which is satisfied iff its 4 Boolean inputs x_1, x_2, x_3, x_4 satisfy $x_1 \leqslant x_2 \leqslant x_3 \leqslant x_4$ or $x_1 \geqslant x_2 \geqslant x_3 \geqslant x_4$. We precisely define "random c-constraint-regular CSP instance" in Section 2, but in brief, we work in the "random lift" model, each variable participates in exactly c constraints, and each constraint is given random negations.[3]

▶ **Theorem 1.** *For random c-constraint-regular instances of the Sort_4-CSP, the SDP-satisfiability threshold occurs (in a sense) at $c = 4 + 2\sqrt{2} \approx 6.83$. Indeed, if $c \geqslant 7$ then even the basic eigenvalue bound certifies unsatisfiability (whp); and, if $c \leqslant 6$ then the basic SDP relaxation fails to certify unsatisfiability (whp).*

We remark that the trivial first-moment calculation shows that a random c-constraint-regular Sort_4-CSP is already unsatisfiable whp at degree $c = 4$. Thus we again have evidence for a gap between the true threshold for unsatisfiability and the efficiently-certifiable threshold.

Generalizing this, the $\mathsf{Forrelation}_k$ constraint is a certain (quantum-inspired) map $\{\pm 1\}^{2^k+2^k} \to [-1, +1]$ that measures how correlated one k-bit Boolean function is with the Fourier transform of a second k-bit Boolean function. We give precise details in Section B.1; here we just additionally remark that $\mathsf{Forrelation}_1$ corresponds to the "CHSH game", and that $\frac{1}{2} + \mathsf{Forrelation}_1$ is equivalent to the Sort_4 predicate.

▶ **Theorem 2.** *For random c-constraint-regular instances of the $\mathsf{Forrelation}_k$-CSP and any constant $\varepsilon > 0$, the SDP value is whp in the range $\frac{2\sqrt{c-1}}{c \cdot 2^{k/2}} \pm \varepsilon$. This is also true of the eigenvalue bound.*

Note that the formula above decreases on $[2^{-k/2}, \infty]$. When considering the SDP value for $\frac{1}{2} + \mathsf{Forrelation}_1$, the formula above crosses the threshold of 1 when $c = 4 + 2\sqrt{2}$, yielding the statement in Theorem 1 about the SDP-satisfiability threshold of random c-constraint-regular Sort_4-CSPs.

[3] Our result holds for either of the following two negation models: (i) each *constraint* is randomly negated; or, (ii) the constraints are not negated, but each constraint is applied to random *literals* rather than random variables.

1.2 Sketch of our techniques

Here we sketch how our results like Theorem 1 and Theorem 2 are proven, using random Sort_4-CSPs as a running example. A key property of the Sort_4 predicate is that it is essentially equivalent to the "2XOR" instance on the left picture of Figure 1.

More precisely, suppose $(x_1, x_2, x_3, x_4) \in \{\pm 1\}^4$ satisfies the Sort_4 predicate. Then in the graph on the left picture of Figure 1, exactly 3 out of 4 edges will be "satisfied" – where an edge is considered satisfied when the product of its endpoint-labels equals the edge's label. Conversely, if (x_1, x_2, x_3, x_4) doesn't satisfy Sort_4 then exactly 1 out of the 4 edges above will be satisfied. Now suppose we choose a random n-vertex c-constraint-regular instance $\mathcal{I}$ of the Sort_4-CSP with, say, $c = 2$. A small piece of such an instance might look like the middle picture of Figure 1.

Up to a trivial affine shift in the objective function, the optimization task is now to label the variables/vertices of $\mathcal{I}$ with ± 1 values $x_1, \dots, x_n$ so as to maximize $\frac{1}{n}\sum_{ij} A_{ij} x_i x_j$, where $A \in \{0, \pm 1\}^{n \times n}$ is the adjacency matrix of the edge-signed graph partially depicted above. The "eigenvalue upper bound" $\mathrm{EIG}(\mathcal{I})$ arises from allowing the x_i's to be arbitrary real numbers, subject to the constraint $\sum_i x_i^2 = n$. The "SDP upper bound" $\mathrm{SDP}(\mathcal{I})$ (which can be shown to be at least as tight: $\mathrm{SDP}(\mathcal{I}) \leqslant \mathrm{EIG}(\mathcal{I})$) arises from allowing the x_i's to be arbitrary unit vectors in $\mathbb{R}^n$, with the inner product $\langle x_i, x_j \rangle$ replacing $x_i x_j$ in the objective function. Our goal is to identify some quantity $f(c)$ (it will be $\frac{1+\sqrt{2}}{2}$ in the $c = 2$ case) such that

$$\mathrm{EIG}(\mathcal{I}) \lesssim f(c) \lesssim \mathrm{SDP}(\mathcal{I}) \tag{1}$$

up to $1 \pm o(1)$ factors, with high probability. This establishes that all three quantities are equal (up to $1 \pm o(1)$, whp), since $\mathrm{SDP}(\mathcal{I}) \leqslant \mathrm{EIG}(\mathcal{I})$ always.

In this section we mainly describe how to obtain the optimal inequality on the left in (1); i.e., how to give a tight bound on the eigenvalues of (the edge-signed graph induced by) $\mathcal{I}$. Notice that if we were studying just random Max-Cut or 2XOR CSPs, we would have to get tight bounds on the eigenvalues of a standard random c-regular graph.[4] Excluding the top eigenvalue of c in the case of Max-Cut, these eigenvalues are (whp) all at most $2\sqrt{c-1} + o_n(1)$ in magnitude. This is thanks to Friedman's (difficult) proof of Alon's Conjecture [29], made moderately less difficult by Bordenave [12]. The "magic number" $2\sqrt{c-1}$ is precisely the spectral radius of the *infinite* c-regular tree – i.e., the infinite graph that random c-regular graphs "locally resemble".

Returning to random 2-constraint-regular instances of the Sort_4-CSP, the (edge-signed) infinite graph X that *they* "locally resemble" is the right picture of Figure 1.

Here $X \coloneqq \mathsf{Sort}_4 \ \dot{+} \ \mathsf{Sort}_4$ is the so-called *additive product* of 2 copies of the Sort_4 graph, a notion recently introduced in [45]. By analogy with Alon's Conjecture, it's natural to guess that the spectral radius of a random 2-constraint-regular Sort_4-CSP instance is whp $\rho(X) \pm o_n(1)$, where $\rho(X)$ denotes the spectral radius of X (which can be shown to be $2\sqrt{2}$). Indeed, our main effort is to prove the upper bound of $\rho(X) + o_n(1)$, thereby establishing the left inequality in (1) with $f(c) = \rho(X)$.

[4] More precisely, for random Max-Cut we have to lower-bound the smallest eigenvalue; for random 2XOR – which includes randomly negating edges – we have to upper-bound the largest eigenvalue. In the Max-Cut version with no negations, there is the usual annoyance that there is always a first "trivial" eigenvalue of c, and one essentially wants to bound the second-largest (in magnitude) eigenvalue. The effect of random negations is generally to eliminate the trivial eigenvalue, allowing one to focus simply on the spectral radius of the adjacency matrix. This technical convenience is one reason we will always work in a model that includes random negations.

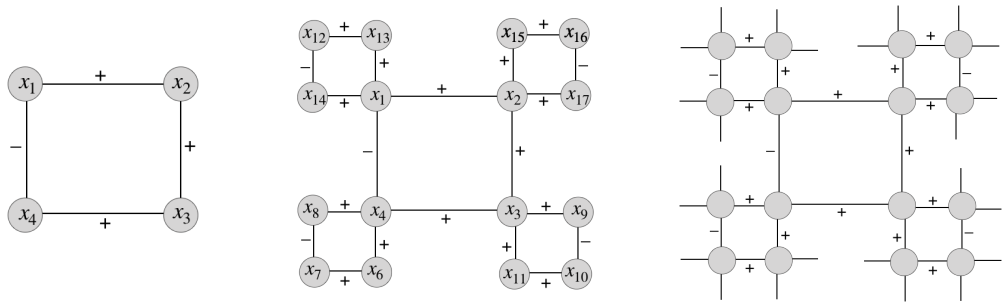


Figure 1 The Sort_4 predicate, a piece of Sort_4 instance and Sort_4 infinite graph.

Regarding this proof of the left inequality, Xinyu Wu has brought to our attention the relevance to our work of a recent paper by Bordenave and Collins [13]. Briefly put, their paper establishes a Friedman/Bordenave theorem for large random graphs whose adjacency matrices are noncommutative polynomials in a fixed number of independent random matching matrices and permutation matrices (together with their transposes), analogously to our Theorem 22 (which we will state in one of the following sections). We detail how both theorems compare in Section A. As for the right inequality, it can proven using the "Gaussian Wave" idea, allowing one to convert approximate eigenvectors of the infinite graph X to matching SDP solutions on random finite graphs $\mathcal{I}$. We carry this out in Section E.

1.2.1 Friedman/Bordenave Theorems for two-eigenvalue additive lifts

As stated, our main task in the context of large random 2-constraint-regular Sort_4-CSP instances is to show that their spectral radius is at most $\rho(X) + o_n(1)$ whp. Incidentally, the lower bound of $\rho(X) - o_n(1)$ indeed holds; it follows from a generalization of the "Alon–Boppana Bound" due to Grigorchuk and Żuk [32]. As for the upper bound, the recent work [45] implies the analogous "Ramanujan graph" statement; namely, that there *exist* arbitrarily large 2-constraint-regular Sort_4-CSP instances with largest eigenvalue exactly upper-bounded by $\rho(X)$. However we need the analogue of Friedman/Bordenave's Theorem. Unlike in [45] we are not able to prove it for arbitrary additive products; we are able to prove it for additive products of "two-eigenvalue" edge-signed graphs. To explain why, we first have to review the proofs of the Alon Conjecture (that c-regular random graphs have their nontrivial eigenvalues bounded by $2\sqrt{c-1} + o_n(1)$).

Both Friedman's and Bordenave's proof of the Alon Conjecture rely on very sophisticated uses of the Trace Method. Roughly speaking, this means counting closed walks of a fixed length k in random c-regular graphs, and (implicitly) comparing these counts to those in the c-regular infinite tree. Actually, both works instead count only *nonbacktracking* walks. The fact that one can relate nonbacktracking walk counts to general walk counts is thanks to an algebraic tool called the *Ihara–Bass Formula* (more on which later); this idea was made more explicit in Bordenave's proof. Incidentally, use of the nonbacktracking walk operator has played a major role in recent algorithmic breakthroughs on community detection and related results (e.g., [37, 49, 43, 14]).

A reason for passing to nonbacktracking closed walks is that it greatly simplifies the counting. Actually, in the case of the infinite c-regular tree, it *over*simplifies the counting; infinite trees have no nonbacktracking closed walks at all! However, the correct quantity to look at is "almost" nonbacktracking walks of length k, meaning ones that are nonbacktracking for the first $k/2$ steps, and for the last $k/2$ steps, but which may backtrack once right in

the middle. There are essentially $(c-1)^{k/2}$ of these in the c-regular infinite tree (one may take $k/2$ arbitrary steps out, but then one must directly walk back home), yielding a value of $((c-1)^{k/2})^{1/k} = \sqrt{c-1}$ for the spectral radius of the nonbacktracking operator of the c-regular infinite tree. Bordenave uses (a very tricky version of) the Trace Method to analogously show that the spectral radius of the nonbacktracking operator of a random c-regular graph is $\sqrt{c-1} + o_n(1)$ whp. Thanks to the Ihara–Bass Formula, this translates into a bound of $2\sqrt{c-1} + o_n(1)$ for the spectral radius of the usual adjacency operator.

Returning now to our scenario of random 2-constraint-regular Sort_4-CSP instances (with their analogous infinite edge-signed graph X), we encounter a severe difficulty. Namely, passing to nonbacktracking walks no longer creates a drastic simplification in the counting, since there are nonbacktracking cycles within the constraint graphs themselves (in our example, 4-cycles graphs). Thus nonbacktracking closed walks in large random instances can have complicated structures, with many internal nonbacktracking cycles.

A saving grace in the case of Sort_4-CSPs, and also ones based on $\mathsf{Forrelation}_k$ or complete-graph constraints for example, is that the adjacency matrices of these graphs have only *two distinct eigenvalues*. (We will also use that their edge weights are ± 1.) For example, after rearranging the variables in the Sort_4 predicate, its adjacency matrix is

$$A = \begin{pmatrix} 0 & 0 & +1 & +1 \\ 0 & 0 & +1 & -1 \\ +1 & +1 & 0 & 0 \\ +1 & -1 & 0 & 0 \end{pmatrix}, \tag{2}$$

which has eigenvalues of $\pm\sqrt{2}$ (with multiplicity 2 each). The two-eigenvalue property implies that A satisfies a quadratic equation, and hence any polynomial in A is equivalent to a polynomial of degree *at most* 1. The upshot is that we can relate general walks in Sort_4-CSPs (or more generally, CSPs with two-eigenvalue constraints) to what we call *nomadic* walks: ones that take *at most* 1 consecutive step within a single constraint. We will formally define and better motivate these in Section 2.3.

2 Preliminaries

2.1 2XOR optimization problems and their relaxations

All of the CSPs studied in this work (Max-Cut, NAE-3SAT, Sort_4, $\mathsf{Forrelation}_k$, etc.) will effectively reduce to 2XOR *optimization problems* – equivalently, the problem maximizing a homogeneous degree-2 polynomial with ± 1 coefficients over the Boolean hypercube.

▶ **Definition 3** (Optimization of 2XOR instances). *Let $G = (V, E)$ be an undirected graph (possibly with parallel edges), with edge-signing $\mathrm{wt} : E \to \{\pm 1\}$. We call the pair $\mathcal{I} = (G, \mathrm{wt})$ an instance. The associated* 2XOR *optimization problem is to determine the (true) optimum value*

$$\mathrm{OPT}(\mathcal{I}) = \max_{x:V\to\{\pm1\}} \operatorname*{avg}_{e=\{u,v\}\in E} \{\mathrm{wt}(e)x_u x_v\} \in [-1, +1].$$

The special case in which $\mathrm{wt} \equiv -1$ is referred to as the Max-Cut *problem on G, as in this case $\frac{1}{2} + \frac{1}{2}\mathrm{OPT}(\mathcal{I}) = \mathsf{Max\text{-}Cut}(G)$, the maximum fraction of edges that can be cut by a bipartition of V.*

Determining $\mathrm{OPT}(\mathcal{I})$ is NP-hard in the worst case, leading to the study of computationally tractable approximations/relaxations. Two such approximations are the *eigenvalue bound* and the *SDP bound*, which we now recall.

▶ **Definition 4** (Adjacency matrix/operator). *The adjacency matrix A of a finite weighted graph (G, wt) has rows and columns indexed by V; the entry $A[u,v]$ equals the sum of $\mathrm{wt}(e)$ over all edges with endpoints $\{u,v\}$. In case G is infinite we can more generally define the adjacency operator A on $\ell_2(V)$ as follows:*

$$\text{for } F \in \ell_2(V), \quad AF(u) = \sum_{e=(u,v)\in E} \mathrm{wt}(e)F(v).$$

▶ **Definition 5** (Eigenvalue bound). *The eigenvalue bound $\mathrm{EIG}(\mathcal{I})$ for* 2XOR *instance $\mathcal{I}$ with adjacency matrix A is $\frac{n}{2|E|}\lambda_{\max}(A)$, where $\lambda_{\max}$ denotes the maximum eigenvalue. We have $\mathrm{OPT}(\mathcal{I}) \leqslant \mathrm{EIG}(\mathcal{I})$ always, as the eigenvalue bound captures the relaxation of* 2XOR *optimization where we allow any $x : V \to \mathbb{R}$ satisfying $\|x\|^2 = n$.*

The *SDP value* provides an even tighter upper bound on $\mathrm{OPT}(\mathcal{I})$, and is still efficiently computable.The SDP bound dates back to Lovász's Theta Function in the context of the IndependentSet problem [40], and was proposed in the context of the Max-Cut problem by Delorme and Poljak [22].

▶ **Definition 6** (SDP bound). *The SDP bound $\mathrm{SDP}(\mathcal{I})$ for* 2XOR *instance $\mathcal{I}$ is*

$$\mathrm{SDP}(\mathcal{I}) = \max_{\vec{x}:V\to S^{m-1}} \operatorname*{avg}_{e=\{u,v\}\in E} \{\mathrm{wt}(e)\langle \vec{x}_u, \vec{x}_v\rangle\} \in [-1,+1],$$

where S^{m-1} refers to the set of unit vectors in $\mathbb{R}^m$ and the maximum is also over m (though $m = n$ is sufficient). The following holds for all $\mathcal{I}$:

$$\mathrm{OPT}(\mathcal{I}) \leqslant \mathrm{SDP}(\mathcal{I}) \leqslant \mathrm{EIG}(\mathcal{I}).$$

The left inequality is obvious. One way to see the right inequality is to use the fact [22], based on SDP duality, that $\mathrm{SDP}(\mathcal{I})$ is also equal to the minimum value of the eigenvalue bound applied to $A + Y$, where A is the adjacency matrix and Y ranges over all matrices of trace 0.

Goemans and Williamson [30] famously showed that

$$\tfrac{1}{2} + \tfrac{1}{2}\mathrm{SDP}(\mathcal{I}) \leqslant 1.138(\tfrac{1}{2} + \tfrac{1}{2}\mathrm{OPT}(\mathcal{I}))$$

holds for every 2XOR instance, and Feige–Schechtman [28] showed their bound can be tight in the worst case. As for directly comparing $\mathrm{SDP}(\mathcal{I})$ and $\mathrm{OPT}(\mathcal{I})$, we have the following:

- ([17]) $\mathrm{SDP}(\mathcal{I}) \leqslant O(\mathrm{OPT}(\mathcal{I}) \cdot \log(1/\mathrm{OPT}(\mathcal{I})))$ always holds.
- When G is bipartite (a special case of particular interest, see Section B.1), it holds that $\mathrm{SDP}(\mathcal{I}) \leqslant K \cdot \mathrm{OPT}(\mathcal{I})$ for constant K. This is known as *Grothendieck's inequality* [33], and the constant is known [15] to satisfy $K < \pi/(2\ln(1+\sqrt{2})) \approx 1.78$.

2.2 2XOR graphs with only 2 distinct eigenvalues

As mentioned, the class of constraints that we treat in this work are those that can be modeled as 2XOR instances with 2 *distinct eigenvalues*. The $\mathsf{Forrelation}_k$ constraint is a prime example; when viewed as an edge-signed graph (i.e., ignoring the 2^{-2k} scaling factors), its eigenvalues are all $\pm 2^{k/2}$. Another example is the complete graph constraint on r variables, which has eigenvalues of $r-1$ and -1 (the latter with multiplicity $r-1$). The $r = 3$ complete-graph case, after a trivial affine shift, also corresponds to a Boolean predicate that is well known in the context of CSPs: the NAE-3SAT predicate, as studied in [24]. This is because

$$\text{NAE-3SAT}(x_1,x_2,x_3) = \frac{3}{4} - \frac{3}{4}(x_1x_2 + x_2x_3 + x_3x_1).$$

Let us make some definitions we will use throughout the paper.

▶ **Definition 7** (2-eigenvalue graphs). *We call an undirected, edge-weighted simple graph $\mathcal{I}$ a 2-eigenvalue graph if there are two real numbers λ_1 and λ_2 such that each eigenvalue of $\mathcal{I}$'s (signed) adjacency matrix A is equal to either λ_1 or λ_2.*

See, e.g., [53] for a paper studying such graphs. In this section, let us use the notation from Definition 7 and prove some properties that will be used throughout the paper.

First, since A is symmetric, its eigenvectors are spanning and therefore every vector can be written as the sum of a vector in $\ker(A - \lambda_1 \mathbb{1})$ and one in $\ker(A - \lambda_2 \mathbb{1})$. Thus:

▶ **Proposition 8.** *$(A - \lambda_1 \mathbb{1})(A - \lambda_2 \mathbb{1}) = 0$, where $\mathbb{1}$ denotes the identity matrix.*

This proposition implies that $A^2 = (\lambda_1 + \lambda_2)A - \lambda_1\lambda_2\mathbb{1}$. Thus we can deduce the following two facts:

▶ **Fact 9.** *For any $v \in V(G)$, $\sum_{u \in V(G)} A[u,v]^2 = A^2[v,v] = -\lambda_1\lambda_2$.*

▶ **Fact 10.** *For any pair of distinct vertices $u, v \in V(G)$,*

$$\sum_{w \in V(G)} A[u,w]A[w,v] = A^2[u,v] = (\lambda_1 + \lambda_2)A[u,v].$$

2.3 Random constraint graphs, instance graphs, and additive products

▶ **Definition 11** (Constraint graphs). *An r-ary, c-atom constraint graph is any n-fold lift $\mathcal{H}$ of the complete bipartite graph $K_{r,c}$. Each vertex on the c-regular side is called a variable vertex, and is typically depicted by a circle. The variable vertices are partitioned into r variable groups each of size n, called the 1st variable group, the 2nd variable group, etc. Each vertex on the r-regular side is called a constraint (or atom) vertex, and is typically depicted by a square. Again, the constraint vertices are partitioned into c constraint (or atom) groups of size n, called the 1st constraint/atom group, 2nd constraint/atom group, etc. When $n = 1$, we call $\mathcal{H}$ a base constraint graph. We also allow "$n = \infty$": this means we take the infinite (r,c)-biregular tree and partition its variable vertices into r groups and its constraint variables into c groups in such a way that every variable vertex in the ith group has exactly one neighbor from each of the c constraint groups, and similarly every constraint vertex in the jth group has exactly one neighbor from each of the r variable groups.*

▶ **Definition 12** (Instance graphs). *Let $\mathcal{A} = (A_1, \ldots, A_c)$ be a sequence of atoms, meaning edge-weighted undirected graphs on a common vertex set $[r]$. (In this paper, the edge-weights will usually be ± 1.) We also think of each atom as a collection of "2XOR-constraints" on variable set r. Now given an r-ary, c-atom constraint graph $\mathcal{H}$, we can combine it with the atom specification $\mathcal{A}$ to form the instance graph $\mathcal{I} \coloneqq \mathcal{A}(\mathcal{H})$. This edge-weighted undirected graph $\mathcal{I}$ has as its vertex set all the variable vertices of $\mathcal{H}$. The edges of $\mathcal{I}$ are formed as follows: We iterate through each $j \in [c]$ and each constraint vertex f in the jth constraint group of $\mathcal{H}$. Given f, with variables neighbors $v_1, \ldots, v_r$ in $\mathcal{H}$, we place a copy of atom A_j onto these vertices in $\mathcal{I}$. ($\mathcal{I}$ may end up with parallel edges.) We refer to the graph obtained by placing a copy of A_j on vertices $v_1, \ldots, v_r$ as A_f, and for any edge e in $\mathcal{I}$ that came from placing A_j, we define $\mathrm{Atom}(e) \coloneqq A_f$. We use $v \sim A_f$ to denote that v is one of $v_1, \ldots, v_r$. For $u, v \in \{v_1, \ldots, v_r\}$, $A_f(u,v)$ denotes the edge in A_f between u and v. And finally, denote the set $\{A_f : f$ constraint vertex in $\mathcal{H}\}$ with $\mathrm{Atoms}(\mathcal{I})$.*

▶ Remark 13. Forming $\mathcal{I}$ from $\mathcal{H}$ is somewhat similar to squaring $\mathcal{H}$ (in the graph-theoretic sense) and then restricting to the variable vertices. With this in mind, here is an alternate way to describe the edges of $\mathcal{I}$: For each pair of distinct vertices v, v' in $\mathcal{I}$ (in variable groups i and i', respectively) we consider all length-2 paths joining v and v' in $\mathcal{H}$. For each such path passing through a constraint vertex in constraint group j, we add the edge (v, v') into $\mathcal{I}$ with edge-weight $A_j[i, i']$ (which may be 0).

▶ Remark 14. We treat atoms as edge-weighted, undirected, complete graphs. Thus, for a constraint vertex f in constraint-graph $\mathcal{H}$, if there is an edge between vertices u and v, and an edge between vertices v and w in the atom A_f, then there is an edge between u and w in A_f. This view is significant in light of the proof of Theorem 38.

The following notions of additive lifts and additive products were introduced in [45]:

▶ **Definition 15** (Random additive lifts)**.** *In the context of r-ary, c-atom constraint graphs, a random n-lifted constraint graph simply means a usual random n-lift $\mathcal{H}$ (see, e.g., [10]) of the base constraint graph. Given atoms $\mathcal{A} = (A_1, \dots, A_c)$, the resulting instance graph $\mathcal{I} = \mathcal{A}(\mathcal{H})$ is called a random additive lift of $\mathcal{A}$.*

▶ **Definition 16** (Additive products)**.** *If instead $\mathcal{H}$ is the "∞-lift" of $K_{r,c}$, the resulting infinite instance graph $\mathcal{I} = \mathcal{A}(\mathcal{H})$ is called the additive product of $A_1, \dots, A_c$, denoted $A_1 \dotplus A_2 \dotplus \cdots \dotplus A_c$.*

We will also extend Definition 12 to allow random additive lifts with *negations*. Eventually we will define a general notion of "1-wise uniform negations", but let us begin with two special cases. In the "constraint negation" model, we assign to each constraint vertex f in $\mathcal{H}$ (from group j) an independent uniformly random sign ξ^f. Then, when the instance graph $\mathcal{I}$ is formed from $\mathcal{H}$, each edge engendered by the constraint f has its weight multiplied by ξ^f. (Thus the edges in this copy of the atom A_j are either all left alone or they are simultaneously negated, with equal probability.) In the "variable negation" model, for each group-j constraint vertex f, adjacent to variable vertices $v_1, \dots, v_r$, we assign independent and uniformly random signs $(\xi_i^f)_{i \in [r]}$ to the variables. Then when the copy of A_j is added into $\mathcal{I}$, the $\{i, i'\}$-edge has its weight multiplied by $\xi_i^f \xi_{i'}^f$. This corresponds to the constraint being applied to random *literals*, rather than variables.

Notice that in both of these negation models, every time a copy of atom A_j is placed into $\mathcal{I}$, its edges are multiplied by a collection of random signs $(\xi_{ij}^f)_{i,j \in [r]}$ which are "1-wise uniform". This is the only property we will require of a negation model.

▶ **Definition 17** (Random additive lifts with negations)**.** *A random additive lift with 1-wise uniform negations is a variant of Definition 12 where, for each constraint vertex f there are associated random signs $\xi_i^{(f)} \in \{\pm 1\}$, where $i \in [r]$. For each fixed f, the random variables $\xi_i^{(f)}$ are required to be ± 1 with probability $1/2$ each, but they may be arbitrarily correlated; across different f's, the collections $(\xi_i^{(f)})_{i \in [r]}$ must be independent. When the instance graph $\mathcal{I}$ is formed as $\mathcal{A}(\mathcal{H})$, and a copy of A_j placed into $\mathcal{I}$ thanks to constraint vertex f, each new edge $\{i, i'\}$ has its weight $A_j[i, i']$ multiplied by $\xi_{ii'}^{(f)} := \xi_i^{(f)} \xi_{i'}^{(f)}$.*

▶ Remark 18. For a given constraint-vertex f of an instance graph $\mathcal{I}$ obtained via a random additive lift with negations, the matrix $\mathrm{Adj}(A_f)$ has the same spectrum as $\mathrm{Adj}(\overline{A_f})$ where $\overline{A_f}$ denotes the subgraph prior to applying random negations, since there is a sign diagonal matrix D such that $\mathrm{Adj}(\overline{A_f}) = D \cdot \mathrm{Adj}(A_f) \cdot D^\dagger$.

2.4 Nomadic walks operators

▶ **Definition 19** (Nomadic walks). *Let $\mathcal{H}$ be a constraint graph, $\mathcal{A} = (A_1, \ldots, A_c)$ a sequence of atoms, and $\mathcal{I} = \mathcal{A}(\mathcal{H})$ the associated instance graph. For initial simplicity, assume the atoms are unweighted (i.e., all edge weights are $+1$). A nomadic walk in $\mathcal{I}$ is a walk where consecutive steps are prohibited from "being in the same atom". Note that if $r = 2$ and the atoms are single edges, a nomadic walk in $\mathcal{I}$ is equivalent to a nonbacktracking walk.*

To make the definition completely precise requires "remembering" the constraint graph structure $\mathcal{H}$. Each step along an edge of $\mathcal{I}$ corresponds to taking two consecutive steps in $\mathcal{H}$ (starting and ending at a variable vertex). The walk in $\mathcal{I}$ is said to be nomadic precisely when the associated walk in $\mathcal{H}$ is nonbacktracking.

Finally, in the general case when the atoms A_j have weights, each walk in $\mathcal{I}$ gets a weight equal to the product of the edge-weights used along the walk.

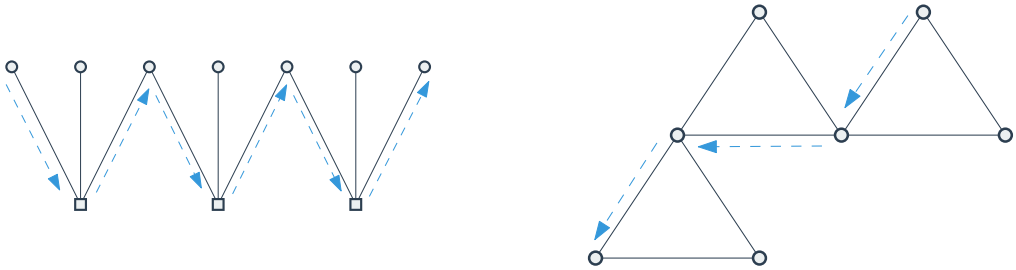


Figure 2 The figure on the left shows a nonbacktracking walk on a subset of a 3-ary constraint graph and the one on the right the same nomadic walk on the corresponding instance graph.

▶ **Definition 20** (Nomadic walk operator). *In the setting of the previous definition, the nomadic walk operator B for $\mathcal{I}$ is defined as follows. Each edge $e = \{u, v\}$ in $\mathcal{I}$ is regarded as two opposing directed edges $\vec{e} = (u, v)$ and $\vec{e}^{-1} = (v, u)$, each having the same edge-weight as e; i.e., $\mathrm{wt}(\vec{e}) = \mathrm{wt}(\vec{e}^{-1}) = \mathrm{wt}(e)$. Let $\vec{E}$ denote the collection of all directed edges. Now B is defined to be the following linear operator on $\ell_2(\vec{E})$:*

$$\text{for } F \in \ell_2(\vec{E}), \quad BF(\vec{e}) = \sum_{\vec{e}'} \mathrm{wt}(\vec{e}')F(\vec{e}'),$$

where the sum is over all directed edges $\vec{e}'$ such that the pair $(\vec{e}, \vec{e}')$ forms a nomadic walk of length-2. In the finite-graph case we also think of B as a matrix; the entry $B[\vec{e}, \vec{e}'] = \mathrm{wt}(\vec{e}')$ whenever $(\vec{e}, \vec{e}')$ is a length-2 nomadic walk. Again, in the case where $r = 2$ and all atoms are single edges, the nomadic walk operator B coincides with the nonbacktracking walk operator. (See, e.g., [5] for more on nonbacktracking walks operators.)

3 Outline of our proof

The utility of the nomadic walk operator is twofold for us. First, for two-eigenvalue CSPs we can relate the eigenvalues of the usual adjacency operator to those of the nomadic walk operator through the following generalization of the Ihara–Bass Formula:

▶ **Theorem 21.** *Let $\mathcal{A}$ be a sequence of atoms such that every atom has the same pair of exactly two distinct eigenvalues, λ_1 and λ_2, and let $\mathcal{H}$ be a constraint graph on variable set V. Let $\mathcal{I} = \mathcal{A}(\mathcal{H})$ be the corresponding instance graph with vertex set V and denote by A and B the adjacency matrix and nomadic walk matrix respectively of $\mathcal{I}$.*

Let $L(t) := \mathbb{1} - At + (\lambda_1 + \lambda_2)t\mathbb{1} + (c-1)(-\lambda_1\lambda_2)t^2$. *Then we have:*

$$(1+\lambda_1 t)^{|V|\frac{c\lambda_2}{\lambda_2-\lambda_1}-1}(1+\lambda_2 t)^{|V|\frac{c\lambda_1}{\lambda_1-\lambda_2}-1}\det L(t) = \det(\mathbb{1} - Bt).$$

We prove Theorem 21 in Section C.

The second utility of nomadic walks is that they provide the key simplification needed to make closed-walk counting in non-tree-like CSPs tractable. Because of this, we are able to establish the following modification of Bordenave's proof of Friedman's Theorem in Section F:

▶ **Theorem 22.** *Let* $\mathcal{A} = (A_1, \dots, A_c)$ *be a sequence of* r*-vertex atoms with edges weights* ± 1. *Let* $|\mathcal{I}_1|$ *denote the instance graph* $\mathcal{A}(K_{r,c})$ *associated to the base constraint graph when the edge-signs are deleted (i.e., converted to* $+1$*), and let* $|B_1|$ *denote the associated nomadic walk matrix. Also, let* $\boldsymbol{\mathcal{H}}_n$ *denote a random* n*-lifted constraint graph and* $\boldsymbol{\mathcal{I}}_n = \mathcal{A}(\boldsymbol{\mathcal{H}}_n)$ *an associated instance graph with* 1*-wise uniform negations* $(\boldsymbol{\xi}^f_{ii'})$. *Finally, let* $\boldsymbol{B}_n$ *denote the nomadic walk matrix for* $\boldsymbol{\mathcal{I}}_n$. *Then for every constant* $\varepsilon > 0$,

$$\mathbf{Pr}[\rho(\boldsymbol{B}_n) \geqslant \sqrt{\rho(|B_1|)} + \varepsilon] \leqslant \delta,$$

where $\delta = \delta(n)$ *is* $o_{n\to\infty}(1)$.

And we can use our version of Ihara–Bass, Theorem 21, to conclude bounds on the spectrum of the adjacency matrix A from Theorem 22, which is worked out in Section D.

▶ **Theorem 23.** *Let* $\boldsymbol{\mathcal{I}}_n$ *be a random additive* n*-lift of* $\mathcal{A}$ *with adjacency matrix* $A_{\boldsymbol{\mathcal{I}}_n}$, *and let* $\epsilon > 0$. *Then:*

$$\mathbf{Pr}\left[\rho(A_{\boldsymbol{\mathcal{I}}_n}) \in [\lambda_1 + \lambda_2 - 2\sqrt{(c-1)(-\lambda_1\lambda_2)} - \varepsilon, \lambda_1 + \lambda_2 + 2\sqrt{(c-1)(-\lambda_1\lambda_2)} + \varepsilon\right] = 1 - o_n(1)$$

Yet another advantage of using nomadic walks instead of closed walks is that in Theorem 23 we are able to bound the left and right spectral edge of $A_{\boldsymbol{\mathcal{I}}_n}$ by *different* values, whereas counting closed walks would, at best, only give an upper bound on $|\lambda|_{\max}(A_{\boldsymbol{\mathcal{I}}_n})$.

Theorem 23 lets us conclude an upper bound on the SDP value, and we complement that with a lower bound via the construction of an SDP solution that nearly matches the upper bound. In particular, we prove the following in Section E.

▶ **Theorem 24.** *For every* $\varepsilon > 0$, *for large enough* n, *there are* $|V(\boldsymbol{\mathcal{I}}_n)| \times |V(\boldsymbol{\mathcal{I}}_n)|$ *positive semidefinite matrices* M_+ *and* M_- *with all-ones diagonals such that*

$$\langle A_{\boldsymbol{\mathcal{I}}_n}, M_+\rangle \geqslant (\lambda_1 + \lambda_2 + 2\sqrt{(c-1)(-\lambda_1\lambda_2)} - \varepsilon)n$$
$$\langle A_{\boldsymbol{\mathcal{I}}_n}, M_-\rangle \leqslant (\lambda_1 + \lambda_2 - 2\sqrt{(c-1)(-\lambda_1\lambda_2)} + \varepsilon)n.$$

with probability $1 - o_n(1)$.

As detailed out in Section G, this lets us conclude the main theorem of this paper:

▶ **Theorem 25.** *For random* c*-constraint-regular instances of a CSP with* 2 *distinct eigenvalues* λ_1 *and* λ_2, *the SDP value is in the range*

$$\frac{\lambda_1 + \lambda_2 + 2\sqrt{(c-1)(-\lambda_1\lambda_2)}}{c(-\lambda_1\lambda_2)} \pm \varepsilon$$

with high probability, for any $\varepsilon > 0$.

Theorem 2 can be viewed as a special case of Theorem 25.

References

1. Scott Aaronson and Andris Ambainis. Forrelation: a problem that optimally separates quantum from classical computing. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing*, pages 307–316, 2015.
2. Noga Alon, Shlomo Hoory, and Nathan Linial. The Moore bound for irregular graphs. *Graphs and Combinatorics*, 18(1):53–57, 2002.
3. Andris Ambainis. Polynomial degree vs. quantum query complexity. *Journal of Computer and System Sciences*, 72(2):220–238, 2006.
4. Andris Ambainis, Krišjānis Prūsis, and Jevgēnijs Vihrovs. Sensitivity versus certificate complexity of Boolean functions. In *Proceedings of the 11th Annual Computer Science Symposium in Russia*, pages 16–28, 2016.
5. Omer Angel, Joel Friedman, and Shlomo Hoory. The non-backtracking spectrum of the universal cover of a graph. *Transactions of the American Mathematical Society*, 367(6):4287–4318, 2015.
6. Benny Applebaum, Boaz Barak, and Avi Wigderson. Public-key cryptography from different assumptions. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing*, pages 171–180, 2010.
7. Alain Aspect, Jean Dalibard, and Gérard Roger. Experimental test of Bell's inequalities using time-varying analyzers. *Physical Review Letters*, 49(25):1804–1807, 1982.
8. Jess Banks, Robert Kleinberg, and Cristopher Moore. The Lovász Theta function for random regular graphs and community detection in the hard regime. In *Proceedings of the 21st Annual International Workshop on Randomized Techniques in Computation*, volume 81, pages 28:1–28:22, 2017.
9. John Bell. On the Einstein Podolsky Rosen paradox. *Physics Physique Fizika*, 1(3):195–200, 1964.
10. Yonatan Bilu and Nathan Linial. Lifts, discrepancy and nearly optimal spectral gap. *Combinatorica. An International Journal on Combinatorics and the Theory of Computing*, 26(5):495–519, 2006.
11. Avrim Blum, Merrick Furst, Michael Kearns, and Richard Lipton. Cryptographic primitives based on hard learning problems. In *Proceedings of the 13th Annual International Cryptography Conference*, pages 278–291, 1993.
12. Charles Bordenave. A new proof of Friedman's second eigenvalue theorem and its extension to random lifts. *arXiv preprint*, 2015. `arXiv:1502.04482`.
13. Charles Bordenave and Benoît Collins. Eigenvalues of random lifts and polynomial of random permutations matrices. *arXiv preprint*, 2018. `arXiv:1801.00876`.
14. Charles Bordenave, Marc Lelarge, and Laurent Massoulié. Non-backtracking spectrum of random graphs: community detection and non-regular Ramanujan graphs. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 1347–1357. IEEE, 2015.
15. Mark Braverman, Konstantin Makarychev, Yury Makarychev, and Assaf Naor. The Grothendieck constant is strictly smaller than Krivine's bound. *Forum of Mathematics. Pi*, 1:e4, 42, 2013.
16. Gerandy Brito, Ioana Dumitriu, and Kameron Decker Harris. Spectral gap in random bipartite biregular graphs and its applications. *arXiv preprint*, 2018. `arXiv:1804.07808`.
17. Moses Charikar and Anthony Wirth. Maximizing quadratic programs: extending Grothendieck's Inequality. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 54–60, 2004.
18. John Clauser, Michael Horne, Abner Shimony, and Richard Holt. Proposed experiment to test local hidden-variable theories. *Physical Review Letters*, 23(15):880–884, 1969.
19. Richard Cleve, Peter Høyer, Benjamin Toner, and John Watrous. Consequences and limits of nonlocal strategies. In *Proceedings of the 19th Annual Computational Complexity Conference*, pages 246–249, 2004.

20 Endre Csóka, Balázs Gerencsér, Viktor Harangi, and Bálint Virág. Invariant Gaussian processes and independent sets on regular graphs of large girth. *Random Structures & Algorithms*, 47(2):284–303, 2015.

21 Amit Daniely and Shai Shalev-Shwartz. Complexity theoretic limitations on learning DNF's. In *Proceedings of the 29th Annual Conference on Learning Theory*, pages 815–830, 2016.

22 Charles Delorme and Svatopluk Poljak. Laplacian eigenvalues and the maximum cut problem. *Mathematical Programming*, 62(1–3):557–574, 1993.

23 Amir Dembo, Andrea Montanari, and Subhabrata Sen. Extremal cuts of sparse random graphs. *The Annals of Probability*, 45(2):1190–1217, 2017.

24 Yash Deshpande, Andrea Montanari, Ryan O'Donnell, Tselil Schramm, and Subhabrata Sen. The threshold for SDP-refutation of random regular NAE-3SAT. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2305–2321, 2019.

25 Jian Ding, Allan Sly, and Nike Sun. Proof of the satisfiability conjecture for large k. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing*, pages 59–68, 2015.

26 Yehonatan Elon. Gaussian waves on the regular tree. Technical report, arXiv, 2009. `arXiv:0907.5065`.

27 Uriel Feige. Relations between average case complexity and approximation complexity. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 543–543, 2002.

28 Uriel Feige and Gideon Schechtman. On the optimality of the random hyperplane rounding technique for Max-Cut. *Randoom Structures and Algorithms*, 20(3):403–440, 2002.

29 Joel Friedman. A proof of Alon's second eigenvalue conjecture and related problems. *Memoirs of the American Mathematical Society*, 195(910):viii+100, 2008.

30 Michel Goemans and David Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:1115–1145, 1995.

31 Oded Goldreich. Candidate one-way functions based on expander graphs. Technical Report 90, Electronic Colloquium on Computational Complexity, 2000.

32 Rostislav Grigorchuk and Andrzej Żuk. On the asymptotic spectrum of random walks on infinite families of graphs. In *Random walks and discrete potential theory (Cortona, 1997)*, Sympos. Math., XXXIX, pages 188–204. Cambridge Univ. Press, Cambridge, 1999.

33 Alexander Grothendieck. Résumé de la théorie métrique des produits tensoriels topologiques. *Boletín de la Sociedad Matemática São Paulo*, 8:1–79, 1953.

34 Viktor Harangi and Bálint Virág. Independence ratio and random eigenvectors in transitive graphs. *The Annals of Probability*, 43(5):2810–2840, 2015.

35 Ari Juels and Marcus Peinado. Hiding cliques for cryptographic security. *Designs, Codes and Cryptography*, 20(3):269–280, 2000.

36 Pravesh Kothari, Ryuhei Mori, Ryan O'Donnell, and David Witmer. Sum of squares lower bounds for refuting any CSP. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing*, pages 132–145, 2017.

37 Florent Krzakala, Cristopher Moore, Elchanan Mossel, Joe Neeman, Allan Sly, Lenka Zdeborová, and Pan Zhang. Spectral redemption in clustering sparse networks. *Proceedings of the National Academy of Sciences of the United States of America*, 110(52):20935–20940, 2013.

38 Carlos S Kubrusly. *Spectral theory of operators on Hilbert spaces*. Springer Science & Business Media, 2012.

39 Huijia Lin. Indistinguishability obfuscation from SXDH on 5-linear maps and locality-5 PRGs. In *Proceedings of the 37th Annual International Cryptography Conference*, pages 599–629, 2017.

40 László Lovász. On the Shannon capacity of a graph. *Institute of Electrical and Electronics Engineers. Transactions on Information Theory*, 25(1):1–7, 1979.

41 Russell Lyons. Factors of IID on trees. *Combinatorics, Probability and Computing*, 26(2):285–300, 2017.

42 Raffaele Marino, Giorgio Parisi, and Federico Ricci-Tersenghi. The backtracking survey propagation algorithm for solving random k-sat problems. *Nature Communications*, 7:12996, 2016.

43 Laurent Massoulié. Community detection thresholds and the weak Ramanujan property. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 694–703. ACM, 2014.

44 Marc Mézard and Andrea Montanari. *Information, physics, and computation.* Oxford University Press, 2009.

45 Sidhanth Mohanty and Ryan O'Donnell. X-Ramanujan graphs, 2018. Available at `arXiv:1904.03500`.

46 Andrea Montanari. Bounds on ground state enery in the Sherrington–Kirkpatrick model, 2017. Open problem from AIM workshop, available at `http://aimpl.org/phaserandom/1/`.

47 Andrea Montanari. Optimization of the Sherrington–Kirkpatrick hamiltonian. *arXiv preprint*, 2018. `arXiv:1812.10897`.

48 Andrea Montanari and Subhabrata Sen. Semidefinite programs on sparse random graphs and their application to community detection. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing*, pages 814–827, 2016.

49 Elchanan Mossel, Joe Neeman, and Allan Sly. A proof of the block model threshold conjecture. *Combinatorica*, 38(3):665–708, 2018.

50 Sam Northshield. Several Proofs of Ihara's theorem, 1997.

51 Ryan O'Donnell, Xiaorui Sun, Li-Yang Tan, John Wright, and Yu Zhao. A composition theorem for parity kill number. In *Proceedings of the 29th Annual Computational Complexity Conference*, pages 144–154, 2014.

52 Prasad Raghavendra, Satish Rao, and Tselil Schramm. Strongly refuting random CSPs below the spectral threshold. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing*, pages 121–131, 2017.

53 Farzaneh Ramezani. On the signed graphs with two distinct eigenvalues. *arXiv preprint*, 2015. `arXiv:1511.03511`.

54 Subhabrata Sen. Optimization on sparse random hypergraphs and spin glasses. *Random Structures & Algorithms*, 53(3):504–536, 2018.

55 Michel Talagrand. The Parisi formula. *Annals of Mathematics. Second Series*, 163(1):221–263, 2006.

56 Boris Tsirelson. Quantum generalizations of Bell's inequality. *Letters in Mathematical Physics*, 4(2):93–100, 1980.

57 Ryan Williams. *Algorithms and resource requirements for fundamental problems.* PhD thesis, Carnegie Mellon University, 2007.

58 Wolfgang Woess. *Random walks on infinite graphs and groups*, volume 138. Cambridge university press, 2000.

A Relationship to the work of Bordenave–Collins

The paper of Bordenave–Collins[13] establishes a Friedman/Bordenave theorem for large random graphs whose adjacency matrices are noncommutative polynomials in a fixed number of independent random matching matrices and permutation matrices (together with their transposes). As a most basic example, it recovers the following form of Friedman's Theorem: whp, the sum of d random perfect matchings has all nontrivial eigenvalues bounded in magnitude by $\rho(\mathbb{Z}_2 * \cdots (d \text{ times}) \cdots * \mathbb{Z}_2) + o_n(1) = 2\sqrt{d-1} + o_n(1)$. However, the Bordenave–Collins work gives much more than this. For example, let $\boldsymbol{G}$ be the n-vertex graph formed as

$$\boldsymbol{P} + \boldsymbol{P}^\top + \boldsymbol{M} - \boldsymbol{P}\boldsymbol{M}\boldsymbol{P}^\top,$$

where $\boldsymbol{M}$ is a random matching matrix and and $\boldsymbol{P}$ is an independent random permutation matrix. It is not hard to see that $\boldsymbol{G}$ will essentially "locally resemble" a 2-constraint-regular Sort_4-CSP instance. And, the Bordenave–Collins work implies that the eigenvalues of $\boldsymbol{G}$ are bounded (whp) by $\rho(\mathsf{Sort}_4 \dashv\vdash \mathsf{Sort}_4)$. Using the theory of free probability, it is possible to directly compute that $\rho(\mathsf{Sort}_4 \dashv\vdash \mathsf{Sort}_4) = 2\sqrt{2}$. In this way, our Theorem 23 in the case of 2-constraint-regular Sort_4-CSPs is covered by Bordenave and Collins. Indeed, it is not hard to generalize this example to the case of c-constraint-regular Sort_4-CSPs for any *even* integer c.

Indeed, the Bordenave–Collins work also treats some kinds of graphs that our work cannot; for example, Wu gave the example when $\boldsymbol{G}$ is the n-vertex graph generated by the polynomial

$$\boldsymbol{P}_1 + \boldsymbol{P}_1^\top + \boldsymbol{P}_2 + \boldsymbol{P}_2^\top + \boldsymbol{P}_3 + \boldsymbol{P}_3^\top + \boldsymbol{P}_4 + \boldsymbol{P}_4^\top + \boldsymbol{P}_1\boldsymbol{P}_2\boldsymbol{P}_3\boldsymbol{P}_4 + \boldsymbol{P}_4^\top\boldsymbol{P}_3^\top\boldsymbol{P}_2^\top\boldsymbol{P}_1^\top,$$

where $\boldsymbol{P}_1, \dots, \boldsymbol{P}_4$ are independent uniformly random permutation matrices. This $\boldsymbol{G}$ "locally resembles" the infinite free product graph $X = \mathbb{Z}_4 * \mathbb{Z}_4 * \mathbb{Z}_4 * \mathbb{Z}_4$, and the Bordenave–Collins work implies that whp, $\boldsymbol{G}$'s nontrivial eigenvalues are bounded in magnitude by $\rho(X) + o_n(1)$. (We remark that computing the numeric value of this $\rho(X)$ is difficult, but possible; see, e.g., [58, Ch. 9C]). Since the 4-cycle graph $\mathbb{Z}_4$ has more than two distinct eigenvalues, it is not covered by our work.

This said, the Bordenave–Collins work does not subsume our Theorem 23, as there are plenty of graph families that our theorem handles but Bordenave–Collins's does not (seem to). For example, Wu has sketched to us a proof that one cannot obtain c-constraint-regular Sort_4 instances for *odd* c through any straightforward use of [13]. Additionally, even in the cases of interest to us where Bordenave–Collins applies, we can point to some (minor) advantages of our methods. For one, our model of random graph generation clearly corresponds to precisely-regular CSP instances, whereas in the Bordenave–Collins model there will be (in expectation) a constant number of local "blemishes" where one cannot interpret a piece of the graph as a constraint. For another, our work directly yields the numerical values of the appropriate spectral radii $\rho(X)$ (though in the cases where our results apply, these can be obtained through standard methods in free probability).

B Further preliminaries

B.1 Quantum games, and some quantum-relevant constraints

In the case when the underlying graph G is bipartite, $\mathrm{SDP}(\mathcal{I})$ has another important interpretation: it is the true *quantum* value of the 2-player 1-round "nonlocal game" associated to $\mathcal{I}$. We give definitions below, but let us mention that the Sort_4 (equivalently, CHSH) and $\mathsf{Forrelation}_k$ constraints from Theorem 1 and Theorem 2 are both: (a) bipartite; (b) directly inspired by quantum theory. Thus those two theorems can be interpreted as determining the true quantum value of random c-constraint-regular nonlocal games based on CHSH and $\mathsf{Forrelation}_k$.

Let us now recall the relevant quantum facts.

▶ **Definition 26** (Nonlocal 2XOR games)**.** *Given a* 2XOR *instance* $\mathcal{I} = (G, \mathrm{wt})$ *with* $G = (U, V, E)$ *bipartite, the associated nonlocal (*2XOR*) game is the following. There are spatially separated players Alice and Bob. A referee chooses* $e = (u, v) \in E$ *uniformly at random, tells* u *to Alice, and tells* v *to Bob. Without communicating, Alice and Bob are required to respond with signs* $x_u, y_v \in \{\pm 1\}$*. The value to the players is the expected value of* $\mathrm{wt}(e)x_u y_v$*. It is easy to see that if Alice and Bob are deterministic, or are allowed classical shared randomness, then the optimum value they can achieve is precisely* $\mathrm{OPT}(\mathcal{I})$*.*

▶ **Theorem 27** ([19, 56]). *In a nonlocal* 2XOR *game, if Alice and Bob are allowed to share unlimited quantumly entangled particles, then the optimal value they can achieve is precisely* $\mathrm{SDP}(\mathcal{I})$.

The fact that there exist bipartite edge-signed $\mathcal{I}$ for which $\mathrm{SDP}(\mathcal{I}) > \mathrm{OPT}(\mathcal{I})$ is foundational for the experimental verification of quantum mechanics, as the following example attests:

▶ **Example 28.** Consider the 2XOR instance depicted in Figure 3, called CHSH after Clauser, Horne, Shimony, and Holt [18]. It has

$$\mathrm{OPT}(\mathsf{CHSH}) = 1/2 < 1/\sqrt{2} = \mathrm{SDP}(\mathsf{CHSH}).$$

The upper bound $4 \cdot \mathrm{OPT}(\mathsf{CHSH}) \leqslant 2$ is often called *Bell's inequality* [9], and the higher lower

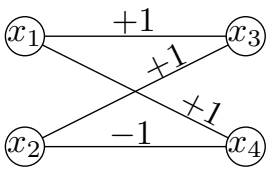


Figure 3 The CHSH game/CSP.

bound $1/\sqrt{2} \leqslant \mathrm{SDP}(\mathsf{CHSH})$ is from [18] (with $\mathrm{SDP}(\mathsf{CHSH}) \leqslant 1/\sqrt{2}$ due to Tsirelson [56]). Aspect and others [7] famously experimentally realized this gap between what can be achieved with classical vs. quantum resources.

In fact, the CHSH instance is nothing more than the Sort_4 predicate in disguise! More precisely (cf. (2)),

$$\mathsf{CHSH}(x_1, x_2, x_3, x_4) = \tfrac{1}{4}(x_1x_3 + x_2x_3 + x_1x_4 - x_2x_4) = \mathsf{Sort}_4(x_2, x_3, x_1, x_4) - \tfrac{1}{2}.$$

Thanks to its degree-2 Fourier expansion, CSPs based on the Sort_4/CHSH constraint have been studied in a variety of contexts, including concrete complexity [3, 4, 51] and fixed parameter algorithms [57].

Though Sort_4 is a "predicate", in the sense that it takes 0/1 (unsat/sat) values, there's nothing necessary about basing a large CSP on predicates. An interesting family of constraints that can be modeled by 2XOR optimization, originally arising in quantum complexity theory [1], is the family of "Forrelation" functions. For any $k \in \mathbb{N}$, the $\mathsf{Forrelation}_k$ function is defined by

$$\mathsf{Forrelation}_k : \{\pm 1\}^{2^k} \times \{\pm 1\}^{2^k} \to [-1, +1], \qquad \mathsf{Forrelation}_k(x_1, \ldots, x_{2^k}, y_1, \ldots, y_{2^k}) = 2^{-2k} x^\top H_k y,$$

where $H_k = \begin{pmatrix} +1 & +1 \\ +1 & -1 \end{pmatrix}^{\otimes k}$ is the kth Walsh–Hadamard matrix. Note that $\mathsf{Forrelation}_0$ corresponds to the single-(positive-)edge 2XOR CSP, and $\mathsf{Forrelation}_1$ is CHSH.

B.2 Operator Theory

The results in this section can be found in a standard textbook on functional analysis or operator theory (see, for e.g. [38]).

Let V be an some countable set and let $T : \ell_2(V) \to \ell_2(V)$ be a bounded, self-adjoint linear operator.

▶ **Definition 29.** *We refer to the spectrum of T,* $\mathrm{Spec}(T)$*, as the set of all complex λ such that $\lambda\mathbb{1} - T$ is not invertible.* $\mathrm{Spec}(T)$ *is a nonempty, compact set.*

▶ **Definition 30.** *We call λ an approximate eigenvalue of T if for every $\varepsilon > 0$, there is unit x in $\mathcal{X}$ such that $\|Tx - \lambda x\| \leqslant \varepsilon$. We call such an x an ε-approximate eigenvector or ε-approximate eigenfunction.*

▶ **Theorem 31.** *If T is a self-adjoint operator, then every $\lambda \in \mathrm{Spec}(T)$ is an approximate eigenvalue.*

▶ **Theorem 32** (Consequence of Proposition 4.L of [38])**.** *If λ is an isolated point in* $\mathrm{Spec}(T)$*, then it is an eigenvalue of T, i.e., it is a 0-approximate eigenvalue.*

▶ **Corollary 33.** $\lambda_{\min} := \min\{\mathrm{Spec}(T)\}$ *and* $\lambda_{\max} := \max\{\mathrm{Spec}(T)\}$ *are both approximate eigenvalues of T.*

▶ **Fact 34.** *Additionally,*

$$\lambda_{\min}(T) = \inf_{\|x\|=1} \langle x, Tx\rangle,$$
$$\lambda_{\max}(T) = \sup_{\|x\|=1} \langle x, Tx\rangle.$$

▶ **Definition 35.** *The spectral radius $\rho(T)$ is defined as* $\max_{\sigma\in\mathrm{Spec}(T)} |\sigma|$.

▶ **Definition 36.** *The operator norm of T, denoted $\|T\|_{\mathrm{op}}$, is defined as*

$$\sup_{\|x\|=1,\|y\|=1} \langle y, Tx\rangle = \sup_{\|x\|=1} \|Tx\|.$$

▶ **Fact 37.** $\rho(T) = \lim_{k\to\infty} \|T^k\|_{\mathrm{op}}^{1/k}$.

C An Ihara–Bass formula for additive lifts of 2-eigenvalue atoms

Let $\mathcal{A}$ be a sequence of atoms such that every atom has the same pair of exactly two distinct eigenvalues, λ_1 and λ_2, and let $\mathcal{H}$ be a constraint graph on variable set V. Let $\mathcal{I} = \mathcal{A}(\mathcal{H})$ be the corresponding instance graph. In this section, we use A and B to refer to the adjacency matrix and nomadic walk matrix respectively of $\mathcal{I}$. The vertex set of $\mathcal{I}$ is V. This section is devoted to proving our generalization of the Ihara–Bass formula, stated below.

▶ **Theorem 38.** *Let $L(t) := \mathbb{1} - At + (\lambda_1 + \lambda_2)t\mathbb{1} + (c-1)(-\lambda_1\lambda_2)t^2\mathbb{1}$. Then we have*

$$(1+\lambda_1 t)^{|V|\frac{c\lambda_2}{\lambda_2-\lambda_1}-1}(1+\lambda_2 t)^{|V|\frac{c\lambda_1}{\lambda_1-\lambda_2}-1} \det L(t) = \det(\mathbb{1} - Bt).$$

Our proof is a modification of one of the proofs of the Ihara–Bass formula from [50].

Nomadic Polynomials. Our first step is to define the following sequence of polynomials.

$$p_0(x) = 1$$
$$p_1(x) = x$$
$$p_2(x) = x^2 - (\lambda_1+\lambda_2)x - c(-\lambda_1\lambda_2)$$
$$p_k(x) = xp_{k-1}(x) - (\lambda_1+\lambda_2)p_{k-1}(x) - (c-1)(-\lambda_1\lambda_2)p_{k-2}(x) \qquad \text{for } k \geqslant 3$$

and introduce the key player in the proof: the matrix of generating functions $F(t)$ defined by

$$F(t)_{u,v} = \sum_{k\geqslant 0} p_k(A)_{uv}t^k.$$

We use $\mathrm{wt}(e)$ to denote the weight on edge e, and define the weight of a walk $W = e_1 e_2 \dots e_\ell$ as

$$\mathrm{wt}(W) := \prod_{i=1}^{\ell} \mathrm{wt}(e_i).$$

We first establish combinatorial meaning for the polynomials $p_k(A)$.

▷ Claim 39. $p_k(A)_{uv}$ is equal to the total weight of nomadic walks of length k from u to v.

Proof. When $k = 0$ and 1, the claim is clear. We proceed by induction.

Supposing the claim is indeed true for $p_s(A)$ when $s \leqslant k-1$, then $Ap_{k-1}(A)_{uv}$ is the total weight of length-k walks from u to v whose first $k-1$ steps are nomadic and whose last step is arbitrary. Call the collection of these walks $\mathcal{W}_{uv}$. For $W \in \mathcal{W}_{uv}$, let W_i denote the edge walked on by the i-th step of W and let $W_{(i)}$ denote the length-i walk obtained by taking the length-i prefix of W. We use lowercase w_i to denote the vertex visited by the ith step of the walk. Each $W \in \mathcal{W}_{uv}$ falls into one of the following three categories.

1. W is a nomadic walk. Call the collection of these walks $\mathcal{W}^{(1)}_{uv}$.
2. $W_k = W_{k-1}^{-1}$. Call the collection of these walks $\mathcal{W}^{(2)}_{uv}$.
3. W_{k-1} and W_k are in the same atom but $W_k \neq W_{k-1}^{-1}$. Call the collection of these walks $\mathcal{W}^{(3)}_{uv}$.

Suppose $k \geqslant 3$.

$$\begin{aligned}
\sum_{W \in \mathcal{W}^{(2)}_{uv}} \mathrm{wt}(W) &= \sum_{W \in \mathcal{W}^{(2)}_{uv}} \mathrm{wt}(W_{k-1})\mathrm{wt}(W_{k-1}^{-1})\mathrm{wt}(W_{(k-2)}) \\
&= \sum_{W \in \mathcal{W}^{(2)}_{uv}} \mathrm{wt}(W_{k-1})^2 \mathrm{wt}(W_{(k-2)}) \\
&= \sum_{\substack{W' \ (k-2)\text{-length nomadic walk} \\ \text{from } u \text{ to } v}} \mathrm{wt}(W') \sum_{e \notin \mathrm{Atom}(W'_{k-2})} \mathrm{wt}(e)^2
\end{aligned}$$

We apply Fact 9 and get

$$\begin{aligned}
&= \sum_{\substack{W' \ (k-2)\text{-length nomadic walk} \\ \text{from } u \text{ to } v}} \mathrm{wt}(W')(c-1)(-\lambda_1\lambda_2) \\
&= (c-1)(-\lambda_1\lambda_2)p_{k-2}(A)_{uv}.
\end{aligned}$$

An identical argument shows that when $k = 2$,

$$\sum_{W \in \mathcal{W}^{(2)}_{uv}} \mathrm{wt}(W) = c(-\lambda_1\lambda_2)$$

We do a similar calculation for $\mathcal{W}^{(3)}_{uv}$ for $k \geqslant 2$. Observe that W_{k-1} and W_k have to be in the same atom, which we denote $\mathrm{Atom}(W_{k-1})$. Thus, there is an edge e^* between w_{k-2} and v in $\mathrm{Atom}(W_{k-1})$ too (see Remark 14).

$$\sum_{W\in\mathcal{W}_{uv}^{(3)}} \mathrm{wt}(W)$$
$$= \sum_{W\in\mathcal{W}_{uv}^{(3)}} \mathrm{wt}(W_{k-1})\mathrm{wt}(W_k)\mathrm{wt}(W_{(k-2)})$$
$$= \sum_{\substack{W' \text{ length-}(k-2) \text{ nomadic walk} \\ W'_0=u, \\ e^* \text{ s.t. } (e^*)_1=w_{k-2},(e^*)_2=v \\ \mathrm{Atom}(W'_{k-2})\neq\mathrm{Atom}(e^*)}} \sum_{\substack{e^{(1)},e^{(2)}: \\ \mathrm{Atom}(e^{(1)})=\mathrm{Atom}(e^{(2)})=\mathrm{Atom}(e^*) \\ (e^{(1)})_1=w_{k-2},(e^{(1)})_2=(e^{(2)})_1,(e^{(2)})_2=v}} \mathrm{wt}(e^{(1)})\mathrm{wt}(e^{(2)})\mathrm{wt}(W')$$

By applying Fact 10, we get

$$= \sum_{\substack{W' \text{ length-}(k-2) \text{ nomadic walk} \\ W'_0=u, \\ e^* \text{ s.t. } (e^*)_1=w_{k-2},(e^*)_2=v \\ \mathrm{Atom}(W'_{k-2})\neq\mathrm{Atom}(e^*)}} (\lambda_1+\lambda_2)\mathrm{wt}(e^*)\mathrm{wt}(W')$$
$$= (\lambda_1+\lambda_2) \sum_{W' \text{ length-}(k-1) \text{ nomadic walk from } u \text{ to } v} \mathrm{wt}(W')$$
$$= (\lambda_1+\lambda_2)p_{k-1}(A)_{uv}.$$

Now, we have for $k \geqslant 3$,

$$\sum_{W\in\mathcal{W}_{uv}} \mathrm{wt}(W) = \sum_{W\in\mathcal{W}_{uv}^{(1)}} \mathrm{wt}(W) + \sum_{W\in\mathcal{W}_{uv}^{(2)}} \mathrm{wt}(W) + \sum_{W\in\mathcal{W}_{uv}^{(3)}} \mathrm{wt}(W)$$
$$Ap_{k-1}(A)_{uv} = \sum_{W\in\mathcal{W}_{uv}^{(1)}} \mathrm{wt}(W) + (c-1)(-\lambda_1\lambda_2)p_{k-2}(A)_{uv} + (\lambda_1+\lambda_2)p_{k-1}(A)_{uv}$$
$$\sum_{W\in\mathcal{W}_{uv}^{(1)}} \mathrm{wt}(W) = Ap_{k-1}(A)_{uv} - ((c-1)(-\lambda_1\lambda_2)p_{k-2}(A)_{uv} + (\lambda_1+\lambda_2)p_{k-1}(A)_{uv})$$
$$\sum_{W\in\mathcal{W}_{uv}^{(1)}} \mathrm{wt}(W) = p_k(A)_{uv}.$$

For the case of $k=2$, we carry out the above calculation by replacing $(c-1)(-\lambda_1\lambda_2)$ with $c(-\lambda_1\lambda_2)$, thus completing the inductive step. ◁

Generic generating functions facts. Before returning to the specifics of our problem, we give some "standard" generating function facts. These are extensions of the following simple idea: if $f(t)$ is a polynomial, then $\frac{d}{dt}\log f(t) = f'(t)\cdot f(t)^{-1}$ is (up to minor manipulations) the generating function for the power sum polynomials of its roots. We start with a general matrix version of this, which is sometimes called *Jacobi's formula* (after minor manipulations):

▶ **Proposition 40.** *Let $M(t)$ be a square matrix polynomial of t. Then*

$$\frac{d}{dt}\log\det M(t) = \mathrm{tr}\big(M'(t)M(t)^{-1}\big)$$

for all $t\in\mathbb{R}$ such that $M(t)$ is invertible.

▶ **Corollary 41.** *Taking $M(t) = \mathbb{1} - Ht$ for a fixed square matrix H yields*

$$\frac{d}{dt}\log\det(\mathbb{1}-Ht) = \mathrm{tr}\big(-H(\mathbb{1}-Ht)^{-1}\big) \quad\Longrightarrow\quad -t\frac{d}{dt}\log\det(\mathbb{1}-Ht) = \sum_{k\geqslant 1}\mathrm{tr}(H^k)t^k.$$

Regarding this corollary, we can derive the statement about the power sums of the roots of a polynomial $f(t)$ by taking $H = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ where the λ_i's are the roots of f. On the other hand, it actually suffices to prove Corollary 41 in the case of diagonal H, since $\det(\mathbb{1} - Ht)$ is invariant to unitary conjugation.

Growth Rate. A key term that shows up in our Ihara–Bass formula is the "growth rate" of the additive product of $\mathcal{A}$. Suppose we take t-step nomadic walk starting at a vertex v in the additive product graph, take a t-step nomadic walk back to v, and then sum over the total weight of such walks. What we get is $((c-1)(-\lambda_1\lambda_2))^t$ (see Lemma 49 for a proof). Thus, the total weight of aforementioned walks grows exponentially in t at a rate of $(c-1)(-\lambda_1\lambda_2)$, which in this section we will refer to as α_{gr}.

The fundamental recurrence. We now relate the generating function matrix $F(t)$ to A. Using the recurrence used to generated the polynomials $p_k(x)$, one can conclude

▶ **Lemma 42.** $F(t) = AF(t)t - (\lambda_1 + \lambda_2)F(t)t - \alpha_{\mathrm{gr}}F(t)t^2 + (1 + t\lambda_1)(1 + t\lambda_2)\mathbb{1}$.

From this recurrence one may express the inverse of $F(t)$ in terms of A and c:

▶ **Corollary 43.** $(1 + \lambda_1 t)^{-1}(1 + \lambda_2 t)^{-1} \cdot (\mathbb{1} - At + (\lambda_1 + \lambda_2)t\mathbb{1} + \alpha_{\mathrm{gr}}t^2\mathbb{1})F(t) = \mathbb{1}$. *In other words,* $F(t) = (1 + \lambda_1 t)(1 + \lambda_2 t)\mathbb{1} \cdot L(t)^{-1}$, *where* $L(t) \coloneqq \mathbb{1} - At + (\lambda_1 + \lambda_2)t\mathbb{1} + \alpha_{\mathrm{gr}}t^2\mathbb{1}$ *is the "deformed Laplacian" appearing in the statement of our Ihara–Bass theorem.*

Strategy for the rest of the proof. The strategy will be to apply Proposition 40 with the deformed Laplacian $L(t)$. On the left side we'll get a determinant involving A. On the right side we'll get a trace involving $L(t)^{-1}$, which is essentially $F(t)$. In turn, $\mathrm{tr}(F(t))$ is a generating function for nomadic closed walks, which we can hope to relate to B (although there will be an edge case to deal with).

Let's begin executing this strategy. By Proposition 40 we have

$$\begin{aligned} -t\frac{d}{dt}\log\det L(t) &= -t \cdot \mathrm{tr}\big(L'(t)L(t)^{-1}\big) \\ &= -t \cdot \mathrm{tr}\big((\mathbb{1}(\lambda_1 + \lambda_2) - A + 2\alpha_{\mathrm{gr}}t\mathbb{1}) \cdot ((1 + \lambda_1 t)(1 + \lambda_2 t))^{-1}F(t)\big) \\ &= \frac{1}{(1 + \lambda_1 t)(1 + \lambda_2 t)}\,\mathrm{tr}\big(-(\lambda_1 + \lambda_2)F(t)t + AF(t)t - 2\alpha_{\mathrm{gr}}F(t)t^2\big) \end{aligned}$$

where we used Corollary 43. Now using Lemma 42 again we may infer

$$-(\lambda_1 + \lambda_2)F(t)t + AF(t)t - 2\alpha_{\mathrm{gr}}F(t)t^2 = (1 - \alpha_{\mathrm{gr}}t^2)F(t) - (1 + \lambda_1 t)(1 + \lambda_2 t)\mathbb{1};$$

combining the previous two identities yields

$$-t\frac{d}{dt}\log\det L(t) = \mathrm{tr}\left(\frac{1 - \alpha_{\mathrm{gr}}t^2}{(1 + \lambda_1 t)(1 + \lambda_2 t)}F(t) - \mathbb{1}\right). \tag{3}$$

Nomadic walks. The right side above is $\mathrm{tr}(F(t))$ up to some scaling/translating. By definition, $\mathrm{tr}(F(t))$ is the generating function for nomadic *circuits* (closed walks) with any starting point. A first instinct is therefore to expect that

$$\mathrm{tr}(F(t)) \stackrel{?}{=} \sum_{k \geqslant 0} \mathrm{tr}(B^k)t^k, \tag{4}$$

as $\mathrm{tr}(B^k)$ is the weight of closed length-k circuits of direct edges in the nomadic world. However this is not quite right: $\mathrm{tr}(B^k)$ only weighs the nomadic circuits whose first and last edge are not in the same atom. The nomadic circuits that are not weighed can be identified either as (i) "tailed" nomadic circuits, i.e., those where the last directed edge is the reverse of the first directed edge; (ii) "stretched" nomadic circuits, i.e., those where the last directed edge is distinct from but in the same atom as the first directed edge. E.g., $\mathrm{tr}(B^k)$ would fail to count the following:

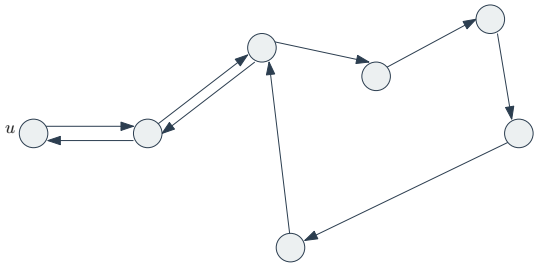


Figure 4 A length-9 nomadic walk from u to u with a *tail* of length 2.

Thus we need to correct (4).

▶ **Definition 44.** *With the $-\mathbb{1}$ taking care of the omission of $k = 0$, we define*

$$\mathrm{Tails}(t) = \sum_{k\geqslant 1}(\textit{weight of nomadic circuits of length } k)t^k = \mathrm{tr}(F(t) - \mathbb{1}). \tag{5}$$

We also define

$$\mathrm{NoTails}(t) = \sum_{k\geqslant 1}(\textit{weight of tail-less nomadic circuits of length } k)t^k$$

and

$$\begin{aligned}\mathrm{Simple}(t) &= \sum_{k\geqslant 1}(\textit{weight of non-stretched, tail-less nomadic circuits of length } k)t^k \\ &= \sum_{k\geqslant 1}\mathrm{tr}(B^k)t^k = -t\frac{d}{dt}\log\det(\mathbb{1} - Bt),\end{aligned} \tag{6}$$

where the last equality used Corollary 41.

Tails vs. no tails vs. simple: more generating functions. We finish by relating $\mathrm{Tails}(t)$, $\mathrm{NoTails}(t)$ and $\mathrm{Simple}(t)$. This is the recipe:

> A general nomadic circuit of length k is constructed from a tail-less nomadic circuit of length $k - 2\ell$ with a tail of length-ℓ attached to one of its vertices.

Tail-less nomadic circuits can be classified as (i) non-stretched tail-less nomadic circuits, and (ii) stretched, tail-less nomadic circuits, for which,

$$\mathrm{NoTails}(t) - \mathrm{Simple}(t) = \sum_{k\geqslant 1}(\text{weight of stretched, tail-less nomadic walks of length } k)t^k.$$

Consider a stretched, tail-less nomadic walk of length k that starts at vertex v, takes the edge e from v to u, goes on a nomadic walk W from u to w, and finally takes edge e' from w to v to end the walk at v. Note that e and e' are part of the same atom A_i. Summing over all v in atom A_i and applying Fact 10 gives

$$\sum_{v\sim A_i} \mathrm{wt}(A_i(v,u))\mathrm{wt}(A_i(w,v))\mathrm{wt}(W) = (\lambda_1+\lambda_2)\mathrm{wt}(A_i(w,u))\mathrm{wt}(W) = (\lambda_1+\lambda_2)\mathrm{wt}(W')$$

where W' is a nomadic circuit of length $k-1$ that starts at w, takes edge $A_i(w,u)$ in the first step, and then takes walk W. From this, we derive

$$\mathrm{NoTails}(t) - \mathrm{Simple}(t) = (\lambda_1+\lambda_2)t \cdot \mathrm{Simple}(t).$$

It's easy to count the total weight of tails of length ℓ one can attach to a given vertex of a tail-less nomadic circuit: if the tail-less nomadic circuit is non-stretched, the first edge can be chosen by picking any edge in $(c-2)$ atoms and each of the remaining $\ell-1$ edges can be chosen by picking any edge $(c-1)$ atoms; and if the tail-less nomadic circuit is stretched, each edge (including the first one) can be chosen anywhere from $(c-1)$ atoms. From this it's easy to derive

$$\begin{aligned}\mathrm{Tails}(t) &= \big(1+(-\lambda_1\lambda_2)(c-2)t^2+(-\lambda_1\lambda_2)^2(c-2)(c-1)t^4+\cdots\big)\mathrm{Simple}(t) \\ &\quad + \big(1+(-\lambda_1\lambda_2)(c-1)t^2+(-\lambda_1\lambda_2)^2(c-1)^2t^4+\cdots\big)(\mathrm{NoTails}(t)-\mathrm{Simple}(t)) \\ &= \frac{1-(-\lambda_1\lambda_2)t^2}{1-(c-1)(-\lambda_1\lambda_2)t^2}\mathrm{Simple}(t) + \frac{(\lambda_1+\lambda_2)t}{1-(c-1)(-\lambda_1\lambda_2)t^2}\mathrm{Simple}(t) \\ &\iff \mathrm{Simple}(t) = \frac{1-\alpha_{\mathrm{gr}}t^2}{(1+\lambda_1 t)(1+\lambda_2 t)}\mathrm{Tails}(t).\end{aligned} \tag{7}$$

Using $\mathrm{Tails}(t) = \mathrm{tr}(F(t)-\mathbb{1})$ (i.e., (5)), we obtain:

▶ **Corollary 45.** $\mathrm{Simple}(t) = \mathrm{tr}\left(\frac{1-\alpha_{\mathrm{gr}}t^2}{(1+\lambda_1 t)(1+\lambda_2 t)}(F(t)-\mathbb{1})\right)$.

But this is *almost* the same as (3). The difference is

$$\begin{aligned}\mathrm{tr}\left(\mathbb{1} - \frac{1-\alpha_{\mathrm{gr}}t^2}{(1+\lambda_1 t)(1+\lambda_2 t)}\mathbb{1}\right) &= \mathrm{tr}\left(\frac{(\lambda_1+\lambda_2)t+(c-2)(-\lambda_1\lambda_2)t^2}{(1+\lambda_1 t)(1+\lambda_2 t)}\mathbb{1}\right) \\ &= |V|\cdot\frac{(\lambda_1+\lambda_2)t+(c-2)(-\lambda_1\lambda_2)t^2}{(1+\lambda_1 t)(1+\lambda_2 t)}.\end{aligned}$$

Combining the above with (3), Corollary 45, and (6), we finally conclude

$$-t\frac{d}{dt}\log\det L(t) + |V|\cdot\frac{(\lambda_1+\lambda_2)t+(c-2)(-\lambda_1\lambda_2)t^2}{(1+\lambda_1 t)(1+\lambda_2 t)} = -t\frac{d}{dt}\log\det(\mathbb{1}-Bt).$$

Finally, dividing by $-t$, integrating (which leaves an unspecified additive constant), and exponentiating (now there is an unspecified multiplicative constant) yields

$$(\text{const.})\cdot(1+\lambda_1 t)^{|V|\frac{c\lambda_2}{\lambda_2-\lambda_1}-1}(1+\lambda_2 t)^{|V|\frac{c\lambda_1}{\lambda_1-\lambda_2}-1}\det L(t) = \det(\mathbb{1}-Bt).$$

By consideration of $t=0$ we see that the constant must be 1.

D Connecting the adjacency and nomadic spectrum

Let $\mathcal{A}=(A_1,\ldots,A_c)$ be a sequence of atoms with two distinct eigenvalues λ_1 and λ_2, let $\mathcal{H}$ be an r-ary, c-atom constraint graph, and let $\mathcal{I}=\mathcal{A}(\mathcal{H})$ be the corresponding instance graph. We use A for the adjacency matrix of $\mathcal{I}$, B for its nomadic walk matrix, V for its vertex set, and E for its edge set. Recall that α_{gr} is defined as $(c-1)(-\lambda_1\lambda_2)$.

We want to use Theorem 38 to describe the spectrum of B with respect to that of A. We will refer to eigenvalues of B with the letter μ and eigenvalues of A with the letter ν.

First, notice that if t is such that $\det(\mathbb{1} - Bt) = 0$, then $\mu = 1/t$ has $\det(\mu\mathbb{1} - B) = 0$, meaning μ is an eigenvalue of B. Thus we want to find for which values of t does the left-hand side of the expression in Theorem 21 become 0 in order to deduce the spectrum of B.

It is easy to see that when $t = -1/\lambda_1$ and $t = -1/\lambda_2$ the left-hand side is always 0, so $-\lambda_1$ is an eigenvalue of B with multiplicity $|V|(\frac{c\lambda_2}{\lambda_2-\lambda_1} - 1)$ and $-\lambda_2$ is an eigenvalue with multiplicity $|V|(\frac{c\lambda_1}{\lambda_1-\lambda_2} - 1)$. The remaining eigenvalues are given by the values of t for which $\det(L(t)) = 0$. Let t be such that $\det(L(t)) = 0$; then we have that $L(t)$ is non-invertible, which means there is some vector v in the nullspace of $L(t)$. By rearranging the equality $L(t)v = 0$ we get:

$$Av = \frac{1 + (\lambda_1 + \lambda_2)t + \alpha_{\text{gr}}t^2}{t}v.$$

This implies that $\frac{1+(\lambda_1+\lambda_2)t+\alpha_{\text{gr}}t^2}{t}$ is an eigenvalue of A. Let ν be some eigenvalue of A; then we have that $\nu = \frac{1+(\lambda_1+\lambda_2)t+\alpha_{\text{gr}}t^2}{t}$ for some t. If we rearrange the previous expression we get the following quadratic equation in t:

$$1 + (\lambda_1 + \lambda_2 - \nu)t + \alpha_{\text{gr}}t^2 = 0.$$

By solving this expression for t and then using the fact that $\mu = 1/t$ we get (notice that $c > 1$ and $\lambda_1\lambda_2 \neq 0$):

$$\mu = \frac{-2\alpha_{\text{gr}}}{\lambda_1 + \lambda_2 - \nu \pm \sqrt{(\lambda_1 + \lambda_2 - \nu)^2 - 4\alpha_{\text{gr}}}}.$$

To analyze the previous we look at three cases:

1. $\nu > \lambda_1 + \lambda_2 + 2\sqrt{\alpha_{\text{gr}}}$. In this case the discriminant is always positive. If we look at the $-$ branch of the $\pm$ we further get that the denominator of the previous formula is always less than $-2\sqrt{\alpha_{\text{gr}}}$ which means we have that μ is real and $\mu > \sqrt{\alpha_{\text{gr}}}$. Additionally, we have that in this interval μ is an increasing function of ν.
2. $\nu < \lambda_1 + \lambda_2 - 2\sqrt{\alpha_{\text{gr}}}$. This is analogous to the previous case; if we look at the $+$ branch we have that μ is real and $\mu < -\sqrt{\alpha_{\text{gr}}}$. Additionally, we have that in this interval μ is a decreasing function of ν.
3. $\nu \in [\lambda_1 + \lambda_2 - 2\sqrt{\alpha_{\text{gr}}}, \lambda_1 + \lambda_2 + 2\sqrt{\alpha_{\text{gr}}}]$, for each such ν we get a pair of anti-conjugate complex numbers, meaning a pair $x, \bar{x}$ such that $x\bar{x} = -1$.

Finally, the spectrum of B also contains 0 with multiplicity $2|E| - |V|\left(2 + (\frac{c\lambda_1}{\lambda_1-\lambda_2} - 1) + (\frac{c\lambda_2}{\lambda_2-\lambda_1} - 1)\right)$, which we get because the degrees of the polynomials in the left-hand side and right-hand do not match; the right-hand side has degree $2|E|$ but we only described $|V|\left(2 + (\frac{c\lambda_1}{\lambda_1-\lambda_2} - 1) + (\frac{c\lambda_2}{\lambda_2-\lambda_1} - 1)\right)$ roots.

We can now summarize the eigenvalues of B in the following way:

- $-\lambda_1$ with multiplicity $|V|(\frac{c\lambda_2}{\lambda_2-\lambda_1} - 1)$;
- $-\lambda_2$ with multiplicity $|V|(\frac{c\lambda_1}{\lambda_1-\lambda_2} - 1)$;
- for each eigenvalue ν of A we get two eigenvalues that are solutions to the previous quadratic equation;
- 0 with multiplicity $2|E| - |V|\left(2 + (\frac{c\lambda_1}{\lambda_1-\lambda_2} - 1) + (\frac{c\lambda_2}{\lambda_2-\lambda_1} - 1)\right)$;

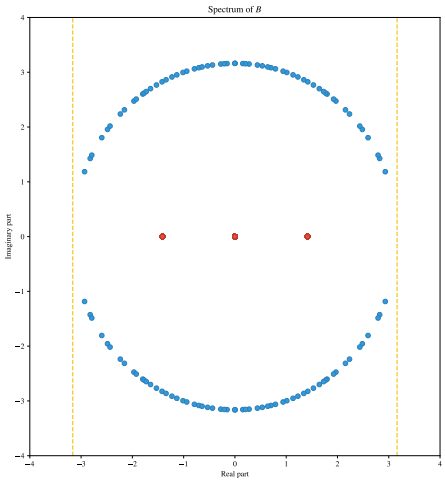


Figure 5 The spectrum of B for a additive 15-lift of 6 copies of a Sort_4 graph. The blue dots are eigenvalues that come from eigenvalues of A, the red dots are either $-\lambda_1$, $-\lambda_2$ or 0 and the yellow line is the limit $\sqrt{\alpha_{\text{gr}}}$.

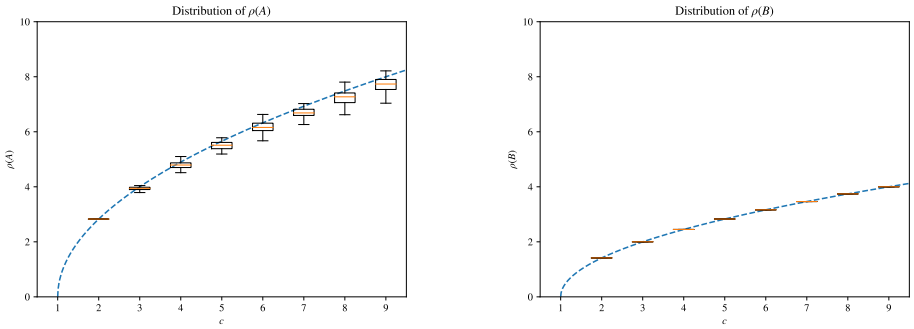


Figure 6 A box plot of $\rho(A)$ and $\rho(B)$ of 100 samples of random instance graphs as a function of c with $n = 15$, $r = 4$ and all atoms are the Sort_4 graph. The dashed line shows the theoretical bound prediction of $2\sqrt{\alpha_{\text{gr}}}$ for A and $\sqrt{\alpha_{\text{gr}}}$ for B.

The distribution of the eigenvalues that come from A forms a sort of semicircle. To showcase this behavior we display an example of the spectrum of typical lifted instance in Figure 5.

We can now prove the central theorem of this section:

▶ **Theorem 46.** *Let $\boldsymbol{\mathcal{I}}_n$ be a random additive n-lift of $\mathcal{A}$ with adjacency matrix $A_{\boldsymbol{\mathcal{I}}_n}$, and let $\epsilon > 0$. Then:*

$$\mathbf{Pr}\left[\rho(A_{\boldsymbol{\mathcal{I}}_n}) \in [\lambda_1 + \lambda_2 - 2\sqrt{\alpha_{\text{gr}}} - \varepsilon, \lambda_1 + \lambda_2 + 2\sqrt{\alpha_{\text{gr}}} + \varepsilon]\right] = 1 - o_n(1)$$

Proof. First recall Theorem 23 (for fully formal statement, see Theorem 79) and notice that $\rho(|B|) = \alpha_{\text{gr}}$, which follows by using the trivial upper bound of α_{gr}^{2k} on $\text{tr}\left(|B|^k \left(|B|^*\right)^k\right)$. From cases 1 and 2 in the previous analysis we get that if $\rho(A_{\boldsymbol{\mathcal{I}}_n}) \notin [\lambda_1 + \lambda_2 - 2\sqrt{\alpha_{\text{gr}}} - \varepsilon, \lambda_1 + \lambda_2 + 2\sqrt{\alpha_{\text{gr}}} + \varepsilon]$ there is some constant δ such that $\rho(B_n) > \sqrt{\alpha_{\text{gr}}} + \delta$, which happens with $o_{n\to\infty}(1)$ probability by Theorem 79. ◀

Also, we note that even though throughout our proof we hide various constant factors, the bounds obtained in Theorem 46 and Theorem 79 are empirically visible for very small values of n and c. To justify this claim we show in Figure 6 a plot of samples of random instance graphs for different values of c with a fixed small n.

E Additive products of 2-eigenvalue atoms

In this section, we let $\mathcal{A} = (A_1, \ldots, A_c)$ be a sequence of $\{\pm 1\}$-*weighted* atoms with the same pair of exactly two distinct eigenvalues, λ_1 and λ_2. We also let $X \coloneqq A_1 \dotplus \cdots \dotplus A_c$ be the additive product graph. We use A_X to denote the adjacency operator of X. In this section, $\boldsymbol{\mathcal{I}}_n$ is the instance graph of a random additive n-lift of $\mathcal{A}$ with negations, and we use $A_{\boldsymbol{\mathcal{I}}_n}$ to denote its adjacency matrix. Finally, we recall $\alpha_{\mathrm{gr}} \coloneqq (c-1)(-\lambda_1\lambda_2)$ and define the quantity $r_X \coloneqq 2\sqrt{\alpha_{\mathrm{gr}}}$.

The main results that this section is dedicated to proving are:

▶ **Theorem 47.** *The following are true about the spectrum of X:*
1. $\mathrm{Spec}(A_X) \subseteq [\lambda_1 + \lambda_2 - r_X, \lambda_1 + \lambda_2 + r_X]$;
2. $\lambda_1 + \lambda_2 - r_X$ *and* $\lambda_1 + \lambda_2 + r_X$ *are both in* $\mathrm{Spec}(A_X)$.

▶ **Theorem 48.** *For every $\varepsilon > 0$, for large enough n, there are $|V(\boldsymbol{\mathcal{I}}_n)| \times |V(\boldsymbol{\mathcal{I}}_n)|$ positive semidefinite matrices M_+ and M_- with all-ones diagonals such that*

$$\langle A_{\boldsymbol{\mathcal{I}}_n}, M_+ \rangle \geqslant (\lambda_1 + \lambda_2 + r_X - \varepsilon)n$$
$$\langle A_{\boldsymbol{\mathcal{I}}_n}, M_- \rangle \leqslant (\lambda_1 + \lambda_2 - r_X + \varepsilon)n.$$

with probability $1 - o_n(1)$.

In this section, when we measure the distance between vertices u and v in an instance graph $\boldsymbol{\mathcal{I}}_n$, we look at the corresponding vertices in the constraint graph $\boldsymbol{\mathcal{H}}$, and define $d(u,v) \coloneqq \frac{d_{\boldsymbol{\mathcal{K}}}(u,v)}{2}$. We use $\mathcal{P}_{uv}$ to refer to the collection of edges comprising the shortest path between u and v. We begin with a statement about the "growth rate" of X.

▶ **Lemma 49.** *For all vertices v in $V(X)$, for $t \geqslant 1$ we have*

$$\sum_{u : d(u,v) = t} \prod_{\{i,j\} \in \mathcal{P}_{uv}} (A_X)^2_{ij} = c(c-1)^{t-1}(-\lambda_1\lambda_2)^t.$$

Proof. We proceed by induction. When $t = 1$, the statement immediately follows from Fact 9. Suppose the equality is true for some $t = \ell - 1$, we will show how statement follows for $t = \ell$.

$$\sum_{u : d(u,v) = \ell} \prod_{\{i,j\} \in \mathcal{P}_{uv}} (A_X)^2_{ij} = \sum_{u : d(u,v) = \ell - 1} \left(\prod_{\{i,j\} \in \mathcal{P}_{uv}} (A_X)^2_{ij} \right) \cdot \left(\sum_{\substack{u' \in N(u) \\ d(u',v) = \ell}} (A_X)^2_{uu'} \right)$$

From Fact 9, $\displaystyle\sum_{\substack{u' \sim u \\ d(u',v) = t}} (A_X)^2_{uu'}$ is equal to $(c-1)(-\lambda_1\lambda_2)$, which means the above is equal to

$$= \sum_{u : d(u,v) = \ell - 1} \left(\prod_{\{i,j\} \in \mathcal{P}_{uv}} (A_X)^2_{ij} \right) (c-1)(-\lambda_1\lambda_2)$$
$$= (c-1)^{\ell-2} c (-\lambda_1\lambda_2)^{\ell-1} (c-1)(-\lambda_1\lambda_2)$$
$$= c(c-1)^{\ell-1}(-\lambda_1\lambda_2)^{\ell}.$$

◀

▶ **Corollary 50.** *Since all the weights of X are $\{\pm 1\}$-valued, the degree of every vertex in X equals $c(-\lambda_1\lambda_2)$.*

E.1 Enclosing the spectrum

Let B_X denote the nomadic walk operator of X. In this section, we show

$$\mathrm{Spec}(A_X) \subseteq [\lambda_1 + \lambda_2 - r_X, \lambda_1 + \lambda_2 + r_X]\,.$$

The first part of the proof will involve showing that the spectral radius of B_X is bounded by $\sqrt{\alpha_{\mathrm{gr}}}$, and the second part translates this bound to the desired one on $\mathrm{Spec}(A_X)$. Both these components closely follow proofs from the work of Angel et al.; the former after [5, Theorem 4.2] and the latter after [5, Theorem 1.5].

▶ **Lemma 51.** $\mathrm{Spec}(B_X) \subseteq \left[-\sqrt{\alpha_{\mathrm{gr}}}, \sqrt{\alpha_{\mathrm{gr}}}\right]$.

Proof. Arbitrarily fix a root r of X. Recall that the spectral radius of B_X is equal to $\lim \left(\|B_X^k\|_{\mathrm{op}}\right)^{1/k}$, and hence it suffices to bound $\left|\langle g, B_X^k f\rangle\right|$ for arbitrary f and g with $\|f\| = \|g\| = 1$.

We can decompose every nomadic walk of length k into two segments, a segment of i steps towards r followed by a sequence of $k-i$ steps away from r; henceforth, we call length-k nomadic walks with such a decomposition (i,k)-nomadic walks. For every pair of directed edges e and e' such that $e, e_1, \ldots, e_{k-1}, e'$ is an (i,k)-nomadic walk, let $a(e,e') := \alpha_{\mathrm{gr}}^{k/2-i}$. From Lemma 49, the number of (i,k)-nomadic walks starting at a fixed e is at most $\frac{c}{c-1}\alpha_{\mathrm{gr}}^{k-i}$. Similarly, the number of (i,k)-nomadic walks ending at fixed e' is at most $\frac{c}{c-1}\alpha_{\mathrm{gr}}^{i}$. Now, we are ready to bound $\left|\langle g, B_X^k f\rangle\right|$ by imitating the proof of [5, Theorem 4.2].

$$\begin{aligned}
&\left|\langle g, B_X^k f\rangle\right| \\
&\leqslant \left| \sum_{e,e_1,\ldots,e_{k-1},e' \text{ nomadic}} f(e')g(e) \right| \\
&\leqslant \sum_{e,e_1,\ldots,e_{k-1},e' \text{ nomadic}} |f(e')g(e)| \\
&\leqslant \sum_{e,e_1,\ldots,e_{k-1},e' \text{ nomadic}} a(e,e')f(e')^2 + \frac{1}{a(e,e')}g(e)^2 \\
&\leqslant \sup_{e'} \left(\sum_{e,e_1,\ldots,e_{k-1},e' \text{ nomadic}} a(e,e') \right) \|f\|_2^2 + \sup_{e} \left(\sum_{e,e_1,\ldots,e_{k-1},e' \text{ nomadic}} \frac{1}{a(e,e')} \right) \|g\|_2^2 \\
&\leqslant \sum_{i=0}^{k} \sup_{e'} \left(\sum_{(i,k)\text{-nomadic walks ending at } e'} a(e,e') \right) + \sup_{e} \left(\sum_{(i,k)\text{-nomadic walks starting at } e} \frac{1}{a(e,e')} \right) \\
&\leqslant \sum_{i=0}^{k} \alpha_{\mathrm{gr}}^{k/2-i} \cdot \frac{c}{c-1}\alpha_{\mathrm{gr}}^{i} + \sum_{i=0}^{k} \alpha_{\mathrm{gr}}^{i-k/2} \cdot \frac{c}{c-1}\alpha_{\mathrm{gr}}^{k-i} \\
&= \frac{2kc}{c-1}\alpha_{\mathrm{gr}}^{k/2}
\end{aligned}$$

Thus, we have

$$\|B_X^k\|_{\mathrm{op}} \leqslant \frac{2kc}{c-1}\alpha_{\mathrm{gr}}^{k/2}$$

and taking the limit of $\|B_X^k\|_{\mathrm{op}}^{1/k}$ for k approaching infinity yields the desired statement. ◀

▶ **Lemma 52.** *If* 0 *is an approximate eigenvalue of* $Q_t := (t^2 + (c-1)(-\lambda_1\lambda_2))\mathbb{1} - A_X t + (\lambda_1 + \lambda_2)\mathbb{1}t$*, then it is also an approximate eigenvalue of* $B_X - t\mathbb{1}$ *as long as* $t \neq -\lambda_1, -\lambda_2$*.*

Proof. Let f be an ε-approximate eigenfunction of unit norm of Q_t, then we construct a $C\varepsilon$-approximate eigenfunction g of $B_X - t\mathbb{1}$ defined on pairs uv such that u and v are incident to a common atom for an absolute constant $C > 0$ as follows,

$$g_{uv} := \left(\sum_{w:\{v,w\}\in \mathrm{Atom}(\{u,v\})} (A_X)_{vw} f_w \right) - (\lambda_1 + \lambda_2 + t) f_v$$

for every edge $\{u, v\}$ of X.

$$\begin{aligned}
&((B_X - t\mathbb{1})g)_{uv} \\
&= \left(\sum_{\substack{w:\\ \{v,w\}\notin \mathrm{Atom}(\{u,v\})}} (B_X)_{uv,vw} g_{vw} \right) - t g_{uv} \\
&= \left(\sum_{\substack{w:\\ \{v,w\}\notin \mathrm{Atom}(\{u,v)\}}} (A_X)_{vw} \left(\sum_{\substack{x:\\ \{w,x\}\in \mathrm{Atom}(\{v,w\})}} (A_X)_{wx} f_x - (\lambda_1 + \lambda_2 + t) f_w \right) \right) - t g_{uv} \\
&= \left(\sum_{\substack{w:\\ \{v,w\}\notin \mathrm{Atom}(\{u,v)\}}} \sum_{\substack{x:\\ \{w,x\}\in \mathrm{Atom}(\{v,w\})}} (A_X)_{vw}(A_X)_{wx} f_x \right) - \\
&\quad \left(\sum_{\substack{w:\\ \{v,w\}\notin \mathrm{Atom}(\{u,v)\}}} (\lambda_1 + \lambda_2 + t)(A_X)_{vw} f_w \right) - t g_{uv}
\end{aligned}$$

Using Fact 9 and Fact 10, the first term of the three above can be rewritten as

$$(c-1)(-\lambda_1\lambda_2) f_v + (\lambda_1 + \lambda_2) \sum_{w:\{v,w\}\notin \mathrm{Atom}(\{u,v\})} (A_X)_{vw} f_w$$

which lets us continue the chain of equalities

$$\begin{aligned}
&= (c-1)(-\lambda_1\lambda_2) f_v - t \sum_{\substack{w:\\ \{v,w\}\notin \mathrm{Atom}(\{u,v)\}}} (A_X)_{vw} f_w \\
&- t \left(\sum_{w:\{v,w\}\in \mathrm{Atom}(\{u,v\})} (A_X)_{vw} f_w \right) + t(\lambda_1 + \lambda_2 + t) f_v \\
&= (c-1)(-\lambda_1\lambda_2) f_v - t(Af)_v + t(\lambda_1 + \lambda_2 + t) f_v \\
&= (Q_t f)_v.
\end{aligned}$$

Thus,

$$\|(B_X - t\mathbb{1})g\|_2^2 = \sum_{\{u,v\}\in E(X)} ((B_X - t\mathbb{1})g)_{uv}^2 + ((B_X - t\mathbb{1})g)_{vu}^2 = d \sum_{v\in V} (Q_t f)_v^2 \leqslant d\varepsilon^2$$

It remains to show that the norm of g is bounded from above and below. Fix a vertex u and an atom $\widetilde{A}$ incident to u. Consider $g^{(u,\widetilde{A})}$, the restriction of g to entries uv such that the edge $\{u, v\}$ is in $\widetilde{A}$, and $f^{(\widetilde{A})}$, the restriction of f to vertices v such that $\widetilde{A}$ is incident to v. Observe

that $g^{(u,\widetilde{A})} = (A_{\widetilde{A}} - (\lambda_1+\lambda_2+t)\mathbb{1})f^{(\widetilde{A})}$. Since the min eigenvalue of $A_{\widetilde{A}} - (\lambda_1+\lambda_2+t)\mathbb{1}$ is nonzero as long as $t \neq -\lambda_1, -\lambda_2$, the ℓ_2 norm of g is bounded from below. To prove that the ℓ_2 norm of g is bounded from above, observe that

$$\begin{aligned}\|g\|_2^2 &= \sum_{\widetilde{A}\in\mathrm{Atoms}(X)} \sum_{(u,v):\{u,v\}\in\widetilde{A}} \left(\left(\sum_{w:\{v,w\}\in\widetilde{A}} (A_X)_{vw} f_w \right) - (\lambda_1+\lambda_2+t) f_v \right)^2 \\ &\leqslant 2 \sum_{\widetilde{A}\in\mathrm{Atoms}(X)} \sum_{(u,v):\{u,v\}\in\widetilde{A}} \left(\sum_{\{v,w\}\in\widetilde{A}} (A_X)_{vw}^2 f_w^2 + (\lambda_1+\lambda_2+t)^2 f_v^2 \right)\end{aligned}$$

There is some coefficient α such that the weight on f_v^2 for each v in the above sum is bounded by α, thereby giving a bound of

$$2\sum_{v\in V} \alpha f_v^2 \leqslant 2\alpha\|f\|_2^2 \leqslant 2\alpha.$$ ◀

Proof of Item 1 in Theorem 47. Let Q_t be as defined in the statement of Lemma 52. It can be verified that 0 is an approximate eigenvalue of either $Q_{-\lambda_1}$ or $Q_{-\lambda_2}$ if and only if $d_X := c(-\lambda_1\lambda_2)$, which we recall from Corollary 50 is the degree of every vertex in X, is in the spectrum of A_X. Let $\mu_+ := \lambda_1+\lambda_2+r_X+\eta$ be in spectrum of A_X. If $\mu_+ \neq d_X$, then we can conclude from Lemma 52 that

$$\alpha_{\mathrm{gr}} + \eta + \sqrt{\eta\alpha_{\mathrm{gr}} + \eta^2/4}$$

is an approximate eigenvalue of B_X. Since $\mathrm{Spec}(B_X)$ is contained in $[-\sqrt{\alpha_{\mathrm{gr}}}, \sqrt{gr}]$, η cannot be positive. A similar argument applied to $\mu_- := \lambda_1+\lambda_2-r_X-\eta$ precludes η from being positive as long as $\mu_- \neq d_X$. As a result, we can conclude that $\mathrm{Spec}(A_X)$ is contained in $[\mu_-, \mu_+] \cup \{d_X\}$. If d_X is in the interval $[\mu_-, \mu_+]$, then we are done. If not, then it remains to show that d_X is not in $\mathrm{Spec}(A_X)$. Since X is $\{\pm1\}$-weighted and the degree of each vertex is d_X, any nonzero x satisfying $A_X x = d_X x$ must have the same nonzero magnitude in all its entries. However, such x has unbounded ℓ_2 norm, and hence A_X has no eigenvectors with eigenvalue d_X in $\ell_2(V)$. If d_X is in $\mathrm{Spec}(A_X)$, it is an isolated point in the spectrum, and hence, by Theorem 32, is an eigenvalue of A_X, which means d_X cannot be in $\mathrm{Spec}(A_X)$. ◀

E.2 Construction of Witness Vectors

▶ **Lemma 53** (Item 2 of Theorem 47 restated). *There exists $\lambda_- \leqslant \lambda_1+\lambda_2-r_X$ and $\lambda_+ \geqslant \lambda_1+\lambda_2+r_X$ in the spectrum of A_X.*

Proof. Let $\delta > 0$ be a parameter to be chosen later. First define ρ as

$$\rho(s) := \frac{s(1-\delta)}{\sqrt{(c-1)(-\lambda_1\lambda_2)}}$$

Then, for vertex v and define $f_v^{(s)}$ in the following way.

$$f_v^{(s)}(u) := \rho(s)^{d(u,v)} \prod_{\{i,j\}\in\mathcal{P}_{uv}} (A_X)_{ij} \quad \text{where } \mathcal{P}_{uv} \text{ is the unique nomadic walk between } u \text{ and } v \tag{8}$$

To show the lemma, it suffices to prove the claim that for every $\varepsilon > 0$, there is suitable choice of δ so that

$$\frac{\langle f_v^{(-1)}, A_X f_v^{(-1)}\rangle}{\langle f_v^{(-1)}, f_v^{(-1)}\rangle} < \lambda_1 + \lambda_2 - r_X + \varepsilon$$

and

$$\frac{\langle f_v^{(1)}, A_X f_v^{(1)}\rangle}{\langle f_v^{(1)}, f_v^{(1)}\rangle} > \lambda_1 + \lambda_2 + r_X - \varepsilon$$

We proceed by analyzing the expression $\langle f_v^{(s)}, A_X f_v^{(s)}\rangle$.

$$\begin{aligned}\langle f_v^{(s)}, A_X f_v^{(s)}\rangle &= \sum_{u\in V} f_v^{(s)}(u) A_X f_v^{(s)}(u)\\ &= f_v^{(s)}(v)\sum_{w\in N(v)} (A_X)_{vw} f_v^{(s)}(w) + \sum_{u\in V, u\neq v} f_v^{(s)}(u)\sum_{w\in N(u)} (A_X)_{uw} f_v^{(s)}(w)\\ &= \sum_{w\in N(v)} (A_X)_{vw}^2 \rho(s) + \sum_{u\in V, u\neq v} f_v^{(s)}(u)\sum_{w\in N(u)} (A_X)_{uw} f_v^{(s)}(w) \qquad (9)\end{aligned}$$

Let $w_0, w_1, \ldots w_{T-1}, w_T$ be the sequence of vertices from the unique nomadic walk between u and v where $w_0 = u$ and $w_T = v$. Now, let $u^* = w_1$. Recall the notation $\mathcal{P}_{u,v}$ used to denote the unique nomadic walk between u and v as a sequence of edges. Let $W_{u,v} := \rho(s)^{d(u,v)} \prod_{\{i,j\}\in\mathcal{P}_{u,v}} (A_X)_{ij}$. Using the notation we just developed, along with applying Fact 9 on the first term of the above, we get

$$\begin{aligned}(9) &= c(-\lambda_1\lambda_2)\rho(s) + \sum_{u\in V, u\neq v} \rho(s) W_{u^*v} (A_X)_{uu^*} \cdot\\ &\quad\left((A_X)_{uu^*} W_{u^*v} + \sum_{w\in \mathrm{Atom}(\{u^*,u\})} \rho(s)(A_X)_{u^*w}(A_X)_{wu} W_{u^*v} + \right.\\ &\qquad\left. \sum_{\substack{w\notin \mathrm{Atom}(\{u,u^*\})\\ w\in N(u)}} \rho(s)^2 (A_X)_{u^*u}(A_X)_{uw}^2 W_{u^*v}\right)\\ &= c(-\lambda_1\lambda_2)\rho(s) + \sum_{u\in V, u\neq v} \rho(s) W_{u^*v}^2 (A_X)_{uu^*}^2 \cdot\\ &\quad\left(1 + \frac{\sum_{w\in \mathrm{Atom}(\{u^*,u\})} \rho(s)(A_X)_{u^*w}(A_X)_{wu}}{A_{uu^*}} + \sum_{\substack{w\notin \mathrm{Atom}(\{u,u^*\})\\ w\in N(u)}} (A_X)_{uw}^2 \rho(s)^2\right)\end{aligned}$$

Now we apply Fact 9 and Fact 10 and get

$$\begin{aligned}&= c(-\lambda_1\lambda_2)\rho(s) + \sum_{u\in V, u\neq v} \rho(s) W_{u^*v}^2 (A_X)_{uu^*}^2 \cdot \left(1 + \rho(s)(\lambda_1+\lambda_2) + (c-1)(-\lambda_1\lambda_2)\rho(s)^2\right)\\ &= c(-\lambda_1\lambda_2)\rho(s) + \sum_{u\in V, u\neq v} W_{uv}^2 \cdot \frac{1 + \rho(s)(\lambda_1+\lambda_2) + (c-1)(-\lambda_1\lambda_2)\rho(s)^2}{\rho(s)}\\ &= c(-\lambda_1\lambda_2)\rho(s) + \left(\|f_v^{(s)}\|^2 - 1\right)\cdot \frac{1 + \rho(s)(\lambda_1+\lambda_2) + (c-1)(-\lambda_1\lambda_2)\rho(s)^2}{\rho(s)}\\ &= c(-\lambda_1\lambda_2)\rho(s) + \left(\|f_v^{(s)}\|^2 - 1\right)\cdot \left(\frac{1 + s^2(1-\delta)^2}{\rho(s)} + (\lambda_1+\lambda_2)\right)\end{aligned}$$

When $s = \pm 1$, the above quantity is equal to

$$c(-\lambda_1\lambda_2)\rho(s) + \left(\|f_v^{(s)}\|^2 - 1\right) \cdot \left(\frac{1+(1-\delta)^2}{\rho(s)} + (\lambda_1+\lambda_2)\right)$$

Now, note that

$$\frac{\langle f_v^{(s)}, A_X f_v^{(s)}\rangle}{\langle f_v^{(s)}, f_v^{(s)}\rangle} = \frac{c(-\lambda_1\lambda_2)\rho(s)}{\|f_v^{(s)}\|^2} + \left(1 - \frac{1}{\|f_v^{(s)}\|^2}\right) \cdot \left(\frac{1+(1-\delta)^2}{\rho(s)} + (\lambda_1+\lambda_2)\right) \tag{10}$$

We now compute $\|f_v^{(s)}\|^2$, and we assume s is either $+1$ or -1.

$$\begin{aligned}
\|f_v^{(s)}\|^2 &= \sum_{t=0}^{\infty} \rho(s)^{2t} \sum_{u:d(u,v)=t} \prod_{\{i,j\}\in\mathcal{P}_{uv}} (A_X)_{ij}^2 \\
&= \sum_{t=0}^{\infty} \rho(s)^{2t} c(c-1)^{t-1}(-\lambda_1\lambda_2)^t && \text{(by Lemma 49)} \\
&= \frac{c}{c-1} \sum_{t=0}^{\infty} \left(\frac{(1-\delta)^{2t}}{(c-1)^t(-\lambda_1\lambda_2)^t}\right)(c-1)^t(-\lambda_1\lambda_2)^t \\
&= \frac{c}{c-1} \sum_{t=0}^{\infty} (1-\delta)^{2t} \\
&= \frac{c}{c-1} \cdot \frac{1}{\delta(2-\delta)}
\end{aligned}$$

Plugging this back in to (10) gives

$$\begin{aligned}
(10) &= \delta(2-\delta)(c-1)(-\lambda_1\lambda_2)\rho(s) + \left(\frac{1+(1-\delta)^2}{\rho(s)} + (\lambda_1+\lambda_2)\right) \cdot \left(1 - \frac{(c-1)\delta(2-\delta)}{c}\right) \\
&= \delta(2-\delta)s(1-\delta)\sqrt{(c-1)(-\lambda_1\lambda_2)} + \\
&\left((1+(1-\delta)^2)\sqrt{(c-1)(-\lambda_1\lambda_2)}\frac{1}{s(1-\delta)} + (\lambda_1+\lambda_2)\right) \cdot \left(1 - \frac{(c-1)\delta(2-\delta)}{c}\right)
\end{aligned}$$

For any $\varepsilon > 0$, we can choose δ small enough so that the above quantity is at least

$$\lambda_1 + \lambda_2 + 2\sqrt{(c-1)(-\lambda_1\lambda_2)} - \varepsilon$$

when $s = 1$ and at most

$$\lambda_1 + \lambda_2 - 2\sqrt{(c-1)(-\lambda_1\lambda_2)} + \varepsilon$$

when $s = -1$. ◀

E.3 SDP solution for random additive lifts

For $\varepsilon > 0$, consider $f_v^{(1)}$ constructed in the proof of Lemma 53, for which

$$\langle f_v^{(1)}, A_X f_v^{(1)}\rangle \geqslant (\lambda_1 + \lambda_2 + r_X - \varepsilon)\|f_v^{(1)}\|^2$$

Let L_ε be an integer chosen such that the total ℓ_2 mass of $\frac{f_v^{(1)}}{\|f_v^{(1)}\|}$ on vertices at distance greater than L_ε from v is at most ε. Define g_v as the vector obtained by zeroing out $\frac{f_v^{(1)}}{\|f_v^{(1)}\|}$ on vertices outside $B(v, L_\varepsilon)$ and normalizing to make its norm 1, where $B(v, L_\varepsilon)$ is the collection of vertices within distance L_ε of v.

For any $\varepsilon' > 0$, we can choose ε so that

$$\langle g_v, A_X g_v \rangle \geqslant \lambda_1 + \lambda_2 + r_X - \varepsilon' \tag{11}$$

g_v enjoys the property of being determined by a constant number of vertices. For any instance graph G such that there is a unique shortest nomadic walk between any pair of vertices u and v, we can explicitly define

$$g_v(u) = \begin{cases} 0 & \text{if } d(u,v) > L_{\varepsilon'} \\ C \prod\limits_{\{i,j\} \in \mathcal{P}_{uv}} \frac{(1-\delta)(A_X)_{ij}}{\sqrt{(c-1)(-\lambda_1\lambda_2)}} & \mathcal{P}_{uv} \text{ unique shortest nomadic walk from } u \text{ to } v \end{cases}$$

where C is a constant chosen so that g_v has unit norm.

Recall that $\boldsymbol{\mathcal{I}}_n$ is a random signed additive n-lift obtained from a sequence of atoms $\mathcal{A}$.

▶ **Definition 54.** *Let G be a graph and let $\phi : E(G) \to \{\pm 1\}$ be a signing of the edges. We call a signing ϕ balanced if for any cycle given by sequence of edges $e_1, \ldots, e_k$ in $E(H)$, we have $\phi(e_1) \cdots \phi(e_k) = 1$.*

We use $A_{\phi(G)}$ to denote the adjacency operator of G signed with respect to ϕ – i.e. $(A_{\phi(G)})_{uv} = \phi(\{u,v\})$ if $\{u,v\}$ is an edge and 0 otherwise.

▶ **Lemma 55.** *Suppose ϕ is a balanced signing of G. Then there exists a diagonal sign operator D such that $A_{\phi(G)} = DA_G D^\dagger$.*

Proof. Without loss of generality, assume G is connected. Take a spanning tree of G and root it at some arbitrary vertex r. Let $D_{rr} = 1$ and for P_x a path from r to x let $D_{xx} = \prod_{e \in P_x} \phi(e)$.

It remains to verify that $DA_G D^\dagger = A_{\phi(G)}$. Let P be the path between x and y in the spanning tree. By virtue of ϕ being balanced, we have $\phi(\{x,y\}) \prod_{e \in P} \phi(e) = 1$, which means $\phi(\{x,y\}) = \prod_{e \in P} \phi(e)$. Also, note that $\prod_{e \in P} \phi(e)$ is equal to $\prod_{e \in P_x} \phi(e) \prod_{e \in P_y} \phi(e)$, which is equal to $D_{xx} D_{yy}$. Thus,

$$(A_{\phi(G)})_{ij} = \phi(\{i,j\})(A_G)_{ij} = D_{ii} D_{jj} (A_G)_{ij} = \left(DA_G D^\dagger\right)_{ij}$$

which proves the claim. ◀

▶ **Lemma 56.** *Let X_D be the graph with the adjacency operator $DA_X D^\dagger$ where D is a diagonal sign matrix. There exists D such that X_D covers $\boldsymbol{\mathcal{I}}_n$.*

Proof. When $\boldsymbol{\mathcal{I}}_n$ is generated, (i) the sequence of atoms $\mathcal{A}$ first undergoes an additive n-lift, and then, (ii) the atoms in the lifted graph are given a random balanced signing. The intermediate graph $\widetilde{\boldsymbol{\mathcal{I}}}_n$ between (i) and (ii) is covered by X via a map $\pi : V(X) \to V(\widetilde{\boldsymbol{\mathcal{I}}}_n)$. Once (ii) is performed, construct X' by taking X and setting the signs on all edges in $\pi^{-1}(e)$ to the sign on e for each $e \in E(\boldsymbol{\mathcal{I}}_n)$. X' can be seen as a balanced signing applied on X, and hence there exists such a D by Lemma 55. ◀

▶ **Definition 57.** *Let π be a covering map from appropriate X_D to $\boldsymbol{\mathcal{I}}_n$. Call a vertex $v \in V(\boldsymbol{\mathcal{I}}_n)$ L-bad if $B(v, L)$ is not isomorphic to $B(v^*, L)$ where $v^* \in V(X_D)$ is such that $\pi(v^*) = v$.*

▶ Remark 58. The condition of a vertex v in $V(\boldsymbol{\mathcal{I}}_n)$ being L-bad according to Definition 57 is equivalent to the corresponding variable v' in the constraint graph having a cycle in its distance $2L$-neighborhood.

With the observation of Remark 58 in hand, we can extract the following as a consequence of [24].

▶ **Lemma 59.** *The number of K-bad vertices in graph $\boldsymbol{\mathcal{I}}_n$ for constant K is bounded by $O(\log n)$ with probability $1 - o_n(1)$.*

Construct a vector $\widetilde{g}_v$ for each vertex v of $\boldsymbol{\mathcal{I}}_n$.

$$\widetilde{g}_v = \begin{cases} e_v & \text{if } v \text{ is } L_{\varepsilon'}\text{-bad} \\ g_v & \text{otherwise} \end{cases}$$

We are finally ready to prove Theorem 48.

Proof of Theorem 48. Let

$$M_+ := \sum_{v \in V(\boldsymbol{\mathcal{I}}_n)} \widetilde{g}_v \widetilde{g}_v^\dagger$$

Writing out $(M_+)_{uu}$ for arbitrary u

$$\begin{aligned}(M_+)_{uu} &= \sum_{v \in V(\boldsymbol{\mathcal{I}}_n)} \widetilde{g}_v(u)\widetilde{g}_v(u) \\ &= \sum_{v \in V(\boldsymbol{\mathcal{I}}_n)} \widetilde{g}_u(v)^2 \\ &= \|\widetilde{g}_u\|^2 = 1\end{aligned}$$

and writing out $\langle A_{\boldsymbol{\mathcal{I}}_n}, M_+ \rangle$ gives the following with probability $1 - o_n(1)$.

$$\begin{aligned}\langle A_{\boldsymbol{\mathcal{I}}_n}, M_+ \rangle &= \sum_{v \in V(\boldsymbol{\mathcal{I}}_n)} \langle \widetilde{g}_v, A_{\boldsymbol{\mathcal{I}}_n} \widetilde{g}_v \rangle \\ &= \sum_{\substack{v \in V(\boldsymbol{\mathcal{I}}_n) \\ v \text{ is not } (L_\varepsilon+1)\text{-bad}}} \langle \widetilde{g}_v, A_{\boldsymbol{\mathcal{I}}_n} \widetilde{g}_v \rangle + \sum_{\substack{v \in V(\boldsymbol{\mathcal{I}}_n) \\ v \text{ is } (L_\varepsilon+1)\text{-bad}}} \langle \widetilde{g}_v, A_{\boldsymbol{\mathcal{I}}_n} \widetilde{g}_v \rangle \\ &\geqslant \sum_{\substack{v \in V(\boldsymbol{\mathcal{I}}_n) \\ v \text{ is not } (L_\varepsilon+1)\text{-bad}}} \lambda_1 + \lambda_2 + r_X - \varepsilon' + \sum_{\substack{v \in V(\boldsymbol{\mathcal{I}}_n) \\ v \text{ is } (L_\varepsilon+1)\text{-bad}}} c(\lambda_1\lambda_2) \qquad \text{(by (11))} \\ &\geqslant (n - O(\log n))(\lambda_1 + \lambda_2 + r_X - \varepsilon') - O(\log n) \qquad \text{(by Lemma 59)} \\ &= (1 - o_n(1))(\lambda_1 + \lambda_2 + r_X - \varepsilon')n\end{aligned}$$

The desired inequality on $\langle A_{\boldsymbol{\mathcal{I}}_n}, M_+ \rangle$ can be obtained by choosing ε' small enough and n large enough. The inequality on $\langle A_{\boldsymbol{\mathcal{I}}_n}, M_- \rangle$ can be proved by repeating the whole section and proof by constructing vectors $\widetilde{g}_v$ from $f_v^{(-1)}$. ◀

F Friedman/Bordenave for additive lifts

▶ **Theorem 60.** *Let $\mathcal{A} = (A_1, \ldots, A_c)$ be a sequence of r-vertex atoms with edges weights ± 1. Let $|\mathcal{I}_1|$ denote the instance graph $\mathcal{A}(K_{r,c})$ associated to the base constraint graph when the edge-signs are deleted (i.e., converted to $+1$), and let $|B_1|$ denote the associated nomadic walk matrix. Also, let $\boldsymbol{\mathcal{H}}_n$ denote a random n-lifted constraint graph and $\boldsymbol{\mathcal{I}}_n = \mathcal{A}(\boldsymbol{\mathcal{H}}_n)$ an associated instance graph with 1-wise uniform negations $(\boldsymbol{\xi}_{ii'}^f)$. Finally, let $\boldsymbol{B}_n$ denote the nomadic walk matrix for $\boldsymbol{\mathcal{I}}_n$. Then for every constant $\varepsilon > 0$,*

$$\mathbf{Pr}[\rho(\boldsymbol{B}_n) \geqslant \sqrt{\rho(|B_1|)} + \varepsilon] \leqslant \delta,$$

where $\delta = \delta(n)$ is $o_{n\to\infty}(1)$.

▶ Remark 61. It might seem that our bound involving $|B_1|$ may be poor, given that it ignores sign information from the atoms. However, it is in fact sharp, and the reason is that the main contribution to $\rho(\boldsymbol{B}_n)$ when using the Trace Method is from walks in which almost all edges are traversed twice. And if an edge is traversed twice, it of course does not matter if its sign is -1 or $+1$.

▶ Remark 62. In fact, it is evident from the theorem statement that without loss of generality we may assume that the atoms are unweighted – i.e., that all weights are $+1$. The reason is that for each constraint f in group j, if we multiply $\boldsymbol{\xi}^f_{ii'}$ by the fixed value $A_j[i, i']$, the resulting signs remain 1-wise uniform – and this has the effect of eliminating all signs from the atoms. Thus henceforth we will indeed assume that the original atoms are all unweighted.

The idea of Friedman/Bordenave proofs. The standard method for trying to prove a theorem such as Theorem 60 involves applying the Trace Method to $\boldsymbol{B}_n$. Since $\boldsymbol{B}_n$ is not a self-adjoint operator, a natural way to do this is to consider $\mathrm{tr}(\boldsymbol{B}_n^\ell \boldsymbol{B}_n^{*\ell})$ for some large ℓ. Roughly speaking, this counts the number of closed walks that walk nomadically in $\boldsymbol{\mathcal{I}}_n$ for the first ℓ steps, and then walk nomadically in the *reverse* of $\boldsymbol{\mathcal{I}}_n$ for the next ℓ steps. A major difficulty is the following: the Trace Method naturally incurs an "extra" factor of n, and to overcome this one wants to choose $\ell \gg \log n$. However, $\Theta(\log n)$ is precisely the radius at which random constraint graphs become dramatically non-tree-like; i.e., they are likely to encounter nontrivial cycles. Based on Friedman's work, Bordenave overcomes this difficulty as follows: First, ℓ is set to $c \log n$ for some small positive constant $c > 0$. Nomadic walks of this length may well encounter cycles, but one can show that with high probability, they will not encounter *tangles* – meaning, *more than one* cycle in a radius of ℓ. (This crucial concept of "tangles" was isolated by Friedman and refined by Bordenave.) Now we set $k = \omega_n(1)$ to be a slowly growing quantity and consider length-$2k\ell$ walks formed by doing ℓ nomadic steps, then ℓ nomadic reverse-steps, all k times in succession. In other words, we consider $\mathrm{tr}((\boldsymbol{B}_n^\ell \boldsymbol{B}_n^{*\ell})^k)$. On one hand, since $2k\ell \gg \log n$, bounding this quantity will be sufficient to overcome the n-factor inherent in the Trace Method. On the other hand, using tangle-freeness at radius ℓ along with very careful combinatorial counting allows us to bound the number of closed length-$2k\ell$ walks.

Our proof follows this methodology and draws ideas from Bordenave's original proof from [12] as well as [24] and [16]. However, our main technical lemma, Lemma 83, uses a new tool that takes advantage of the random negations our model employs that simplifies the equivalent proofs in the three mentioned papers and also allows us to generalize it to our model.

F.1 Trace Method setup, and getting rid of tangles

To begin carrying out this proof strategy, we first define tangle-freeness.

▶ **Definition 63** (Tangles-free). *Let G be an undirected graph. A vertex v is said to be ℓ-tangle-free within G if the subgraph of G induced by v's distance-4ℓ neighborhood contains at most one cycle.*[5]

It is straightforward to show that random lifts have all vertices $\Theta(\log n)$-tangle-free; we can quote the relevant result directly from Bordenave (Lemma 27 from [12]):

[5] We chose the factor 4 here for "safety". For quantitative aspects of our theorem, constant factors on ℓ will be essentially costless.

▶ **Proposition 64.** *There is a universal constant $\kappa > 0$ depending only on r, c such that, for $\ell = \kappa \log n$, a random n-lift $\boldsymbol{\mathcal{H}}$ of $K_{r,c}$ has all vertices ℓ-tangle free, except with probability $O(1/n^{.99})$.*

We now begin the application of the Trace Method. We have:

$$\begin{aligned}
&\mathrm{tr}((\boldsymbol{B}_n^\ell \boldsymbol{B}_n^{*\ell})^k) \\
&= \sum_{\vec{e}_0,\dots,\vec{e}_{2k\ell-1},\vec{e}_{2k\ell}=\vec{e}_0} \boldsymbol{B}_n[\vec{e}_0,\vec{e}_1]\cdots\boldsymbol{B}_n[\vec{e}_{\ell-1},\vec{e}_\ell]\boldsymbol{B}_n^*[\vec{e}_\ell,\vec{e}_{\ell+1}]\cdots\boldsymbol{B}_n^*[\vec{e}_{2\ell-1},\vec{e}_{2\ell}]\cdots\boldsymbol{B}_n^*[\vec{e}_{2k\ell-1},\vec{e}_{2k\ell}] \\
&= \sum_{\vec{e}_0,\dots,\vec{e}_{2k\ell-1},\vec{e}_{2k\ell}=\vec{e}_0} \boldsymbol{B}_n[\vec{e}_0,\vec{e}_1]\cdots\boldsymbol{B}_n[\vec{e}_{\ell-1},\vec{e}_\ell]\boldsymbol{B}_n[\vec{e}_{\ell+1},\vec{e}_\ell]\cdots\boldsymbol{B}_n[\vec{e}_{2\ell},\vec{e}_{2\ell-1}]\cdots\boldsymbol{B}_n[\vec{e}_{2k\ell},\vec{e}_{2k\ell-1}]
\end{aligned}$$

$$\begin{aligned}
= \sum \mathrm{wt}(e_1)N_{\vec{e}_0,\vec{e}_1}\cdots\mathrm{wt}(e_\ell)N_{\vec{e}_{\ell-1},\vec{e}_\ell}\mathrm{wt}(e_\ell)N_{\vec{e}_\ell^{-1},\vec{e}_{\ell+1}^{-1}}\cdots \\
\cdots\mathrm{wt}(e_{2\ell-1})N_{\vec{e}_{2\ell-1}^{-1}\cdots\vec{e}_{2\ell}^{-1}}\cdots\mathrm{wt}(e_{2k\ell-1})N_{\vec{e}_{2k\ell-1}^{-1},\vec{e}_{2k\ell}^{-1}},
\end{aligned} \tag{12}$$

where $\mathrm{wt}(e)$ is the sign on edge e coming from the random 1-wise negations (it is the same for both directed versions of the edge), and where $N_{\vec{e},\vec{f}}$ is an indicator that $(\vec{e},\vec{f})$ forms a length-2 nomadic walk. Roughly speaking, this quantity counts (with some ± 1 sign) closed walks in $\boldsymbol{\mathcal{I}}_n$ consisting of $2k$ consecutive nomadic walks of length ℓ. However, there is some funny business concerning the joints between these nomadic walks. To be more precise, in each of the $2k$ segments we have a nomadic walk of $\ell+1$ edges; and, the last edge in each segment must be the reverse of the first edge in the subsequent segment. We will call these necessarily-duplicated edges "spurs". Furthermore, when computing the sign with which the closed walk is counted, spurs' signs are counted either zero times or twice, depending on the parity of the segment. Hence they are effectively discounted, since $(-1)^2 = (-1)^0 = +1$. Let us make some definitions encapsulating all of this.

▶ **Definition 65** (Nomadic linkages, and spurs)**.** *In an instance graph, a $(2k \times \ell)$-nomadic linkage $\mathcal{L}$ is the concatenation of $2k$ many nomadic walks ("segments"), each of length $\ell+1$, in which the last directed edge of each walk is the reverse of first directed edge of the subsequent walk (including wrapping around from the $2k$th segment to the 1st). These $2k$ directed edges which are necessarily the reverse of the preceding directed edge are termed spurs. The weight of $\mathcal{L}$, denoted $\mathrm{wt}(\mathcal{L})$, is the product of the signs of the non-spur edges in $\mathcal{L}$.*

▶ **Definition 66** (Nonbacktracking $\mathcal{A}$-linkages)**.** *Recall that, strictly speaking, the nomadic property requires "remembering" which atom each edge comes from. Thus the $\mathcal{L}$ above is really associated to what we will call a $(2k \times 2\ell)$-nonbacktracking $\mathcal{A}$-linkage – call it $\mathcal{C}$ – in the underlying constraint graph. Formally:*

- *("linkage") $\mathcal{C}$ is a closed concatenation of $2k$ walks (called "segments") in the constraint graph, each consisting of $\ell+1$ length-2 variable-constraint-variable subpaths. The last such length-2 subpath in each segment ("spur") is equal to (the reverse of) the first length-2 subpath in the subsequent segment (including wraparound from the $2k$th segment to the 1st).*
- *("$\mathcal{A}$-linkage") For each length-2 subpath (v,f,v') in $\mathcal{C}$, where v is in variable group i, f is in constraint group j, and v' is in variable group i', it holds that $\{i,i'\}$ is an edge in A_j.*
- *("nonbacktracking") Each of the $2k$ segments is a nonbacktracking walk of length $2(\ell+1)$ in the constraint graph.*

We write $\mathrm{wt}(\mathcal{C}) \in \{\pm 1\}$ for the weight of the associated nomadic linkage in the instance graph.

Given these definitions, (12) tells us:

$$\operatorname{tr}((\boldsymbol{B}_n^{\ell}\boldsymbol{B}_n^{*\ell})^k) = \sum_{\substack{(2k\times 2\ell)\text{-nonbacktracking} \\ \mathcal{A}\text{-linkages } \mathcal{C} \text{ in } \boldsymbol{\mathcal{H}}_n}} \operatorname{wt}(\mathcal{C}). \tag{13}$$

Next, we make the observation that *if* $\boldsymbol{\mathcal{H}}_n$ proves to have all vertices ℓ-tangle-free, then we would get the same result if we only summed over "externally tangle-free" linkages.

▶ **Definition 67** (Externally tangle-free linkages)**.** *We say that a $(2k \times 2\ell)$-nonbacktracking linkage in a constraint graph $\mathcal{H}_n$ is externally ℓ-tangle-free if every vertex it touches is ℓ-tangle-free within $\mathcal{H}_n$. (The "externally" adjective emphasizes that we are concerned with cycles not just within the linkage's edges, but also among nearby edges of $\mathcal{H}_n$.)*

Thus in light of Proposition 64 we have:

▶ **Lemma 68.** *Provided $\ell \leqslant \kappa \log n$ for a certain universal $\kappa > 0$, we get that $\operatorname{tr}((\boldsymbol{B}_n^{\ell}\boldsymbol{B}_n^{*\ell})^k) = \boldsymbol{S}$ holds except with probability $O(1/n^{.99})$, where*

$$\boldsymbol{S} \coloneqq \sum_{\substack{(2k\times 2\ell)\text{-nonbacktracking} \\ \text{externally } \ell\text{-tangle-free} \\ \mathcal{A}\text{-linkages } \mathcal{C} \text{ in } \boldsymbol{\mathcal{H}}_n}} \operatorname{wt}(\mathcal{C}).$$

In order to apply Markov's inequality later, we will need the following technical claim:

▷ Claim 69. $\boldsymbol{S}$ is a nonnegative random variable.

Proof. Given $\boldsymbol{\mathcal{I}}_n$, recall that

$$\boldsymbol{B}_n^{\ell}[\vec{e},\vec{f}] = \sum_{\substack{\text{nomadic walks} \\ \vec{e}=\vec{e}_0,\vec{e}_1,\ldots,\vec{e}_\ell=\vec{f} \text{ in } \boldsymbol{\mathcal{I}}_n}} \operatorname{wt}(e_1)\operatorname{wt}(e_2)\cdots\operatorname{wt}(e_\ell).$$

Using a key idea of Bordenave (based on the "selective trace" of Friedman), define the related operator $\boldsymbol{B}_n^{(\ell)}$ via

$$\boldsymbol{B}_n^{(\ell)}[\vec{e},\vec{f}] = \sum_{\substack{\textit{externally } \ell\textit{-tangle-free} \text{ nomadic walks} \\ \vec{e}=\vec{e}_0,\vec{e}_1,\ldots,\vec{e}_\ell=\vec{f} \text{ in } \boldsymbol{\mathcal{I}}_n}} \operatorname{wt}(e_1)\operatorname{wt}(e_2)\cdots\operatorname{wt}(e_\ell),$$

where again the walk is said to be "externally ℓ-tangle-free" if every vertex it touches is ℓ-tangle-free with $\boldsymbol{\mathcal{H}}_n$. Then very similar to the analysis that gave us (12) and (13), we get that

$$\boldsymbol{S} = \operatorname{tr}((\boldsymbol{B}_n^{(\ell)}(\boldsymbol{B}_n^{(\ell)})^*)^k).$$

Thus $\boldsymbol{S}$ is visibly always nonnegative, being the trace of the kth power of the positive semidefinite matrix $\boldsymbol{B}_n^{(\ell)}(\boldsymbol{B}_n^{(\ell)})^*$. ◀

With these results in place, we can proceed to the main goal of the Trace Method: bounding $\mathbf{E}[\boldsymbol{S}]$. Such a bound can be used in the following lemma:

▶ **Lemma 70.** *Assume that $\ell \leqslant \kappa \log n$ and $k\ell = \omega(\log n)$. Then from $\mathbf{E}[\boldsymbol{S}] \leqslant R$ we may conclude that $\rho(\boldsymbol{B}_n) \leqslant (1 + o_n(1)) \cdot R^{\frac{1}{2k\ell}}$ holds, except with probability $O(1/n^{.99})$.*

Proof. Let $\boldsymbol{T} = \mathrm{tr}((\boldsymbol{B}_n^\ell \boldsymbol{B}_n^{*\ell})^k)$. On one hand, with λ denoting eigenvalues and σ denoting singular values, we have

$$\boldsymbol{T} \geqslant \lambda_{\max}((\boldsymbol{B}_n^\ell \boldsymbol{B}_n^{*\ell})^k) = \lambda_{\max}\left(\sqrt{\boldsymbol{B}_n^\ell \boldsymbol{B}_n^{*\ell}}\right)^{2k} = \sigma_{\max}(\boldsymbol{B}_n^\ell)^{2k} \geqslant \rho(\boldsymbol{B}_n^\ell)^{2k} = \rho(\boldsymbol{B}_n)^{2k\ell}.$$

On the other hand, since $\boldsymbol{S}$ is a nonnegative random variable (Claim 69), we can apply Markov's Inequality to deduce that $\boldsymbol{S} \leqslant n \cdot R$ except with probability at most $1/n$. Now from Lemma 68 we may infer that except with probability $O(1/n^{.99})$,

$$\boldsymbol{T} = \boldsymbol{S} \leqslant n \cdot R \quad \Longrightarrow \quad \rho(\boldsymbol{B}_n)^{2k\ell} \leqslant n \cdot R.$$

The result now follows by taking $2k\ell$-th roots. ◀

F.2 Eliminating singletons, and reduction to counting

Our next step toward bounding $\mathbf{E}[\boldsymbol{S}]$ is typical of the Trace Method: Rather than first choosing $\boldsymbol{\mathcal{H}}_n$ randomly and then summing over the linkages therein, we instead sum over all *potentially-appearing* linkages and insert an indicator that they actually appear in the realized random constraint graph. Defining

$\mathcal{K}_n$ = the "complete" constraint graph with cn constraint vertices and rn variable vertices,

this means that

$$\boldsymbol{S} = \sum_{\substack{(2k\times 2\ell)\text{-nonbacktracking} \\ \mathcal{A}\text{-linkages } \mathcal{C} \text{ in } \mathcal{K}_n}} 1[\mathcal{C} \text{ is in } \boldsymbol{\mathcal{H}}_n]\cdot 1[\mathcal{C} \text{ is externally } \ell\text{-tangle-free within } \boldsymbol{\mathcal{H}}_n]\cdot \mathrm{wt}_{\boldsymbol{\mathcal{I}}_n}(\mathcal{C}). \tag{14}$$

Here we wrote $\mathrm{wt}_{\boldsymbol{\mathcal{I}}_n}(\mathcal{C})$ to emphasize that even once $\mathcal{C}$ is in $\boldsymbol{\mathcal{H}}_n$ and is externally ℓ-tangle-free, its weight is still a random variable arising from the 1-wise uniform negations. These negations will create another simplification (one not available to Friedman/Bordenave). For this we will need another definition:

▶ **Definition 71** (Singleton-free $\mathcal{C}$'s). *Let $\mathcal{C}$ be a $(2k \times 2\ell)$-nonbacktracking circuit in $\mathcal{K}_n$. If there is an atom vertex that is passed through exactly once, we call it a singleton. If $\mathcal{C}$ contains no singleton, we call it singleton-free.*

Referring to (14), consider $\mathbf{E}[\boldsymbol{S}]$. If $\mathcal{C}$ contains any singleton, then it will contribute 0 to this expectation. The reason is that, provided $\mathcal{C}$ appears in $\boldsymbol{\mathcal{H}}_n$ and is externally ℓ-tangle-free therein, the 1-wise uniform negations will assign a uniformly random ± 1 sign to the edge engendered by $\mathcal{C}$'s singleton, and this sign will be independent of all other signs that go into $\mathrm{wt}_{\boldsymbol{\mathcal{I}}_n}(\mathcal{C})$. On the other hand, when $\mathcal{C}$ is singleton-free, we will simply upper-bound the (conditional) expectation of $\mathrm{wt}_{\boldsymbol{\mathcal{I}}_n}(\mathcal{C})$ by $+1$. We conclude that

$$\mathbf{E}[\boldsymbol{S}] \leqslant \sum_{\substack{(2k\times 2\ell)\text{-nonbacktracking} \\ \textit{singleton-free} \\ \mathcal{A}\text{-linkages } \mathcal{C} \text{ in } \mathcal{K}_n}} \mathbf{Pr}[\mathcal{C} \text{ is in } \boldsymbol{\mathcal{H}}_n \text{ and is externally } \ell\text{-tangle-free therein}]. \tag{15}$$

Let us now begin to simplify the probability calculation.

▶ **Definition 72** ($E(\mathcal{C})$, $V(\mathcal{C})$, $G(\mathcal{C})$)**.** *Let $\mathcal{C}$ be a $(2k \times 2\ell)$-nonbacktracking $\mathcal{A}$-linkage in $\mathcal{K}_n$. Write $E(\mathcal{C})$ for the set of undirected edges in $\mathcal{K}_n$ formed by "undirecting" all the directed edges in $\mathcal{C}$ (this includes reducing from a multiset to a set, if necessary). Then let $G(\mathcal{C})$ denote the undirected subgraph of $\mathcal{K}_n$ induced by $E(\mathcal{C})$, and write $V(\mathcal{C})$ for its vertices.*

Let's simplify the "tangle-freeness" situation.

▶ **Definition 73** (Internal tangle-free linkages)**.** *We say that a $(2k \times 2\ell)$-nonbacktracking linkage $\mathcal{C}$ in $\mathcal{K}_n$ is internally ℓ-tangle-free if every vertex it touches is ℓ-tangle-free within $G(\mathcal{C})$.*

We certainly have:

$$\text{linkage } \mathcal{C} \text{ not even internally } \ell\text{-tangle-free} \implies \mathbf{Pr}[\mathcal{C} \text{ is in } \boldsymbol{\mathcal{H}}_n \text{ and is externally } \ell\text{-tangle-free therein}] = 0.$$

Thus we can restrict the sum in (15) to internally ℓ-tangle-free linkages. Having done that, we will upper bound the sum by dropping this insistence on *external* tangle-freeness. Thus

$$\mathbf{E}[\boldsymbol{S}] \leqslant \sum_{\substack{(2k\times 2\ell)\text{-nonbacktracking} \\ \textit{interally } \ell\textit{-tangle-free}\text{, singleton-free} \\ \mathcal{A}\text{-linkages } \mathcal{C} \text{ in } \mathcal{K}_n}} \mathbf{Pr}[\mathcal{C} \text{ is in } \boldsymbol{\mathcal{H}}_n]. \tag{16}$$

We will now bound $\mathbf{Pr}[\mathcal{C}$ is in $\boldsymbol{\mathcal{H}}_n]$, so as to reduce all our remaining problems to counting. Towards this, recall that $\boldsymbol{\mathcal{H}}_n$ is a random n-lift of the complete graph $K_{r,c}$. One thing this implies is that every group-i variable-vertex in $\boldsymbol{\mathcal{H}}_n$ will have exactly one edge to each of c groups of constraint-vertices, and vice versa. Let us codify the $\mathcal{C}$'s that don't flagrantly violate this property:

▶ **Definition 74** (Valid $\mathcal{C}$'s)**.** *We say a $(2k \times 2\ell)$-nonbacktracking $\mathcal{A}$-linkage $\mathcal{C}$ in $\mathcal{K}_n$ is valid if $G(\mathcal{C})$ has the property that every variable-vertex in it is connected to at most 1 constraint-vertex from each of the c groups, and each constraint-vertex is connected to at most 1 variable-vertex from each of the r groups.*

Evidently, $\mathbf{Pr}[\mathcal{C}$ is in $\boldsymbol{\mathcal{H}}_n] = 0$ if $\mathcal{C}$ is invalid. Thus from (16) we can deduce:

$$\mathbf{E}[\boldsymbol{S}] \leqslant \sum_{\substack{(2k\times 2\ell)\text{-nonbacktracking} \\ \textit{valid}\text{, internally } \ell\text{-tangle-free, singleton-free} \\ \mathcal{A}\text{-linkages } \mathcal{C} \text{ in } \mathcal{K}_n}} \mathbf{Pr}[\mathcal{C} \text{ is in } \boldsymbol{\mathcal{H}}_n]. \tag{17}$$

Next, it is straightforward to show the following lemma (see Proposition A.8 of [24] for essentially the same observation):

▶ **Lemma 75.** *If $\mathcal{C}$ is a valid $(2k \times 2\ell)$-nonbacktracking $\mathcal{A}$-linkage in $\mathcal{K}_n$, and $k\ell = o(\sqrt{n})$, then*

$$\mathbf{Pr}[\mathcal{C} \text{ is in } \boldsymbol{\mathcal{H}}_n] = (1 + o_n(1)) \cdot n^{-|E(\mathcal{C})|}.$$

Proof. (Sketch.) Proceed through the edges in $E(\mathcal{C})$ in an arbitrary order. Each has approximately a $1/n$ chance of appearing in $\boldsymbol{\mathcal{H}}_n$, even conditioned on the appearance of the preceding edges. For example, this is exactly true for the first edge. For subsequent edges $e = \{u, v\}$, validity ensures that no preceding edge already connects u to a vertex in v's part, or vice versa. Thus the conditional probability of e appearing in $\boldsymbol{\mathcal{H}}_n$ is essentially the

probability that a particular edge appears in a random matching on $n+n$ vertices (which is $1/n$), except that a "small" number of vertex pairs may already have been matched. This "small" quantity is at most $|E(\mathcal{C})| \leqslant 4k\ell$, so the $1/n$ probability becomes $1/(n-4k\ell)$ at worst. Multiplying these conditional probabilities across all $|E(\mathcal{C})|$ edges yields a quantity that is off from $n^{-|E(\mathcal{C})|}$ by a factor of at most $(1+O(k\ell)/n)^{4k\ell} \leqslant 1+o_n(1)$, the inequality using $(k\ell)^2 = o(n)$. ◀

Combining this lemma with (17) and Lemma 70, we are able to reduce bounding $\rho(\boldsymbol{B}_n)$ to a counting problem:

▶ **Lemma 76.** *Assume that $\ell \leqslant \kappa \log n$ and $\omega(\log n) < k\ell < o(\sqrt{n})$. Then except with probability $O(1/n^{.99})$,*

$$\rho(\boldsymbol{B}_n) \leqslant (1+o_n(1)) \cdot R^{\frac{1}{2k\ell}}, \quad \text{where } R := \sum_{\substack{(2k\times 2\ell)\text{-nonbacktracking} \\ \text{valid, internally } \ell\text{-tangle-free, singleton-free} \\ \mathcal{A}\text{-linkages } \mathcal{C} \text{ in } \mathcal{K}_n}} n^{-|E(\mathcal{C})|}.$$

F.3 Tangle-free, singleton-free linkages are nearly duplicative

Our goal in this subsection is to show that each linkage $\mathcal{C}$ we sum over in Lemma 76 is "nearly duplicative": the number of variable-vertices is at most $(1+o(1))k\ell$, and the same is true of constraint-vertices – even though the obvious a priori upper bound for each of them is $2k\ell$. This factor-$\frac{1}{2}$ savings is precisely the source of the square-root in Theorem 60. We begin with a graph-theoretic lemma and then deduce the nearly-duplicative property.

▶ **Lemma 77.** *Let $\mathcal{C}$ be a $(2k \times 2\ell)$-nonbacktracking, internally ℓ-tangle-free linkage in $\mathcal{K}_n$. Assume $\log(k\ell) = o(\ell)$. Then $G(\mathcal{C})$ has at most $O(k \log(k\ell))$ vertices of degree exceeding 2.*

Proof. For brevity, let us write $G = G(\mathcal{C})$, $w = |V(\mathcal{C})|$, and note that we have a trivial upper bound of $w \leqslant 4k\ell$. Let t denote the number of cycles of length at most ℓ in G. By deleting at most t edges, we can form a graph $\widetilde{G}$ with girth at least ℓ. A theorem of Alon, Hoory, and Linial [2] implies that any (possibly irregular) graph with w vertices and girth at least ℓ must have average degree at most $2 + O(\log(w)/\ell)$ (this uses $\log(w) = o(\ell)$). Thus $\widetilde{G}$ has such a bound on its average degree. After restoring the deleted edges, we can still conclude that the average degree in G is at most $2 + O(\log(w)/\ell) + \frac{2t}{w}$. Writing w_1, w_2, w_{3+} for the number of vertices in G of degree 1, 2, and 3-or-more respectively, this means

$$2 + O(\log(w)/\ell) + \frac{2t}{w} \geqslant \frac{w_1 + 2w_2 + 3w_{3+}}{w} = \frac{w_1 + 2(w - w_1 - w_{3+}) + 3w_{3+}}{w} = 2 - \frac{w_1}{w} + \frac{w_{3+}}{w}$$
$$\implies \quad w_{3+} \leqslant O(w \log(w)/\ell) + w_1 + 2t.$$

The first term here is $O(k \log(k\ell))$ as desired, since $w \leqslant 4k\ell$. We will also show the next two terms are $O(k)$. Regarding w_1, degree-1 vertices in G can only arise from the spurs of $\mathcal{C}$, and hence $w_1 \leqslant 2k$. Finally, $2t \leqslant O(k)$ follows from the below claim combined with $w \leqslant 4k\ell$:

$$t \leqslant \frac{w}{2\ell} + 1. \tag{18}$$

We establish (18) using the tangle-free property of $\mathcal{C}$. Recall that t is the number of "short" cycles in G, meaning cycles of length at most ℓ. By the ℓ-tangle-free property of $\mathcal{C}$ (recalling the factor 4 in its definition), every $v \in V$ has at most one short cycle within distance 3ℓ of it. Thus if we choose paths in G that connect all short cycles (recall G is connected), then to each short cycle we can uniquely charge at least $3\ell - 1 \geqslant 2\ell$ vertices from these paths. It follows that $w = |V| \geqslant 2\ell(t-1)$, establishing (18). ◀

▶ **Corollary 78.** *In the setting of Lemma 77, assume also that $\mathcal{C}$ is singleton-free and valid. Then the number of variable-vertices $\mathcal{C}$ visits is at most $k\ell + O(k\log(k\ell))$, and the same is true of constraint-vertices.*

Proof. Think of $\mathcal{C}$ as a succession of $2k(\ell+1)$ "two-steps", where a two-step is a length-2 directed path going from a variable-vertex, to a constraint-vertex, to a (distinct) variable-vertex. Call two such two-steps "duplicates" if they use the same three variables (possibly going in the opposite direction). We claim that "almost all" two-steps have at least one duplicate. To see this, consider the constraint-vertex in some two-step a. Since $\mathcal{C}$ is singleton-free, at least one other two-step b must pass through the constraint-vertex of a. If b is not a duplicate of a, then this constraint-vertex will have degree exceeding 2 in $G(\mathcal{C})$. By Lemma 77 there are at most $O(k\log(k\ell))$ such constraint-vertices. Further, by validity each constraint-vertex can support at most $\binom{r}{2} = O(1)$ unduplicated two-steps. Thus at most $O(k\log(k\ell))$ of the $2k(\ell+1)$ two-steps are unduplicated.

Now imagine we walk through the two-steps of $\mathcal{C}$ in succession. Each two-step can visit at most one "new" variable-vertex and one "new" constraint-vertex. However each two-step which is a duplicate of a previously-performed two-step visits no new vertices. Among the $2k(\ell+1)$ two-steps, at most $O(k\log(k\ell))$ are unduplicated. Thus at least $(2k(\ell+1) - O(k\log(k\ell)))/2 = k(\ell+1) - O(k\log(k\ell))$ two-steps are duplicates of previously-performed two-steps. It follows that at most $k(\ell+1) + O(k\log(k\ell))$ two-steps visit any new vertex. This completes the proof. ◀

F.4 The final countdown

We now wish to count the objects summed in the definition of R from Lemma 76. The remainder of this section will be devoted to proving:

▶ **Theorem 79.** *For every $\varepsilon > 0$, except with probability $O(1/n^{.99})$,*

$$\rho(\boldsymbol{B}_n) \leqslant (1+o_n(1)) \cdot (1+\varepsilon) \cdot \sqrt{\rho(|B_1|)}.$$

The bulk of the technical matter in the proof of Theorem 79 will involve analyzing

$$(2k \times 2\ell)\text{-nonbacktracking, valid, internally } \ell\text{-tangle-free, singleton-free, } \mathcal{A}\text{-linkages } \mathcal{C} \quad (19)$$

in $\mathcal{K}_n$.

▶ **Definition 80** (Steps: stale, fresh, and boundary)**.** *We call each of the $4k(\ell+1)$ directed edges from which $\mathcal{C}$ is composed a step. If we imagine traversing these steps in order, they "reveal" vertices and edges of $G(\mathcal{C})$ as we go along. We call a step stale if the edge it traverses was previously traversed in $\mathcal{C}$ (in some direction). Note that both endpoints of the edge must also have been previously visited. Otherwise, if the step traverses a "new" edge, it will be designated either "fresh" or "boundary". It is designated fresh if the vertex it reaches was never previously visited in $\mathcal{C}$. Otherwise, the step is boundary; i.e., the step goes between two previously-visited vertices, but along a new edge. For the purposes of defining fresh/boundary, we specify that the initial vertex of $\mathcal{C}$ is always considered to be "previously visited".*

The following facts are immediate:

▶ **Fact 81.** *The number of fresh steps in $\mathcal{C}$ is $|V(\mathcal{C})| - 1$. (The -1 accounts for the fact that the initial vertex is considered "previously visited".) Since the number of fresh and boundary steps together is $|E(\mathcal{C})|$, it follows that the number of boundary steps is $|E(\mathcal{C})| - |V(\mathcal{C})| + 1$.*

▶ **Definition 82.** *We write* $\mathrm{Lkgs}(f,b)$ *for the collection of linkages as in* (19) *having exactly* f *fresh edges and* b *boundary edges.*

Our goal is to show:

▶ **Lemma 83.** *For every* $\hat{\rho} > \rho(|B_1|)$ *we have:*

$$|\mathrm{Lkgs}(f,b)| \leqslant \mathrm{poly}(k,\ell)^{b+k} \cdot n^{f+1} \cdot \hat{\rho}^{f/2}$$

where the constants in the poly *factor depend on* $\hat{\rho}$.

Before proving this lemma, observe that many linkages are the same modulo the labels between 1 and n that are defined by the lifting. To make this formal we first introduce some notation and follow by using it to aid in the proof of Lemma 83.

Given a linkage $\mathcal{C}$ we write $\mathcal{C} = ((v_1,i_1),(v_2,i_2),\ldots,(v_{4k(\ell+1)},i_{4k(\ell+1)}))$, where (v_j,i_j) are vertices from $\mathcal{K}_n$ and v_j indicates the base vertex (from $K_{r,c}$) and i_j is an integer (between 1 and n) that indicates the lifted copy. This notation means that $\mathcal{C}$ traverses this sequence of vertices in this order.

▶ **Definition 84** (Isomorphism of linkages)**.** *Given two linkages* $\mathcal{C}$ *and* $\mathcal{C}'$ *that visit* $|V(\mathcal{C})| = |V(\mathcal{C}')|$ *vertices, we say they are isomorphic if are the same modulo the labels between* 1 *and* n *that are defined by the lifting. Formally, letting* $\mathcal{C} = ((v_1,i_1),\ldots,(v_{4k(\ell+1)},i_{4k(\ell+1)}))$ *and* $\mathcal{C}' = ((v'_1,i'_1),\ldots,(v'_{4k(\ell+1)},i'_{4k(\ell+1)}))$*, there exist permutations* π_v *on* $[n]$ *for each* $v \in V(K_{r,c})$ *such that for all* j *we have* $v'_j = v_j$ *and* $i'_j = \pi_{v_j}(i_j)$.

This isomorphism relation induces equivalence classes for which we want to assign representative elements. We do so as follows.

▶ **Definition 85** (Canonical linkages)**.** *A linkage* $\mathcal{C}$ *is said to be canonical if for every vertex* $v \in K_{r,c}$*, if* $\mathcal{C}$ *visits* j *distinct lifted copies of* v *then it first visits* $(v,1)$*, then* $(v,2),\ldots,$ *and finally* (v,j)*. We write* $\mathrm{Lkgs}^c(f,b)$ *for the collection of canonical linkages as in* (19) *having exactly* f *fresh steps and* b *boundary steps.*

▶ **Proposition 86.** $|\mathrm{Lkgs}(f,b)| \leqslant n^{f+1}|\mathrm{Lkgs}^c(f,b)|$.

Proof. It suffices to show that for every canonical linkage $\mathcal{C} \in \mathrm{Lkgs}^c(f,b)$, it has at most n^{f+1} isomomorphic linkages $\mathcal{C}' \in \mathrm{Lkgs}(f,b)$. By Fact 81, $\mathcal{C}$ visits exactly $f+1$ distinct vertices, call them $\{(v^{(1)},i^{(1)}),\ldots,(v^{(f+1)},i^{(f+1)})\}$. Every isomorphic $\mathcal{C}'$ may be obtained by taking a list of numbers $(i'_1,\ldots,i'_{f+1}) \in [n]^{f+1}$ and replacing all appearances of $(v^{(j)},i^{(j)})$ in $\mathcal{C}$ with $(v^{(j)},i'_j)$. (Not all such lists lead to isomorphic $\mathcal{C}'$, but we don't mind overcounting.) This completes the proof, as there are n^{f+1} such lists. ◀

We now have all the tools to prove the desired lemma.

Proof of Lemma 83. With Proposition 86 in place, it suffices to bound the number of canonical linkages as follows:

$$|\mathrm{Lkgs}^c(f,b)| \leqslant \mathrm{poly}(k,\ell)^{b+k} \cdot \hat{\rho}^{f/2}.$$

Our strategy is to give an encoding of linkages in $\mathrm{Lkgs}^c(f,b)$, and then bound the number of possible encodings. Let $\mathcal{C}$ be an arbitrary linkage in $\mathrm{Lkgs}^c(f,b)$. To encode $\mathcal{C}$, we first partition it into $2k$ many "$2(\ell+1)$-segments", each of which corresponds to nonbacktracking walks between spurs,and specify how to encode each $2(\ell+1)$-segment. We then partition each $2(\ell+1)$-segment into maximal contiguous blocks of the same type of step ("type" as in Definition 80) and store an encoding of information about the steps therein. Ultimately, it will be possible to uniquely decipher $\mathcal{C}$ from its constructed encoding.

Towards describing our encoding, we first define the sequence S_{visited}, constructed from the $f+1$ vertices in $V(\mathcal{C})$ sorted in increasing order of first-visit time.

Encoding positions of blocks. We define P_{fresh}, P_{boundary} and P_{stale}, which are sequences noting the starting positions and ending positions of fresh, boundary, and stale blocks respectively, in the order visited in $\mathcal{C}$.

Encoding fresh steps. Let S_{fresh} be the sequence obtained by replacing each vertex of S_{visited} with its corresponding base vertex in $K_{r,c}$.

Encoding boundary steps. Let β be a block of boundary steps $(v_0, v_1), \ldots, (v_{|\beta|-1}, v_{|\beta|})$. Let t_i be such that v_i is the t_i-th vertex in S_{visited}. We define $\text{Enc}_b(\beta)$ as the sequence $(t_0, t_1), \ldots, (t_{|\beta|-1}, t_{|\beta|})$. Let $\beta_1, \ldots, \beta_T$ be the blocks of boundary steps in the order in which they appear in $\mathcal{C}$. We store the concatenation of $\text{Enc}_b(\beta_1), \ldots, \text{Enc}_b(\beta_T)$, which we call S_{boundary}.

Encoding stale steps. For each block β of stale steps, let u be the first vertex and v be the last vertex of β, and let $p(\beta)$ be the position in $\mathcal{C}$ where the block β starts. Let $\mathcal{S}_{p(\beta),uv,|\beta|}$ denote the list (in, say, lexicographic order) of all possible nonbacktracking walks from u to v of length $|\beta|$ that only use edges visited by $\mathcal{C}$ before position $p(\beta)$; note that β occurs in $\mathcal{S}_{p(\beta),uv,|\beta|}$. We let $\text{Enc}_s(\beta) = (t, m)$ such that the t-th vertex in S_{visited} is the last vertex visited in β (that is v), and m is the position of β in $\mathcal{S}_{p(\beta),uv,|\beta|}$. Let $\beta_1, \ldots, \beta_T$ be the blocks of stale steps in the order they appear in $\mathcal{C}$. We store the concatenation of $\text{Enc}_s(\beta_1), \ldots, \text{Enc}_s(\beta_T)$, which we call S_{stale}.

We refer to the constructed $(P_{\text{fresh}}, P_{\text{boundary}}, P_{\text{stale}}, S_{\text{fresh}}, S_{\text{boundary}}, S_{\text{stale}})$ as the *encoding* of $\mathcal{C}$.

Unique reconstruction of linkage. In this part of the proof, we show that we can uniquely recover $\mathcal{C}$ from its encoding. First, since $\mathcal{C}$ is a canonical linkage we can correctly reconstruct S_{visited} from S_{fresh} because the labels are visited in canonical (increasing) order. From $P_{\text{fresh}}, P_{\text{boundary}}$ and P_{stale}, we can infer a partition of $[4k(\ell+1)]$ into blocks in order $\beta_1, \ldots, \beta_T$ and the type of each block. We sketch an inductive proof that shows how $\mathcal{C}$ can be uniquely recovered from its encoding. As our base case, the first block is a fresh block and hence all the steps that comprise it can be recovered from S_{visited}. Towards our inductive step, suppose we know the edges in $\mathcal{C}$ from blocks $\beta_1, \ldots, \beta_i$, we show how to recover the edges in β_{i+1} from the encoding of $\mathcal{C}$. If β_{i+1} is a fresh or boundary block, its recovery is straightforward. Suppose β_{i+1} is a stale block. Then from P_{stale} and S_{stale}, we can infer the last vertex v visited by β_{i+1} and the length of the block $|\beta_{i+1}|$. We know the first vertex u in β_{i+1} and can reconstruct $\mathcal{S}_{p(\beta_{i+1}),uv,|\beta_{i+1}|}$ since we have complete information about the steps in $\mathcal{C}$ prior to β_{i+1}. We can then infer β_{i+1} from $\mathcal{S}_{p(\beta_{i+1}),uv,|\beta_{i+1}|}$ and S_{stale}.

Bounding the number of metadata encodings. A fresh block must either be followed by a boundary step, or must occur at the end of a $2(\ell+1)$-segment; analogously, a stale block must either be preceded by a boundary step, or must occur at the start of a $2(\ell+1)$-segment. Thus, the number of fresh blocks and stale blocks are each bounded by $b + 2k$. Further, the number of boundary blocks is clearly bounded by b. Since there are at most $(4k(\ell+1))^2$ distinct combinations of starting and ending positions of a block, the number of distinct possibilities that the triple $(P_{\text{fresh}}, P_{\text{stale}}, P_{\text{boundary}})$ can be bounded by $(4k(\ell+1))^{6b+8k}$.

Bounding number of fresh step encodings. For a fixed P_{fresh}, we give an upper bound on the number of possibilities for S_{fresh}. Fixing P_{fresh} fixes a number T as well as $q_1, \ldots, q_T$ such that there are T fresh blocks in $\mathcal{C}$ and such that the i-th block has length q_i. Let us

focus on a single fresh block β. The sequence of vertices in S_{fresh} corresponding to β give a nonbacktracking walk W_β in the base constraint graph $K_{r,c}$. Additionally, for a consecutive triple (i, j, i') in this nonbacktracking walk, $\{i, i'\}$ must be an edge in the corresponding base instance graph $\mathcal{I}_1$ due $\mathcal{C}$ being an $\mathcal{A}$-linkage. Let $\widetilde{W}_\beta$ be the maximal subwalk of W_β that starts and ends with a variable vertex. Note that $\widetilde{W}_\beta$ corresponds exactly to a nomadic walk in $\mathcal{I}_1$ whose length is at most $|\beta|/2$. Now regarding W_β, either W_β is equal to $\widetilde{W}_\beta$ (there is 1 way in which this can happen), or both the first and last steps of W_β are not in $\widetilde{W}_\beta$ (there are c^2 ways in which this can happen), or exactly one of the first and last steps of W_β is not in $\widetilde{W}_\beta$ (there are $2c$ ways in which this can happen). This tells us that the number of distinct possibilities for W_β is bounded by $(c+1)^2\delta_{\lfloor|\beta|/2\rfloor}$, where δ_s denotes the number of nomadic walks of length s in $\mathcal{I}_1$. Thus, we obtain an upper bound of $(c+1)^{2T}\prod_{i=1}^{T}\delta_{\lfloor q_i/2\rfloor}$ on the number of possibilities for S_{fresh}, which is bounded by $(c+1)^{2b+4k}\prod_{i=1}^{T}\delta_{\lfloor q_i/2\rfloor}$. Towards simplifying the expression, we bound δ_s. Observe that for a given edge $e \in E(|\mathcal{I}_1|)$, the number of nomadic walks of length s starting with e is given by $\|(|B_1|)^s\mathbf{1}_e\|_1$. This implies that $\delta_s \leqslant \|(|B_1|)^s\|_1$, where $\|(|B_1|)^s\|_1 = \sup\{\|(|B_1|)^s x\| : \|x\|_1 = 1\}$.

To bound the above, first observe that we have a simple bound $\|(|B_1|)^s\|_1 \leqslant \kappa^s$ provided κ is a large enough constant (for example, the maximum degree of $\mathcal{I}_1$ is a possible such value). Next, it is known that

$$\lim_{s\to\infty}\left(\|(|B_1|)^s\|\right)^{1/s} = \rho(|B_1|),$$

and hence for any $\hat{\rho} > \rho(|B_1|)$, there is a constant ℓ_0 such that $\|(|B|)^s\|_1 \leqslant (\hat{\rho})^s$ for all $s \geqslant \ell_0$. Putting these two bounds together we get that for any $s \geqslant \ell_0$,

$$\delta_s \leqslant \|(|B_1|)^s\|_1 \leqslant (\hat{\rho})^{s-\ell_0}\kappa^{\ell_0}.$$

Thus the number of possibilities for S_{fresh} is bounded by $(c+1)^{2b+4k}\prod_{i=1}^{T}(\hat{\rho})^{\lfloor q_i/2\rfloor-\ell_0}\kappa^{\ell_0}$, which can, in turn, be bounded by $\left((c+1)^2\kappa^{\ell_0}\hat{\rho}^{-\ell_0}\right)^{b+2k}(\hat{\rho})^{f/2}$.

Bounding number of stale step encodings. For any stale block β, let u and v be the first and last visited vertices respectively. S_{stale} specifies a number in $[f+1]$ to encode v, and a number between 1 and M where M is the total number of nonbacktracking walks from u to v of length $|\beta|$. Since the number of stale blocks is bounded by $b+2k$, the number of possibilities for what S_{stale} can be is at most $(M(f+1))^{b+2k}$. We show that $M \leqslant 2$, and hence translate our upper bound to $(2(f+1))^{b+2k}$.

Since all blocks are contained within $2(\ell+1)$-segments and the $\mathcal{A}$-linkage being encoded is 4ℓ-tangle-free, the steps traversed by β are in a connected subgraph H with at most one cycle. Our goal is to show that there are at most 2 nonbacktracking walks of a given length L between any pair of vertices x, y. There is at most one nonbacktracking walk between x and y that does not visit vertices on C, the single cycle in H, and if such a walk exists, it is the unique shortest path. Any nonbacktracking walk between x and y that visits vertices of C can be broken down into 3 phases – (i) a nonbacktracking walk from x to v_x, the closest vertex in C to x, (ii) a nonbacktracking walk from v_x to v_y, the closest vertex in C to y, (iii) a nonbacktracking walk from v_y to y. Phases (i) and (iii) are always of fixed length, whose sum is some L'. Thus, it suffices to show that there are at most 2 nonbacktracking walks from v_x to v_y of length $L - L'$. Any nonbacktracking walk takes r rotations in C and then takes an acyclic path from v_x to v_y, whose length is observed to be strictly less than $|C|$, for $r \geqslant 0$. The steps in a nonbacktracking walk from v_x to v_y are either all in a clockwise direction, or all in an anticlockwise direction, and hence for any r there are at most 2 nonbacktracking walks from v_x to v_y of length strictly between $(r-1)|C|$ and $r|C|+1$. In particular, there are at most 2 nonbacktracking walks between v_x and v_y of length equal to $L - L'$.

Bounding number of boundary step encodings. S_{boundary} is a sequence of b tuples in $[f+1]^2$, and hence there are at most $(f+1)^{2b}$ distinct sequences that S_{boundary} can be.

Final bound. The above gives us a final bound of:

$$(4k(\ell+1))^{6b+8k}((c+1)^2\kappa^{\ell_0}(\hat{\rho})^{-\ell_0})^{b+2k}(\hat{\rho})^{f/2}2^{b+2k}(f+1)^{3b+2k} \tag{20}$$

which, when combined with Proposition 86 gives the desired claim. ◀

We wrap everything up by combining the results of Lemma 83 with Lemma 76 to prove Theorem 79.

Proof of Theorem 79. Let $\ell = \kappa \log n$, where κ is the universal constant from Proposition 64, let k be chosen so that $k\ell = \omega(\log n)$, let R be as in Lemma 76, and let $\hat{\rho}$ be any constant greater than $\rho(|B_1|)$. Then we have

$$\begin{aligned}
R &= \sum_{\substack{(2k\times 2\ell)\text{-nonbacktracking} \\ \text{valid, internally } \ell\text{-tangle-free, singleton-free} \\ \mathcal{A}\text{-linkages } \mathcal{C} \text{ in } \mathcal{K}_n}} n^{-|E(\mathcal{C})|} \\
&= \sum_{f=0}^{\infty}\sum_{b=0}^{\infty} |\mathrm{Lkgs}(f,b)| n^{-(f+b)} \\
&= \sum_{f=0}^{2k\ell+O(k\log(k\ell))}\sum_{b=0}^{\infty} |\mathrm{Lkgs}(f,b)| n^{-(f+b)} && \text{(by Corollary 78)} \\
&\leqslant \sum_{f=0}^{2k\ell+O(k\log(k\ell))}\sum_{b=0}^{\infty} \frac{\mathrm{poly}(k,\ell)^b \cdot \mathrm{poly}(k,\ell)^k \cdot (\hat{\rho})^{f/2} \cdot n}{n^b} && \text{(by Lemma 83)} \\
&= \sum_{f=0}^{2k\ell+O(k\log(k\ell))} n \cdot \mathrm{poly}(k,\ell)^k \cdot (\hat{\rho})^{f/2} \sum_{b=0}^{\infty} \left(\frac{\mathrm{poly}(k,\ell)}{n}\right)^b \\
&= \sum_{f=0}^{2k\ell+O(k\log(k\ell))} n \cdot \mathrm{poly}(k,\ell)^k \cdot (\hat{\rho})^{f/2} \cdot \left(\frac{1}{1-\frac{\mathrm{poly}(k,\ell)}{n}}\right) \\
&\leqslant 2n \cdot \mathrm{poly}(k,\ell)^k (2k\ell + O(k\log(k\ell)))(\hat{\rho})^{k\ell+O(k\log(k\ell))}
\end{aligned}$$

For the choice of k and ℓ in the theorem statement, we can use Lemma 76 to conclude that

$$\rho(\boldsymbol{B}_n) \leqslant (1+o_n(1)) \cdot \sqrt{\hat{\rho}}.$$

with probability $1 - O(n^{.99})$. Since the above bound holds for any $\hat{\rho} > \rho(|B_1|)$, for any $\varepsilon > 0$, it can be rewritten as

$$\rho(\boldsymbol{B}_n) \leqslant (1+o_n(1)) \cdot (1+\varepsilon) \cdot \sqrt{\rho(|B_1|)}.$$ ◀

G The SDP value for random two-eigenvalue CSPs

In this section, we put all the ingredients together to conclude our main theorem. We start with an elementary and well known fact and include a short proof for self containment.

▶ **Fact 87.** *Let A be a real $n \times n$ symmetric matrix. Then*

$$\frac{1}{n} \max_{X \succeq 0, X_{ii}=1} \langle A, X \rangle \leqslant \lambda_{\max}(A)$$
$$\frac{1}{n} \min_{X \succeq 0, X_{ii}=1} \langle A, X \rangle \geqslant \lambda_{\min}(A)$$

Proof. We prove the upper bound below. The proof of the lower bound is identical.

$$\begin{aligned} \frac{1}{n} \max_{X \succeq 0, X_{ii}=1} \langle A, X \rangle &\leqslant \frac{1}{n} \max_{X \succeq 0, \mathrm{tr}(X)=n} \langle A, X \rangle \\ &= \max_{X \succeq 0, \mathrm{tr}(X)=1} \langle A, X \rangle \\ &= \lambda_{\max}(A). \end{aligned}$$

◀

Recall $\alpha_{\mathrm{gr}} := (c-1)(-\lambda_1\lambda_2)$ and $r_X := 2\sqrt{\alpha_{\mathrm{gr}}}$.

▶ **Theorem 88.** *Let $\mathcal{A} = (A_1, \ldots, A_c)$ be a sequence of r-vertex atoms with edge weights ± 1. Let $\boldsymbol{\mathcal{H}}_n$ denote a random n-lifted constraint graph and $\boldsymbol{\mathcal{I}}_n = \mathcal{A}(\boldsymbol{\mathcal{H}}_n)$ an associated instance graph with 1-wise uniform negations $(\boldsymbol{\xi}^f_{ii'})$. Let $\boldsymbol{A}_n$ be the adjacency matrix of $\boldsymbol{\mathcal{I}}_n$. Then, with probability $1 - o_n(1)$,*

$$\max_{X \succeq 0, X_{ii}=1} \langle \boldsymbol{A}_n, X \rangle = (\lambda_1 + \lambda_2 + r_X \pm \varepsilon)n$$
$$\min_{X \succeq 0, X_{ii}=1} \langle \boldsymbol{A}_n, X \rangle = (\lambda_1 + \lambda_2 - r_x \pm \varepsilon)n.$$

Proof. $\max_{X \succeq 0, X_{ii}=1} \langle \boldsymbol{A}_n, X \rangle \geqslant (\lambda_1 + \lambda_2 + r_X - \varepsilon)n$ follows from Theorem 48 and $\max_{X \succeq 0, X_{ii}=1} \langle \boldsymbol{A}_n, X \rangle \leqslant (\lambda_1 + \lambda_2 + r_X + \varepsilon)n$ follows from Fact 87. The upper and lower bounds on $\min_{X \succeq 0, X_{ii}=1} \langle \boldsymbol{A}_n, X \rangle$ can be determined identically. ◀

Asymptotic Divergences and Strong Dichotomy

Xiang Huang
Le Moyne College, Syracuse, NY 13214, USA
Iowa State University, Ames, IA 50011, USA
huangx@iastate.edu

Jack H. Lutz
Iowa State University, Ames, IA 50011, USA
lutz@iastate.edu

Elvira Mayordomo
Departamento de Informática e Ingeniería de Sistemas,
Instituto de Investigación en Ingeniería de Aragón,
Universidad de Zaragoza, 50018 Zaragoza, Spain
elvira@unizar.es

Donald M. Stull
Iowa State University, Ames, IA 50011, USA
dstull@iastate.edu

Abstract

The *Schnorr-Stimm dichotomy theorem* [31] concerns finite-state gamblers that bet on infinite sequences of symbols taken from a finite alphabet Σ. The theorem asserts that, for any such sequence S, the following two things are true.

(1) If S is not normal in the sense of Borel (meaning that every two strings of equal length appear with equal asymptotic frequency in S), then there is a finite-state gambler that wins money at an infinitely-often exponential rate betting on S.

(2) If S is normal, then any finite-state gambler betting on S loses money at an exponential rate betting on S.

In this paper we use the Kullback-Leibler divergence to formulate the *lower asymptotic divergence* $\mathrm{div}(S||\alpha)$ of a probability measure α on Σ from a sequence S over Σ and the *upper asymptotic divergence* $\mathrm{Div}(S||\alpha)$ of α from S in such a way that a sequence S is α-normal (meaning that every string w has asymptotic frequency $\alpha(w)$ in S) if and only if $\mathrm{Div}(S||\alpha) = 0$. We also use the Kullback-Leibler divergence to quantify the *total risk* $\mathrm{Risk}_G(w)$ that a finite-state gambler G takes when betting along a prefix w of S.

Our main theorem is a *strong dichotomy theorem* that uses the above notions to *quantify* the exponential rates of winning and losing on the two sides of the Schnorr-Stimm dichotomy theorem (with the latter routinely extended from normality to α-normality). Modulo asymptotic caveats in the paper, our strong dichotomy theorem says that the following two things hold for prefixes w of S.

(1′) The infinitely-often exponential rate of winning in 1 is $2^{\mathrm{Div}(S||\alpha)|w|}$.

(2′) The exponential rate of loss in 2 is $2^{-\mathrm{Risk}_G(w)}$.

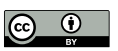
We also use (1′) to show that $1 - \mathrm{Div}(S||\alpha)/c$, where $c = \log(1/\min_{a\in\Sigma}\alpha(a))$, is an upper bound on the finite-state α-dimension of S and prove the dual fact that $1 - \mathrm{div}(S||\alpha)/c$ is an upper bound on the finite-state strong α-dimension of S.

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

Keywords and phrases finite-state dimension, finite-state gambler, Kullback-Leibler divergence, normal sequences

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.51

Funding *Xiang Huang*: This research was supported in part by National Science Foundation Grants 1247051, 1545028, and 1900716.
Jack H. Lutz: This research was supported in part by National Science Foundation Grants 1247051, 1545028, and 1900716.



Elvira Mayordomo: Part of this work was supported by a grant from the Spanish Ministry of Science, Innovation and Universities (TIN2016-80347-R) and was partly done during a research stay at the Iowa State University supported by National Science Foundation Research Grant 1545028.
Donald M. Stull: This research was supported in part by National Science Foundation Grants 1247051, 1545028, and 1900716.

Acknowledgements We thank students in Iowa State University's fall, 2017, advanced topics in computational randomness course for listening to a preliminary version of this research. We thank three anonymous reviewers of an earlier draft of this paper for useful comments and corrections.

1 Introduction

An infinite sequence S over a finite alphabet is *normal* in the 1909 sense of Borel [7] if every two strings of equal length appear with equal asymptotic frequency in S. Borel normality played a central role in the origins of measure-theoretic probability theory [6] and is intuitively regarded as a weak notion of randomness. For a masterful discussion of this intuition, see section 3.5 of [22], where Knuth calls normal sequences "∞-distributed sequences."

The theory of computing was used to make this intuition precise. This took place in three steps in the 1960s and 1970s. First, Martin-Löf [28] used constructive measure theory to give the first successful formulation of the randomness of individual infinite binary sequences. Second, Schnorr [30] gave an equivalent, and more flexible, formulation of Martin-Löf's notion in terms of gambling strategies called *martingales*. In this formulation, an infinite binary sequences S is random if no lower semicomputable martingale can make unbounded money betting on the successive bits of S. Third, Schnorr and Stimm [31] proved that an infinite binary sequence S is normal if and only if no martingale that is computed by a finite-state automaton can make unbounded money betting on the successive bits of S. That is, *normality is finite-state randomness.*

This equivalence was a breakthrough that has already had many consequences (discussed later in this introduction), but the Schnorr-Stimm result said more. It is a *dichotomy theorem* asserting that, for any infinite binary sequence S, the following two things are true.

1. If S is not normal, then there is a finite-state gambler that makes money at an infinitely-often exponential rate when betting on S.
2. If S is normal, then every finite-state gambler that bets infinitely many times on S loses money at an exponential rate.

The main contribution of this paper is to *quantify* the exponential rates of winning and losing on the two sides (1 and 2 above) of the Schnorr-Stimm dichotomy.

To describe our main theorem in some detail, let Σ be a finite alphabet. It is routine to extend the above notion of normality to an arbitrary probability measure α on Σ. Specifically, an infinite sequence S over Σ is *α-normal* if every finite string w over Σ appears with asymptotic frequency $\alpha^{|w|}(w)$ in S, where α^ℓ is the natural (product) extension of α to strings of length ℓ. Schnorr and Stimm [31] correctly noted that their dichotomy theorem extends to α-normal sequences in a straightforward manner, and it is this extension whose exponential rates we quantify here.

The quantitative tool that drives our approach is the Kullback-Leibler divergence [23], also known as the relative entropy [12]. If α and β are probability measures on Σ, then the *Kullback-Leibler divergence of β from α* is

$$D(\alpha||\beta) = E_\alpha \log \frac{\alpha}{\beta},$$

i.e., the expectation with respect to α of the random variable $\log \frac{\alpha}{\beta} : \Sigma \to \mathbb{R} \cup \{\infty\}$, where the logarithm is base-2. Although the Kullback-Leibler divergence is not a metric on the space of probability measures on Σ, it does quantify "how different" β is from α, and it has the crucial property that $D(\alpha||\beta) \geq 0$, with equality if and only if $\alpha = \beta$.

Here we use the empirical frequencies of symbols in S to define the *asymptotic lower divergence* $\operatorname{div}(S||\alpha)$ of α from S and the *asymptotic upper divergence* $\operatorname{Div}(S||\alpha)$ of α from S in a natural way, so that S is α-normal if and only if $\operatorname{Div}(S||\alpha) = 0$.

The first part of our *strong dichotomy theorem* says that the infinitely-often exponential rate that can be achieved in 1 above is essentially at least $2^{\operatorname{Div}(S||\alpha)|w|}$, where w is the prefix of S on which the finite-state gambler has bet so far. More precisely, it says the following.

1′. If S is not α-normal, then, for every $\gamma < 1$, there is a finite-state gambler G such that, when G bets on S with payoffs according to α, there are infinitely many prefixes w of S after which G's capital exceeds $2^{\gamma \operatorname{Div}(S||\alpha)|w|}$.

The second part of our strong dichotomy theorem, like the second part of the Schnorr-Stimm dichotomy theorem, is complicated by the fact that a finite-state gambler may, in some states, decline to bet. In this case, its capital after a bet is the same as it was before the bet, regardless of what symbol actually appears in S. Once again, however, it is the Kullback-Leibler divergence that clarifies the situation. As explained in section 3 below, in any particular state q, a finite-state gambler's betting strategy is a probability measure $B(q)$ on Σ. If $B(q) = \alpha$, then the gambler does not bet in state q. We thus define the *risk* that the gambler G takes in state q to be

$$\operatorname{risk}_G(q) = D(\alpha||B(q)),$$

i.e., the divergence of $B(q)$ from not betting. We then define the *total risk* that the gambler takes along a prefix w of the sequence S on which it is betting to be the sum $\operatorname{Risk}_G(w)$ of the risks $\operatorname{risk}_G(q)$ in the states that G traverses along w. The second part of our strong dichotomy theorem says that, if S is α-normal and G is a finite-state gambler betting on S, then after each prefix w of S, the capital of G on prefixes w of S is essentially bounded above by $2^{-\operatorname{Risk}_G(w)}$. In some sense, then, G loses all that it risks. More precisely, the second part of our strong dichotomy says the following.

2′. If S is α-normal, then, for every finite-state gambler G and every $\gamma < 1$, after all but finitely many prefixes w of S, the gambler G's capital is less than $2^{-\gamma \operatorname{Risk}_G(w)}$.

A routine ergodic argument, already present in [31], shows that, if a finite-state gambler G bets on an α-normal sequence S, then every state of G that occurs infinitely often along S occurs with positive frequency along S. Hence 2 above follows from 2′ above.

Our strong dichotomy theorem has implications for finite-state dimensions. For each probability measure α on Σ and each sequence S over Σ, the *finite-state α-dimension* $\dim^{\alpha}_{\mathrm{FS}}(S)$ and the *finite-state strong α-dimension* $\operatorname{Dim}^{\alpha}_{\mathrm{FS}}(S)$ (defined in section 4 below) are finite-state versions of Billingsley dimension [5, 10] introduced in [26]. When α is the uniform probability measure on Σ, these are the finite dimension $\dim_{\mathrm{FS}}(S)$, introduced in [14] as a finite-state version of Hausdorff dimension [20, 17], and the finite-state strong dimension $\operatorname{Dim}_{\mathrm{FS}}(S)$, introduced in [2] as a finite-state version of packing dimension [35, 34, 17]. Intuitively, $\dim^{\alpha}_{\mathrm{FS}}(S)$ and $\operatorname{Dim}^{\alpha}_{\mathrm{FS}}(S)$ measure the lower and upper asymptotic α-densities of the finite-state information in S.

Here we use part 1 of our strong dichotomy theorem to prove that, for every positive probability measure α on Σ and every sequence S over Σ,

$$\dim^{\alpha}_{\mathrm{FS}}(S) \leq 1 - \mathrm{Div}(S||\alpha)/c,$$

where $c = \log(1/\min_{a\in\Sigma} \alpha(a))$. We also establish the dual result that, for all such α and S,

$$\mathrm{Dim}^{\alpha}_{\mathrm{FS}}(S) \leq 1 - \mathrm{div}(S||\alpha)/c.$$

Research on normal sequences and normal numbers (real numbers whose base-b expansions are normal sequences for various choices of b) connected the theory of normal numbers so directly to the theory of computing. Further work along these lines has been continued in [21, 29, 3, 33]. After the discovery of algorithmic dimensions in the present century [24, 25, 14, 2], the Schnorr-Stimm dichotomy led to the realization [8] that the finite-state world, unlike any other known to date, is one in which maximum dimension is not only necessary, but also sufficient, for randomness. This in turn led to the discovery of nontrivial extensions of classical theorems on normal numbers [11, 36] to new quantitative theorems on finite-state dimensions [19, 16], a line of inquiry that will certainly continue. It has also led to a polynomial-time algorithm [4] that computes real numbers that are provably absolutely normal (normal in every base) and, via Lempel-Ziv methods, to a nearly linear time algorithm for this [27]. In parallel with these developments, connections among normality, Weyl equidistribution theorems, and Diophantine approximation have led to a great deal of progress surveyed in the books [15, 9]. This paragraph does not begin to do justice to the breadth and depth of recent and ongoing research on normal numbers and their growing involvement with the theory of computing. It is to be hoped that our strong dichotomy theorem and the quantitative methods implicit in it will further accelerate these discoveries.

2 Divergence and normality

This section reviews the discrete Kullback-Leibler divergence, introduces asymptotic extensions of this divergence, and uses these to give useful characterizations of Borel normal sequences.

2.1 The Kullback-Leibler divergence

We work in a finite alphabet Σ with $2 \leq |\Sigma| < \infty$. We write Σ^{ℓ} for the set of strings of length ℓ over Σ, $\Sigma^* = \bigcup_{\ell=0}^{\infty} \Sigma^{\ell}$ for the set of (finite) *strings* over Σ, Σ^{ω} for the set of (infinite) *sequences* over Σ, and $\Sigma^{\leq\omega} = \Sigma^* \cup \Sigma^{\omega}$. We write λ for the empty string, $|w|$ for the length of a string $w \in \Sigma^*$, and $|S| = \omega$ for the length of a sequence $S \in \Sigma^{\omega}$. For $x \in \Sigma^{\leq\omega}$ and $0 \leq i < |x|$, we write $x[i]$ for the i-th symbol in x, noting that $x[0]$ is the leftmost symbol in x. For $x \in \Sigma^{\leq\omega}$ and $0 \leq i \leq j < |x|$, we write $x[i..j]$ for the string consisting of the i-th through j-th symbols in x. Specially, we write $x \upharpoonright n$ to mean $x[0..n-1]$. For $x,\ y \in \Sigma^{\leq\omega}$, we write $x \sqsubseteq y$ if x is a prefix of y. We write $x \sqsubset y$ to denote x being a strict prefix of y, which excludes the case $x = y$.

A *(discrete) probability measure* on a nonempty finite set Ω is a function $\pi : \Omega \to [0, 1]$ satisfying

$$\sum_{\omega\in\Omega} \pi(\omega) = 1. \tag{2.1}$$

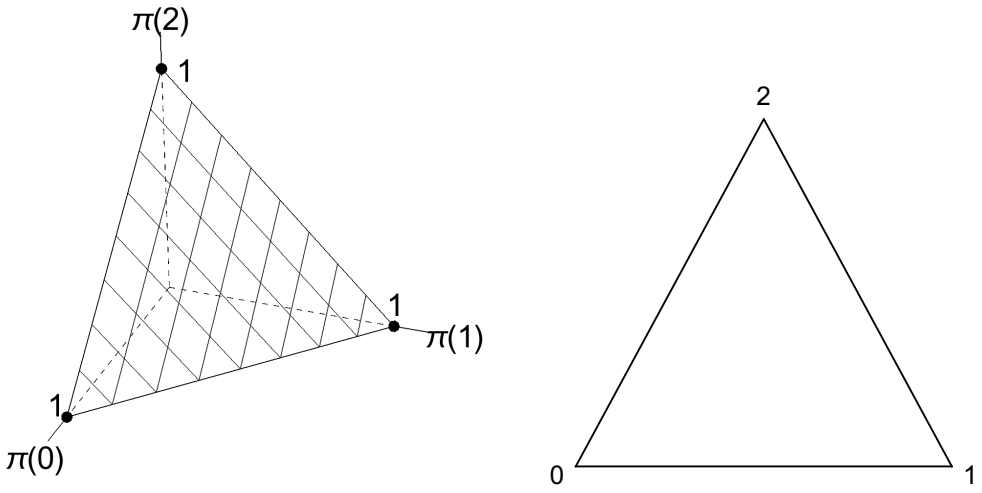


Figure 1 Two views of the simplex $\Delta(\{0,1,2\})$.

We write $\Delta(\Omega)$ for the set of all probability measures on Ω, $\Delta^+(\Omega)$ for the set of all $\pi \in \Delta(\Omega)$ that are *strictly positive* (i.e., $\pi(\omega) > 0$ for all $\omega \in \Omega$), $\Delta_{\mathbb{Q}}(\Omega)$ for the set of all $\pi \in \Delta(\Omega)$ that are rational-valued, and $\Delta^+_{\mathbb{Q}}(\Omega) = \Delta^+(\Omega) \cap \Delta_{\mathbb{Q}}(\Omega)$. In this paper we are most interested in the case where $\Omega = \Sigma^\ell$ for some $\ell \in \mathbb{Z}^+$.

Intuitively, we identify each probability measure $\pi \in \Delta(\Omega)$ with the point in $\mathbb{R}^{|\Omega|}$ whose coordinates are the probabilities $\pi(\omega)$ for $\omega \in \Omega$. By (2.1) this implies that $\Delta(\Omega)$ is the $(|\Omega| - 1)$-dimensional simplex in $\mathbb{R}^{|\Omega|}$ whose vertices are the points at 1 on each of the coordinate axes. (See Figure 1 for an illustration with $|\Omega| = 3$.) For each $\omega \in \Omega$, the vertex on axis ω is the degenerate probability measures π_ω with $\pi_\omega(\omega) = 1$. The centroid of the simplex $\Delta(\Omega)$ is the uniform probability measure on Ω, and the (topological) interior of $\Delta(\Omega)$ is $\Delta^+(\Omega)$. We write $\partial\Delta(\Omega) = \Delta(\Omega) \setminus \Delta^+(\Omega)$ for the boundary of $\Delta(\Omega)$.

▶ **Definition.** ([23]). Let $\alpha, \beta \in \Delta(\Omega)$, where Ω is a nonempty finite set. The *Kullback-Leibler divergence* (or *KL-divergence*) *of* β *from* α is

$$D(\alpha||\beta) = E_\alpha \log \frac{\alpha}{\beta}, \tag{2.2}$$

where the logarithm is base-2.

Note that the right-hand side of (2.2) is the α-expectation of the random variable

$$\log \frac{\alpha}{\beta} : \Omega \longrightarrow \mathbb{R}$$

defined by

$$\left(\log \frac{\alpha}{\beta}\right)(\omega) = \log \frac{\alpha(\omega)}{\beta(\omega)}$$

for each $\omega \in \Omega$. Hence (2.2) is a convenient shorthand for

$$D(\alpha||\beta) = \sum_{\omega \in \Omega} \alpha(\omega) \log \frac{\alpha(\omega)}{\beta(\omega)}.$$

Note also that $D(\alpha||\beta)$ is infinite if and only if $\alpha(\omega) > 0 = \beta(\omega)$ the some $\omega \in \Omega$.

The Kullback-Leibler divergence $D(\alpha||\beta)$ is a useful measure of how different β is from α. It is not a metric (because it is not symmetric and does not satisfy the triangle inequality), but it has the crucial property that $D(\alpha||\beta) \geq 0$, with equality if and only if $\alpha = \beta$. The two most central quantities in Shannon information theory, entropy and mutual information, can both be defined in terms of divergence as follows.

1. *Entropy is divergence from certainty.* The *entropy* of a probability measure $\alpha \in \Delta(\Omega)$, conceived by Shannon [32] as a measure of the uncertainty of α, is

$$H(\alpha) = \sum_{\omega \in \Omega} \alpha(\omega) D(\pi_\omega||\alpha), \tag{2.3}$$

   i.e., the α-average of the divergences of α from the "certainties" π_ω.
2. *Mutual information is divergence from independence.* If α, $\beta \in \Delta(\Omega)$ have a joint probability measure $\gamma \in \Delta(\Omega \times \Omega)$ (i.e., are the marginal probability measures of γ), then the *mutual information* between α and β, conceived by Shannon [32] as a measure of the information shared by α and β, is

$$I(\alpha\,;\,\beta) = D(\alpha\beta||\gamma), \tag{2.4}$$

   i.e., the divergence of γ from the probability measure in which α and β are independent.

Two additional properties of the Kullback-Leibler divergence are useful for our asymptotic concerns. First, the divergence $D(\alpha||\beta)$ is continuous on $\Delta(\Omega)^2$ (as a function into $[0, \infty]$). Hence, if $\alpha_n \in \Delta(\Omega)$ for each $n \in \mathbb{N}$ and $\lim_{n\to\infty} \alpha_n = \alpha$ in the sense of the Euclidean metric on the simplex $\Delta(\Omega)$, then $\lim_{n\to\infty} D(\alpha_n||\alpha) = \lim_{n\to\infty} D(\alpha||\alpha_n) = 0$. Second, the converse holds. It is well known [12] that

$$D(\alpha||\beta) \geq \frac{1}{2\ln 2}\|\alpha - \beta\|_1^2,$$

where $\|\alpha - \beta\|_1 = \sum_{\omega\in\Omega}|\alpha(\omega) - \beta(\omega)|$ is the $\mathcal{L}_1$-norm. Hence, if either $\lim_{n\to\infty} D(\alpha_n||\alpha) = 0$ or $\lim_{n\to\infty} D(\alpha||\alpha_n) = 0$, then $\lim_{n\to\infty} \alpha_n = \alpha$.

More extensive discussions of the Kullback-Leibler divergence appear in [12, 13].

2.2 Asymptotic divergences

For nonempty strings $w, x \in \Sigma^*$, we write

$$\#_\square(w, x) = \left|\{m \leq \frac{|x|}{|w|} - 1 \;\middle|\; x[m|w|..(m+1)|w| - 1] = w\}\right|$$

for the *number of block occurrences* of w in x. Note that $0 \leq \#_\square(w, x) \leq \frac{|x|}{|w|}$.

For each $S \in \Sigma^\omega$, $n \in \mathbb{Z}^+$, and $\lambda \neq w \in \Sigma^*$, the n-th *block frequency* of w in S is

$$\pi_{S,n}(w) = \frac{\#_\square(w, S[0..n|w|-1])}{n} \tag{2.5}$$

Note that (2.5) defines, for each $S \in \Sigma^\omega$ and $n \in \mathbb{Z}^+$, a function

$$\pi_{S,n} : \Sigma^* \smallsetminus \{\lambda\} \longrightarrow \mathbb{Q}.$$

For each such S and n and each $\ell \in \mathbb{Z}^+$, let $\pi_{S,n}^{(\ell)} = \pi_{S,n} \restriction \Sigma^\ell$ be the restriction of the function $\pi_{S,n}$ to the set Σ^ℓ of strings of length ℓ.

▶ **Observation 2.1.** For each $S \in \Sigma^\omega$ and n, $\ell \in \mathbb{Z}^+$,

$$\pi^{(\ell)}_{S,n} \in \Delta_{\mathbb{Q}}(\Sigma^\ell),$$

i.e., $\pi^{(\ell)}_{S,n}$ is a rational-valued probability measure on Σ^ℓ.

We call $\pi^{(\ell)}_{S,n}$ the n-th *empirical probability measure on* Σ^ℓ *given by* S.

A probability measure $\alpha \in \Delta(\Sigma)$ naturally induces, for each $\ell \in \mathbb{Z}^+$, a probability measure $\alpha^{(\ell)} \in \Delta(\Sigma^\ell)$ defined by

$$\alpha^{(\ell)}(w) = \prod_{i=0}^{|w|-1} \alpha(w[i]). \tag{2.6}$$

The empirical probability measures $\pi^{(\ell)}_{S,n}$ provide a natural way to define useful empirical divergences of probability measures from sequences.

▶ **Definition.** Let $\ell \in \mathbb{Z}^+$, $S \in \Sigma^\omega$, and $\alpha \in \Delta(\Sigma)$.

1. The *lower ℓ-divergence of α from S* is $\mathrm{div}_\ell(S||\alpha) = \liminf_{n\to\infty} D(\pi^{(\ell)}_{S,n}||\alpha^{(\ell)})$.
2. The *upper ℓ-divergence of α from S* is $\mathrm{Div}_\ell(S||\alpha) = \limsup_{n\to\infty} D(\pi^{(\ell)}_{S,n}||\alpha^{(\ell)})$.
3. The *lower divergence of α from S* is $\mathrm{div}(S||\alpha) = \sup_{\ell\in\mathbb{Z}^+} \mathrm{div}_\ell(S||\alpha)/\ell$.
4. The *upper divergence of α from S* is $\mathrm{Div}(S||\alpha) = \sup_{\ell\in\mathbb{Z}^+} \mathrm{Div}_\ell(S||\alpha)/\ell$.

A similar approach gives useful empirical divergences of one sequence from another.

▶ **Definition.** Let $\ell \in \mathbb{Z}^+$ and $S, T \in \Sigma^\omega$.

1. The *lower ℓ-divergence of T from S* is $\mathrm{div}_\ell(S||T) = \liminf_{n\to\infty} D(\pi^{(\ell)}_{S,n}||\,\pi^{(\ell)}_{T,n})$.
2. The *upper ℓ-divergence of T from S* is $\mathrm{Div}_\ell(S||T) = \limsup_{n\to\infty} D(\pi^{(\ell)}_{S,n}||\,\pi^{(\ell)}_{T,n})$.
3. The *lower divergence of T from S* is $\mathrm{div}(S||T) = \sup_{\ell\in\mathbb{Z}^+} \mathrm{div}_\ell(S||T)/\ell$.
4. The *upper divergence of T from S* is $\mathrm{Div}(S||T) = \sup_{\ell\in\mathbb{Z}^+} \mathrm{Div}_\ell(S||T)/\ell$.

2.3 Normality

The following notions are essentially due to Borel [7].

▶ **Definition.** Let $\alpha \in \Delta(\Sigma)$, $S \in \Sigma^\omega$, and $\ell \in \mathbb{Z}^+$.

1. S is *α-ℓ-normal* if, for all $w \in \Sigma^\ell$,

$$\lim_{n\to\infty} \pi_{S,n}(w) = \alpha^{(\ell)}(w).$$

2. S is *α-normal* if, for all $\ell \in \mathbb{Z}^+$, S is α-ℓ-normal.
3. S is *ℓ-normal* if S is μ-ℓ-normal, where μ is the uniform probability measure on Σ.
4. S is *normal* if, for all $\ell \in \mathbb{Z}^+$, S is ℓ-normal.

▶ **Lemma 2.2.** *For all $\alpha \in \Delta(\Sigma)$, $S \in \Sigma^\omega$, and $\ell \in \mathbb{Z}^+$, the following four conditions are equivalent.*

(1) *S is α-ℓ-normal.*
(2) $\mathrm{Div}_\ell(S||\alpha) = 0$.
(3) *For every α-ℓ-normal sequence $T \in \Sigma^\omega$, $\mathrm{Div}_\ell(S||T) = 0$.*
(4) *There exists an α-ℓ-normal sequence $T \in \Sigma^\omega$ such that $\mathrm{Div}_\ell(S||T) = 0$.*

Lemma 2.2 (proved in the appendix) immediately implies the following.

▶ **Theorem 2.3** (divergence characterization of normality). *For all $\alpha \in \Delta(\Sigma)$ and $S \in \Sigma^\omega$, the following conditions are equivalent.*

(1) *S is α-normal.*
(2) $\mathrm{Div}(S||\alpha) = 0$.
(3) *For every α-normal sequence $T \in \Sigma^\omega$, $\mathrm{Div}(S||T) = 0$.*
(4) *There exists an α-normal sequence $T \in \Sigma^\omega$ such that $\mathrm{Div}(S||T) = 0$.*

3 Strong Dichotomy

This section presents our main theorem, the strong dichotomy theorem for finite-state gambling. We first review finite-state gamblers.

Fix a finite alphabet Σ with $|\Sigma| \geq 2$.

▶ **Definition** ([31, 18, 14]). A *finite-state gambler (FSG)* is a 4-tuple

$$G = (Q, \delta, s, B),$$

where Q is a finite set of *states*, $\delta : Q \times \Sigma \to Q$ is the *transition function*, $s \in Q$ is the *start state*, and $B : Q \to \Delta_{\mathbb{Q}}(\Sigma)$ is the *betting function*.

The transition structure (Q, δ, s) here works as in any deterministic finite-state automaton. For $w \in \Sigma^*$, we write $\delta(w)$ for the state reached from s by processing w.

Intuitively, a gambler $G = (Q, \delta, s, B)$ bets on the successive symbols of a sequence $S \in \Sigma^\omega$. The payoffs in the betting are determined by a *payoff probability measure* $\alpha \in \Delta(\Sigma)$. (We regard α and S as external to the gambler G.) We write $d_{G,\alpha}(w)$ for the gambler G's capital (amount of money) after betting on the successive bits of a prefix $w \sqsubseteq S$, and we assume that the initial capital is $d_{G,\alpha}(\lambda) = 1$.

The meaning of the betting function B is as follows. After betting on a prefix $w \sqsubseteq S$, the gambler is in state $\delta(w) \in Q$. The betting function B says that, for each $a \in \Sigma$, the gambler bets the fraction $B(\delta(w))(a)$ of its current capital $d_{G,\alpha}(w)$ that $wa \sqsubseteq S$, i.e., that the next symbol of S is an a. If it then turns out to be the case that $wa \sqsubseteq S$, the gambler's capital will be

$$d_{G,\alpha}(wa) = d_{G,\alpha}(w)\frac{B(\delta(w))(a)}{\alpha(a)}. \tag{3.1}$$

(Note: If $\alpha(a) = 0$ here, we may define $d_{G,\alpha}(wa)$ however we wish.)

The payoffs in (3.1) are fair with respect to α, which means that the conditional α-expectation

$$\sum_{a \in \Sigma} \alpha(a) d_{G,\alpha}(wa)$$

of $d_{G,\alpha}(wa)$, given that $w \sqsubseteq S$, is exactly $d_{G,\alpha}(w)$. This says that the function $d_{G,\alpha}$ is an *α-martingale*.

If $\delta(w) = q$ is a state in which $B(q) = \alpha$, then (3.1) says that, for each $a \in \Sigma$, $d_{G,\alpha}(wa) = d_{G,\alpha}(w)$. That is, the condition $B(q) = \alpha$ means that G *does not bet* in state q. Accordingly, we define the *risk* that G takes in a state $q \in Q$ to be

$$\mathrm{risk}_G(q) = D(\alpha||B(q)).$$

i.e., the divergence of $B(q)$ from not betting. We also define the *total risk* that G takes *along* a string $w \in \Sigma^*$ to be

$$\mathrm{Risk}_G(w) = \sum_{u \sqsubset w} \mathrm{risk}_G(\delta(u)).$$

We now state our main theorem.

▶ **Theorem 3.1** (strong dichotomy theorem). *Let $\alpha \in \Delta(\Sigma)$, $S \in \Sigma^\omega$, and $\gamma < 1$.*

1. *If S is not α-normal, then there is a finite-state gambler G such that, for infinitely many prefixes $w \sqsubseteq S$,*

$$d_{G,\alpha}(w) > 2^{\gamma\,\mathrm{Div}(S||\alpha)|w|}.$$

2. *If S is α-normal, then, for every finite-state gambler G, for all but finitely many prefixes $w \sqsubseteq S$,*

$$d_{G,\alpha}(w) < 2^{-\gamma\,\mathrm{Risk}_G(w)}.$$

Proof. To prove the first part, let S be a non-normal sequence. Then by Theorem 2.3, we know that $\mathrm{Div}(S||\alpha) > 0$. Let $r < 1$ and let $\epsilon > 0$. By the definition of $\mathrm{Div}(S||\alpha)$ there must exist ℓ such that

$$\mathrm{Div}_\ell(S||\alpha)/\ell > r\,\mathrm{Div}(S||\alpha). \tag{3.2}$$

That is

$$\limsup_{n\to\infty} D(\pi^{(\ell)}_{S,n}||\alpha^{(\ell)}) > \ell r\,\mathrm{Div}(S||\alpha).$$

We can pick a subsequence of indices n_k's, such that $\lim_{k\to\infty} D(\pi^{(\ell)}_{S,n_k}||\alpha^{(\ell)}) = \mathrm{Div}_\ell(S||\alpha)$. Therefore by inequality (3.2)

$$D(\pi^{(\ell)}_{S,n_k}||\alpha^{(\ell)}) > \ell r\,\mathrm{Div}(S||\alpha) \tag{3.3}$$

for sufficiently large k. In particular, by compactness of $[0,1]^{|\Sigma^\ell|}$ equipped with $\mathcal{L}_1$-norm, we can further request that

$$\lim_{k\to\infty} \pi^{(\ell)}_{S,n_k} \quad \text{exists.} \tag{3.4}$$

Let $\pi_0 = \pi_0(r,m) \in \Delta_{\mathbb{Q}}(\Sigma^\ell)$ be the m-th $\pi^{(\ell)}_{S,n_k}$ that satisfies (3.3), indexed by k. By the way we define π_0, we have

$$D(\pi_{S,n_k}||\alpha^{(\ell)}) \geq D(\pi_0||\alpha^{(\ell)}) > \ell r\,\mathrm{Div}(S||\alpha), \tag{3.5}$$

and

$$\|\pi_0 - \pi^{(\ell)}_{S,n_k}\| \to 0, \quad \text{as } m \to \infty \text{ and } k \to \infty, \tag{3.6}$$

whence D's continuity in section 2.1 tells us that

$$D(\pi^{(\ell)}_{S,n_k}||\pi_0) \to 0, \quad \text{as } m \to \infty \text{ and } k \to \infty. \tag{3.7}$$

Also note that,

$$\lim_{m\to\infty} D(\pi_0||\alpha^{(\ell)}) = \lim_{k\to\infty} D(\pi^{(\ell)}_{S,n_k}||\alpha^{(\ell)}) = \mathrm{Div}_\ell(S||\alpha) > 0. \tag{3.8}$$

For a fixed $\pi_0 = \pi_0(r, m)$, by the definition, for any n_k sufficiently large, we have

$$D(\pi^{(\ell)}_{S,n_k} || \alpha^{(\ell)}) > D(\pi_0 || \alpha^{(\ell)})(1-\epsilon) > 0. \tag{3.9}$$

By doing the above we pick a probability measure π_0 that is "far" away from $\alpha^{(\ell)}$, we now hard code π_0 in a gambler $G = (Q, \delta, s, B)$, where

$$Q = \Sigma^{\leq \ell-1}, \qquad \delta(w,a) = \begin{cases} wa & \text{if } |wa| < \ell \\ \lambda & \text{if } |wa| = \ell \end{cases}, \qquad s = \lambda, \qquad \textit{and} \qquad B(w)(a) = \pi_0(a|w),$$

where $\pi_0(a|w)$ describes the conditional probability (induced by π_0) of occurrence of an a after $w \in Q$, and is defined by $\pi_0(a|w) = \pi_0(wa)/\pi_0(w)$, where for $u \in Q$, the notation $\pi_0(u)$ is defined recursively by $\pi_0(w) = \sum_{a\in\Sigma} \pi_0(wa)$.

Let $u = a_0 \cdots a_{\ell-1}$ be in Σ^ℓ. The following observation captures the above intuition:

$$\frac{B(\lambda)(a_0)\cdots B(u[0..\ell-2])(a_{\ell-1})}{\alpha(a_0)\cdots\alpha(a_{\ell-1})} = \frac{\pi_0(u)}{\alpha^{(\ell)}(u)}.$$

Now let $w = S \upharpoonright n_k$ for some k. We can view w as

$$w = u_0 u_1 \cdots u_{n-1} u_n, \quad \text{where } |u_i| = \ell \text{ for } 0 \leq i \leq n-1 \text{ and } u_n = a_0 \cdots a_m \text{ with } m < \ell.$$

Then we have

$$d_{G,\alpha}(w) = \Big(\prod_0^{n-1} \frac{\pi_0(u_i)}{\alpha^{(\ell)}(u_i)}\Big) \frac{B(\lambda)(a_0)\cdots B(u_n[0..m-1])(a_m)}{\alpha(a_0)\cdots\alpha(a_m)} \geq C_0 \prod_0^{n-1} \frac{\pi_0(u_i)}{\alpha^{(\ell)}(u_i)}, \tag{3.10}$$

where C_0 is the minimum value of $\frac{B(\lambda)(a_0)\cdots B(u_n[0..m-1])(a_m)}{\alpha(a_0)\cdots\alpha(a_{\ell-1})}$, where $u_n = a_0\cdots a_m$ ranges over $\Sigma^{<\ell}$. Taking log on both sides of (3.10) we get

$$\begin{aligned}
\log d_{G,\alpha}(w) - \log C_0 &\geq \sum_{i=0}^{n-1} \log \frac{\pi_0(u_i)}{\alpha^{(\ell)}(u_i)} = \sum_{|u|=\ell} \#_\square(u,w) \log \frac{\pi_0(u)}{\alpha^{(\ell)}(u)}, \\
&= n \sum_{|u|=\ell} \frac{\#_\square(u,w)}{n} \log \frac{\pi_0(u)}{\alpha^{(\ell)}(u)} = n \sum_{|u|=\ell} \pi^{(\ell)}_{S,n}(u) \log \frac{\pi_0(u)}{\alpha^{(\ell)}(u)}, \\
&= n \sum_{|u|=\ell} \Big[\pi^{(\ell)}_{S,n}(u) \log \frac{\pi^{(\ell)}_{S,n}(u)}{\alpha^{(\ell)}(u)} - \pi^{(\ell)}_{S,n}(u) \log \frac{\pi^{(\ell)}_{S,n}(u)}{\pi_0(u)}\Big] \\
&= n\Big(D(\pi^{(\ell)}_{S,n} || \alpha^{(\ell)}) - D(\pi^{(\ell)}_{S,n} || \pi_0)\Big)
\end{aligned} \tag{3.11}$$

Then by (3.9), for $w = S \upharpoonright n_k$ long enough, we have

$$\begin{aligned}
\log d_{G,\alpha}(w) - \log C_0 &\geq n\Big(D(\pi^{(\ell)}_{S,n} || \alpha^{(\ell)}) - D(\pi^{(\ell)}_{S,n} || \pi_0)\Big) \\
&\geq n\Big(D(\pi_0 || \alpha^{(\ell)})(1-\epsilon) - D(\pi^{(\ell)}_{S,n} || \pi_0)\Big) \geq \frac{|w|}{\ell} D(\pi_0 || \alpha^{(\ell)})(1-2\epsilon).
\end{aligned}$$

Therefore, by (3.5) we have

$$d_{G,\alpha}(w) \geq C_0 2^{\frac{|w|(1-2\epsilon)}{\ell} D(\pi_0 || \alpha^{(\ell)})} \geq 2^{|w| r(1-2\epsilon)\, \mathrm{Div}(S||\alpha)}.$$

Since r and $1-2\epsilon$ can be picked arbitrary close to 1, take $r(1-2\epsilon) > \gamma$, then

$$d_{G,\alpha}(w) \geq 2^{\gamma\, \mathrm{Div}(S||\alpha)|w|}$$

for $w = S \upharpoonright n_k$ long enough.

We now prove the second part of the main theorem.

Let S be a normal number, G an arbitrary finite-state gambler. By Proposition 2.5 of [31], $G = (Q, \delta, s, B)$ will eventually reach a *bottom strongly connected component* (a component that has no path to leave) when processing S. A similar argument can also be found in [33]. Without loss of generality, we will therefore assume that every state in G is recurrent in processing S.

Let $w = a_0 \cdots a_{n-1} \sqsubseteq S$. Then

$$\begin{aligned} d_{G,\alpha}(w) &= \frac{B(\delta(\lambda))(a_0) \cdots B(\delta(w[a_0..a_{n-2}]))(a_{n-1})}{\alpha(a_0) \cdots \alpha(a_{n-1})} \\ &= \prod_{q \in Q} \prod_{a \in \Sigma} \left(\frac{B(q)(a)}{\alpha(a)} \right)^{\#_{G,w}(q,a)}, \end{aligned} \tag{3.12}$$

where the notation $\#_{G,w}(q, a)$ denotes the number of times G lands on state q and the next symbol is a while processing w. Similarly, we use the notation $\#_{G,w}(q)$ to denote the number of times G lands on q in the same process.

Taking the logarithm of both sides of (3.12), we have

$$\begin{aligned} \log d_{G,\alpha}(w) &= \sum_{q \in Q} \sum_{a \in \Sigma} \#_{G,w}(q, a) \log \frac{B(q)(a)}{\alpha(a)} \\ &= \sum_{q \in Q} \#_{G,w}(q) \sum_{a \in \Sigma} \frac{\#_{G,w}(q, a)}{\#_{G,w}(q)} \log \frac{B(q)(a)}{\alpha(a)} \end{aligned} \tag{3.13}$$

By a result of Agafonov [1], which extends easily to the arbitrary probability measures considered here, we have that, for every state q, the limit of $\frac{\#_{G,w}(q,a)}{\#_{G,w}(s)}$ along S exists and converges to $\alpha(a)$. That is

$$\lim_{w \to S} \frac{\#_{G,w}(q, a)}{\#_{G,w}(s)} = \alpha(a), \tag{3.14}$$

for every state q.

Therefore, by equations (3.13) and (3.14), and the fact that there are finitely many states, we have

$$\begin{aligned} \log d_{G,\alpha}(w) &\leq \sum_{q \in Q} \#_{G,w}(q) \sum_{a \in \Sigma} (\alpha(a) + o(1)) \log \frac{B(q)(a)}{\alpha(a)} \\ &= \sum_{q \in Q} -\operatorname{risk}_G(q) \#_{G,w}(q) + \sum_{q \in Q} \#_{G,w}(q) \sum_{a \in \Sigma} o(1) \log \frac{B(q)(a)}{\alpha(a)} \\ &= -\operatorname{Risk}_G(w) + \sum_{q \in Q} \#_{G,w}(q) \sum_{a \in \Sigma} o(1) \log \frac{B(q)(a)}{\alpha(a)} \\ &= \operatorname{Risk}_G(w)(-1 + o(1)). \end{aligned}$$

It follows that

$$d_{G,\alpha}(w) \leq 2^{-(1+o(1)) \operatorname{Risk}_B(w)},$$

so part 2 of the theorem holds. ◀

4 Dimension

Finite-state dimensions give a particularly sharp formulation of part 1 of the strong dichotomy theorem, along with a dual of this result.

Finite-state dimensions were introduced for the uniform probability measure on Σ in [14, 2] and extended to arbitrary probability measure on Σ in [26]. For each $\alpha \in \Delta(\Sigma)$ and each $S \in \Sigma^\omega$, define the sets

$$\mathfrak{G}^\alpha(S) = \left\{ s \in [0,\infty) \,\middle|\, (\exists \text{FSG } G) \limsup_{w \to S} \alpha^{|w|}(w)^{1-s} d_{G,\alpha}(w) = \infty \right\}$$

and

$$\mathfrak{G}^\alpha_{\text{str}}(S) = \left\{ s \in [0,\infty) \,\middle|\, (\exists \text{FSG } G) \liminf_{w \to S} \alpha^{|w|}(w)^{1-s} d_{G,\alpha}(w) = \infty \right\}$$

The limits superior and inferior here are taken for successively longer prefixes $w \sqsubseteq S$. The "strong" subscript of $\mathfrak{G}_{\text{str}}(S)$ refers to the fact that $\alpha^{|w|}(w)^{1-s} d_{G,\alpha}(w)$ is required to converge to infinity in a stronger sense than in $\mathfrak{G}^\alpha(S)$.

▶ **Definition** ([26]). Let $\alpha \in \Delta(\Sigma)$ and $S \in \Sigma^\omega$.

1. The *finite-state α-dimension* of S is $\dim^\alpha_{\text{FS}}(S) = \inf \mathfrak{G}^\alpha(S)$.
2. The *finite-state strong α-dimension* of S is $\text{Dim}^\alpha_{\text{FS}}(S) = \inf \mathfrak{G}^\alpha_{\text{str}}(S)$

It is easy to see that, for all $\alpha \in \Delta^+(\Sigma)$ and $S \in \Sigma^\omega$, $0 \le \dim^\alpha_{\text{FS}}(S) \le \text{Dim}^\alpha_{\text{FS}}(S) \le 1$.

▶ **Theorem 4.1.** *For all $\alpha \in \Delta(\Sigma)$ and $S \in \Sigma^\omega$ let $c = \log(1/\min_{a\in\Sigma} \alpha(a))$. Then,*

$$\dim^\alpha_{\text{FS}}(S) \le 1 - \text{Div}(S||\alpha)/c$$

and

$$\text{Dim}^\alpha_{\text{FS}}(S) \le 1 - \text{div}(S||\alpha)/c.$$

Proof. Let $t < \text{Div}(S||\alpha)/c$, and let $s = 1 - t$. Fix ℓ such that $\text{Div}_\ell(S||\alpha)/\ell > tc$, then for i.o. n, $D(\pi^{(\ell)}_{S,n}||\alpha^{(\ell)}) > \ell tc$. Note that $\alpha^{|w|}(w) \ge (1/2^c)^{|w|}$ for every $w \in \Sigma^*$.

Define the gambler G be $G = (Q, \delta, s_0, B_n)$, where $Q = \Sigma^{\le \ell-1}$,

$$\delta(w,a) = \begin{cases} wa & \text{if } |wa| < \ell \\ \lambda & \text{if } |wa| = \ell \end{cases}$$

$s_0 = \lambda$, and $B_n(w)(a) = \pi^{(\ell)}_{S,n}(a|w)$, where $\pi^{(\ell)}_{S,n}(a|w)$ describes the conditional probability (induced by $\pi^{(\ell)}_{S,n}$) of occurrence of an a after $w \in Q$.

Let $u = a_0 \cdots a_{\ell-1}$ be in Σ^ℓ.

$$\frac{B_n(\lambda)(a_0)\cdots B_n(u[0..\ell-2])(a_{\ell-1})}{\alpha(a_0)\cdots\alpha(a_{\ell-1})} = \frac{\pi^{(\ell)}_{S,n}(u)}{\alpha^{(\ell)}(u)}.$$

Then for $z \in \Sigma^*$ with $z \sqsubseteq S$ and $|z| = \ell n$, we have

$$\begin{aligned} \alpha^{|z|}(z)^{1-s} d_{G,\alpha}(z) &= \alpha^{|z|}(z)^t d_{G,\alpha}(z) \\ &= \alpha^{|z|}(z)^t \prod_{u\in\Sigma^\ell} \left(\frac{\pi^{(\ell)}_{S,n}(u)}{\alpha^{(\ell)}(u)} \right)^{n\pi^{(\ell)}_{S,n}(u)} \end{aligned}$$

Therefore,

$$\alpha^{|z|}(z)^t d_{G,\alpha}(z) \geq \frac{1}{2^c}^{|z|t} 2^{nD(\pi^{(\ell)}_{S,n}||\alpha^{(\ell)})}$$
$$\geq 2^{-c|z|t+c|z|t}$$

Since the number of states is fixed, this implies $\dim^{\alpha}_{\mathrm{FS}}(S) \leq 1 - \mathrm{Div}(S||\alpha)/c$. The proof of the other case is similar, where we use the fact that, for a.e. n,

$$D(\pi^{(\ell)}_{S,n}||\alpha^{(\ell)}) > \ell tc.$$ ◀

References

1 Valerii Nikolaevich Agafonov. Normal sequences and finite automata. In *Doklady Akademii Nauk*, volume 179 (2), pages 255–256, 1968.
2 Krishna B. Athreya, John M. Hitchcock, Jack H. Lutz, and Elvira Mayordomo. Effective strong dimension in algorithmic information and computational complexity. *SIAM Journal on Computing*, 37(3):671–705, 2007.
3 Verónica Becher and Pablo Ariel Heiber. Normal numbers and finite automata. *Theoretical Computer Science*, 477:109–116, 2013.
4 Verónica Becher, Pablo Ariel Heiber, and Theodore A. Slaman. A polynomial-time algorithm for computing absolutely normal numbers. *Information and Computation*, 232:1–9, 2013.
5 Patrick Billingsley. Hausdorff dimension in probability theory. *Illinois Journal of Mathematics*, 4(2):187–209, 1960.
6 Patrick Billingsley. *Probability and Measure*. Wiley-Interscience, 1995.
7 Émile Borel. Les probabilités dénombrables et leurs applications arithmétiques. *Rendiconti del Circolo Matematico di Palermo (1884-1940)*, 27(1):247–271, 1909.
8 Chris Bourke, John M. Hitchcock, and N. V. Vinodchandran. Entropy rates and finite-state dimension. *Theoretical Computer Science*, 349(3):392–406, 2005.
9 Yann Bugeaud. *Distribution Modulo One and Diophantine Approximation*. Cambridge University Press, 2012.
10 Helmut Cajar. *Billingsley Dimension in Probability Spaces*. Springer-Verlag, 1981.
11 Arthur H. Copeland and Paul Erdös. Note on normal numbers. *Bulletin of the American Mathematical Society*, 52(10):857–860, 1946.
12 Thomas R. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley & Sons, Inc., 2006.
13 Imre Csiszár and Paul C. Shields. *Information Theory and Statistics: A Tutorial*, volume 1 (4). Now Publishers, Inc., 2004.
14 Jack J. Dai, James I. Lathrop, Jack H. Lutz, and Elvira Mayordomo. Finite-state dimension. *Theoretical Computer Science*, 310(1-3):1–33, 2004.
15 Karma Dajani and Cor Kraaikamp. *Ergodic Theory of Numbers*. Cambridge University Press, 2002.
16 David Doty, Jack H. Lutz, and Satyadev Nandakumar. Finite-state dimension and real arithmetic. *Information and Computation*, 205(11):1640–1651, 2007.
17 Kenneth Falconer. *Fractal Geometry: Mathematical Foundations and Applications*. Wiley, 2014.
18 Meir Feder. Gambling using a finite state machine. *IEEE Transactions on Information Theory*, 37(5):1459–1465, 1991.
19 Xiaoyang Gu, Jack H. Lutz, and Philippe Moser. Dimensions of Copeland–Erdös sequences. *Information and Computation*, 205(9):1317–1333, 2007.
20 F. Hausdorff. Dimension und äußeres maß. *Mathematische Annalen*, 79:157–179, 1919.
21 Teturo Kamae and Benjamin Weiss. Normal numbers and selection rules. *Israel Journal of Mathematics*, 21(2-3):101–110, 1975.

22 Donald E. Knuth. *Art of Computer Programming, volume 2: Seminumerical Algorithms*. Addison-Wesley Professional, 2014.

23 Solomon Kullback and Richard A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

24 Jack H. Lutz. Dimension in complexity classes. *SIAM Journal on Computing*, 32(5):1236–1259, 2003.

25 Jack H. Lutz. The dimensions of individual strings and sequences. *Information and Computation*, 187(1):49–79, 2003. doi:10.1016/S0890-5401(03)00187-1.

26 Jack H. Lutz. A divergence formula for randomness and dimension. *Theoretical Computer Science*, 412(1-2):166–177, 2011.

27 Jack H. Lutz and Elvira Mayordomo. Computing absolutely normal numbers in nearly linear time. *arXiv preprint*, 2016. arXiv:1611.05911.

28 Per Martin-Löf. The definition of random sequences. *Information and Control*, 9(6):602–619, 1966.

29 Wolfgang Merkle and Jan Reimann. Selection functions that do not preserve normality. *Theory of Computing Systems*, 39(5):685–697, 2006.

30 Claus-Peter Schnorr. A unified approach to the definition of random sequences. *Mathematical Systems Theory*, 5(3):246–258, 1971.

31 Claus-Peter Schnorr and Hermann Stimm. Endliche Automaten und Zufallsfolgen. *Acta Informatica*, 1(4):345–359, 1972.

32 Claude Elwood Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

33 Alexander Shen. Automatic Kolmogorov complexity and normality revisited. In *International Symposium on Fundamentals of Computation Theory*, pages 418–430. Springer, 2017.

34 Dennis Sullivan. Entropy, Hausdorff measures old and new, and limit sets of geometrically finite Kleinian groups. *Acta Mathematica*, 153(1):259–277, 1984.

35 Claude Tricot. Two definitions of fractional dimension. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 91 (1), pages 57–74. Cambridge University Press, 1982.

36 Donald Dines Wall. *Normal numbers*. Ph.D. thesis, University of California, Berkeley, 1949.

A Appendix

The following is a proof for Lemma 2.2.

Proof. Let α, S, and ℓ be as given.

To see that (1) implies (2), assume (1). Then $\lim_{n\to\infty} \pi^{(\ell)}_{S,n} = \alpha^{(\ell)}$, so the continuity of KL-divergence tells us that

$$\mathrm{Div}_\ell(S||\alpha) = \lim_{n\to\infty} D(\pi^{(\ell)}_{S,n}||\alpha^{(\ell)}) = 0,$$

i.e., that (2) holds.

To see that (2) implies (3), assume (2). Then $\lim_{n\to\infty} D(\pi^{(\ell)}_{S,n}||\alpha^{(\ell)}) = \mathrm{Div}_\ell(S||\alpha) = 0$, whence the $\mathcal{L}_1$ bound in section 2.1 tells us that $\lim_{n\to\infty} \pi^{(\ell)}_{S,n} = \alpha^{(\ell)}$. For any α-ℓ-normal sequence $T \in \Sigma^\omega$, we have $\lim_{n\to\infty} \pi^{(\ell)}_{T,n} = \alpha^{(\ell)}$, whence the continuity of KL-divergence tells us that

$$\mathrm{Div}_\ell(S||T) = \lim_{n\to\infty} D(\pi^{(\ell)}_{S,n}||\pi^{(\ell)}_{T,n}) = D(\alpha^{(\ell)}||\alpha^{(\ell)}) = 0,$$

i.e., that (3) holds.

Since α-ℓ-normal sequences exist, it is trivial that (3) implies (4).

Finally, to see that (4) implies (1), assume that (4) holds. Then we have

$$\lim_{n\to\infty} D(\pi_{S,n}^{(\ell)}||\pi_{T,n}^{(\ell)}) = \mathrm{Div}_\ell(S||T) = 0,$$

whence the $\mathcal{L}_1$ bound in section 2.1 tells us that

$$\lim_{n\to\infty} \|\pi_{S,n}^{(\ell)} - \pi_{T,n}^{(\ell)}\|_1 = 0. \tag{A.1}$$

We also have

$$\lim_{n\to\infty} \pi_{T,n}^{(\ell)} = \alpha^{(\ell)},$$

whence

$$\lim_{n\to\infty} \|\pi_{T,n}^{(\ell)} - \alpha^{(\ell)}\|_1 = 0. \tag{A.2}$$

By (A.1), (A.2), and the triangle inequality for the $\mathcal{L}_1$-norm, we have

$$\lim_{n\to\infty} \|\pi_{S,n}^{(\ell)} - \alpha^{(\ell)}\|_1 = 0,$$

whence

$$\lim_{n\to\infty} \pi_{S,n}^{(\ell)} = \alpha^{(\ell)},$$

i.e., (1) holds. ◀

Lemma 2.2 immediately implies the following.

▶ **Theorem 2.3** (divergence characterization of normality). *For all $\alpha \in \Delta(\Sigma)$ and $S \in \Sigma^\omega$, the following conditions are equivalent.*

(1) *S is α-normal.*
(2) $\mathrm{Div}(S||\alpha) = 0$.
(3) *For every α-normal sequence $T \in \Sigma^\omega$, $\mathrm{Div}(S||T) = 0$.*
(4) *There exists an α-normal sequence $T \in \Sigma^\omega$ such that $\mathrm{Div}(S||T) = 0$.*

Perfect Resolution of Conflict-Free Colouring of Interval Hypergraphs

S. M. Dhannya
Dept. of Computer Science and Engineering, IIT Madras, Chennai, India
dhannya@cse.iitm.ac.in

N. S. Narayanaswamy
Dept. of Computer Science and Engineering, IIT Madras, Chennai, India
swamy@cse.iitm.ac.in

Abstract

Given a hypergraph H, the conflict-free colouring problem is to colour vertices of H using minimum colours so that in every hyperedge e of H, there is a vertex whose colour is different from that of all other vertices in e. Our results are on a variant of the conflict-free colouring problem considered by Cheilaris et al.[4], known as the 1-Strong Conflict-Free (1-SCF) colouring problem, for which they presented a polynomial time 2-approximation algorithm for interval hypergraphs. We show that an optimum 1-SCF colouring for interval hypergraphs can be computed in polynomial time. Our results are obtained by considering a different view of conflict-free colouring which we believe could be useful in general. For interval hypergraphs, this different view brings a connection to the theory of perfect graphs which is useful in coming up with an LP formulation to select the vertices that could be coloured to obtain an optimum conflict-free colouring. The perfect graph connection again plays a crucial role in finding a minimum colouring for the vertices selected by the LP formulation.

2012 ACM Subject Classification Mathematics of computing → Hypergraphs; Mathematics of computing → Graph algorithms; Theory of computation → Design and analysis of algorithms

Keywords and phrases Conflict-free Colouring, Interval Hypergraphs

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.52

Acknowledgements We thank the anonymous reviewers for their comments which have greatly improved the quality of the paper.

1 Introduction

A vertex colouring function $C : \mathcal{V} \to \{0, 1, 2, \ldots, k\}$ of a hypergraph $H = (\mathcal{V}, \mathcal{E})$ using k non-zero colours is a 1-*SCF colouring* of H, if for every hyperedge $e \in \mathcal{E}$ there exists a non-zero colour $j \in \{1, 2, \cdots, k\}$ such that $|e \cap C^{-1}(j)| = 1$. This problem was first studied by Cheilaris et al. [4] and is a variant of a well-studied hypergraph colouring problem known as the *Conflict-Free colouring* problem. A Conflict-Free (CF, in short) colouring is a vertex colouring of a hypergraph that colours every vertex of the hypergraph such that every hyperedge e has at least one vertex whose colour is different from that of every other vertex in e. The CF colouring problem seeks to find a CF colouring using minimum number of colours. The number of colours used in any optimum CF colouring of a hypergraph H is called the *CF colouring number* of H. In the 1-SCF colouring, the algorithm is presented with an input in which all vertices are initially coloured with colour 0, and the goal is to modify the colour of some vertices to a non-zero colour such that the resulting colouring is a 1-SCF colouring. We observe that a 1-SCF colouring can be used to find a CF colouring of a given hypergraph by adding one interval of length 1 for each vertex in H (this simple transformation ensures that each vertex must be given a non-zero colour). We refer to the number of non-zero colours used in any optimum 1-SCF colouring of a hypergraph H as the 1-*SCF colouring number* of H. Observe that the CF colouring number of a hypergraph

H is at most one more than the 1-SCF colouring number of H. Our main result is an algorithm that solves the 1-SCF colouring problem optimally in polynomial time for interval hypergraphs, thus solving one part of an open problem posed by Cheilaris et al. [4].

▶ **Theorem 1.** *The* 1*-SCF colouring problem in interval hypergraphs can be optimally solved in polynomial time.*

As a corollary the 1-SCF colouring number of a given interval hypergraph can be computed in polynomial time.

Past Work in CF colouring. The survey due to Smorodinsky [17] presents a general framework for CF colouring that involves finding a proper colouring in every iteration and giving the largest colour class a new colour. Smorodinsky [17] showed that if for every induced sub-hypergraph $H' \subseteq H$, the chromatic number of H' is at most k, then $\chi_{cf}(H) \leq \log_{1+\frac{1}{k-1}} n = O(k \log n)$, where $n = |\mathcal{V}|$. Pach and Tardos [15] showed that if $|\mathcal{E}(H)| < \binom{s}{2}$ for some positive integer s, and Δ is the maximum degree of a vertex in H, then $\chi_{cf}(H) < s$ and $\chi_{cf}(H) \leq \Delta + 1$. The CF colouring problem has also been studied on different types of hypergraphs. Even et al. [8] have studied a number of hypergraphs induced by geometric regions on the plane including discs, axis-parallel rectangles, regular hexagons, and general congruent centrally symmetric convex regions in the plane. Let $\mathcal{D}$ be a set of n finite discs in $\mathbb{R}^2$. For a point $p \in \mathbb{R}^2$, define $r(p) = \{D \in \mathcal{D} : p \in D\}$. The hypergraph $(\mathcal{D}, \{r(p)\}_{p \in \mathbb{R}^2})$, denoted by $H(\mathcal{D})$, is called the hypergraph induced by $\mathcal{D}$. Smorodinsky showed that $\chi_{cf}(H(\mathcal{D})) \leq \log_{4/3} n$ [16, 17]. Similarly, if $\mathcal{R}$ is a set of n axis-parallel rectangles in the plane, then, $\chi_{cf}(H(\mathcal{R})) = O(\log^2 n)$. There have been many studies on hypergraphs induced by neighbourhoods in simple graphs. Given a simple graph $G = (V, E)$, the *open neighbourhood* (or simply neighbourhood) of a vertex $v \in V$ is defined as follows: $N(v) = \{u \in V \mid uv \in E\}$. The set $N(v) \cup v$ is known as the *closed neighbourhood* of v. Pach and Tardos [15] have shown that the vertices of a graph G with maximum degree Δ can be coloured with $O(\log^{2+\epsilon} \Delta)$ colours, so that the closed neighbourhood of every vertex in G is CF coloured. They also showed that if the minimum degree of vertices in G is $\Omega(\log \Delta)$, then the open neighbourhood can be CF coloured with at most $O(\log^2 \Delta)$ colours. Abel et al. [1] gave the following tight worst-case bound for neighbourhoods in planar graphs: three colours are sometimes necessary and always sufficient. Keller and Smorodinsky [14] studied conflict-colourings of intersection graphs of geometric objects. They showed that the intersection graph of n pseudo-discs in the plane admits a CF colouring with $O(\log n)$ colours, with respect to both closed and open neighbourhoods. Ashok et al. [2] studied an optimization variant of the CF colouring problem, namely Max-CFC. Given a hypergraph $H = (\mathcal{V}, \mathcal{E})$ and integer $r \geq 2$, the problem is to find a maximum-sized subfamily of hyperedges that can be CF coloured with r colours. They have given an exact algorithm running in $O(2^{n+m})$ time. The paper also studies the problem in the parametrized setting where one must find if there exists a subfamily of at least k hyperedges that can be CF coloured using r colours. They showed that the problem is FPT and gave an algorithm with running time $2^{O(k \log\log k + k \log r)}(n+m)^{O(1)}$.

CF colouring in Interval Hypergraphs. A hypergraph $H_n = ([n], \mathcal{I}_n)$, where $[n] = \{1, \ldots, n\}$ and $\mathcal{I}_n = \big\{\{i, i+1, \ldots, , j\} \mid i \leq j \text{ and } i, j \in [n]\big\}$ is known as a *complete interval hypergraph* [4]. A hypergraph such that the set of hyperedges is a family of intervals $\mathcal{I} \subseteq \mathcal{I}_n$ is known as an *interval hypergraph*. One can view the CF colouring problem in interval hypergraphs as modelling the frequency assignment problem in a chain of units discs

[4, 8]. In the case of hypergraphs induced by arbitrary unit discs in the plane, the problem is known to be NP-complete [13]. It was shown in [8] that a complete interval hypergraph can be CF coloured using $\Theta(\log n)$ colours. Chen et al.[5] presented results on an online variant of CF colouring problem in complete interval hypergraphs. In the online variant, points arrive online and a point has to be assigned a colour upon its arrival such that the resulting colouring is conflict-free with respect to all intervals. Chen et al.[5] gave a greedy algorithm that uses $\Omega(\sqrt{n})$ colours, a deterministic algorithm that uses $\Theta(\log^2 n)$ colours and a randomized algorithm that uses $O(\log n)$ colours. There have been some studies [4, 13] on CF colouring in interval hypergraphs (instead of complete interval hypergraphs), in which case a subset of intervals in $\mathcal{I}_n$ is given as part of the input.

1-SCF colouring in Interval Hypergraphs. Katz et al.[13] gave a polynomial-time approximation algorithm for 1-SCF colouring an interval hypergraph with approximation ratio 4. Cheilaris et al.[4] improved this result in their paper on *k-Strong CF colouring* (*k-SCF*) problem. The *k-SCF* problem seeks to find a vertex colouring of the hypergraph such that in every hyperedge e, there are at least $\min\{|e|, k\}$ vertices that are uniquely coloured. Cheilaris et al.[4] gave a polynomial-time approximation algorithm with approximation ratio 2 for $k = 1$ and $5 - \frac{2}{k}$ for $k \geq 2$. Further, they presented a quasi-polynomial time algorithm for the decision version of the *k-SCF* problem. This clearly ruled out the possibility of the decision version being NP-complete, unless NP-complete problems have quasi-polynomial time algorithms. The main result in this paper is a polynomial time optimum 1-SCF colouring algorithm for interval hypergraphs. We achieve this by observing a natural connection between the 1-SCF colouring problem and the problem of solving an exact hitting set problem with some constraints. We then formulate this exact hitting set problem as a linear program (LP) for interval hypergraphs, and show that this LP can be solved in polynomial time, and the LP solution can be rounded to an integer solution in polynomial time. For the rest of the paper, since we work entirely on 1-SCF colouring, we refer to non-zero colours as simply "colours".

Our Approach. We outline the approach towards proving our main result which is Theorem 1. The initial steps of our approach are simple observations regarding a different view of 1-SCF colouring which we present as Theorem 2 and Theorem 3. We present these observations as theorems because they bring a different perspective on 1-SCF colouring which turns out to be surprisingly useful for interval hypergraphs. However, in spite of the positive result with interval hypergraphs, we do not know of other hypergraphs for which our approach yields a better understanding of the 1-SCF colouring problem and the 1-SCF colouring number.
Theorem 2 brings into our perspective that 1-SCF colouring is actually the problem of computing a special type of colouring of the vertices of an exact hitting set. We observe that 1-SCF colouring of a hypergraph can be naturally seen as the proper colouring of a related simple graph which we call a *co-occurrence graph*. A co-occurrence graph is obtained from a 1-SCF colouring based on a function defined on $\mathcal{E}$, and we call this function a *representative function*. For a representative function t, the co-occurrence graph is denoted by Γ_t. We show that the search for the co-occurrence graph with the minimum chromatic number is equivalent to the search for the corresponding representative function.

▶ **Theorem 2.** *Let $H = (\mathcal{V}, \mathcal{E})$ be a hypergraph. Let $\chi_{cf}(H)$ be the number of colours used in any optimal 1-SCF colouring of H. Let $\chi_{min}(H)$ be the minimum chromatic number over all possible co-occurrence graphs of H. Then, $\chi_{cf}(H) = \chi_{min}(H)$.*

We then show that the search for an appropriate representative function is answered by finding an exact hitting set of some cliques in a graph $\hat{G}$ called the conflict graph of H. We identify two sets of cliques in $\hat{G}$, namely *hyperedge cliques* denoted by $\mathcal{Q}_1$ and *colour cliques* denoted by $\mathcal{Q}_2$. These cliques are defined in Definition 7. The conflict graph is designed in such a way that there exists an exact hitting set S of cliques in $\mathcal{Q}_1$ that hits every maximal clique in $\mathcal{Q}_2$ at most q times if and only if there exists a 1-SCF colouring of H with q colours (Lemma 9). We establish the relation between the conflict graph and a co-occurrence graph of a hypergraph in the theorem below. Theorem 3 gives us a framework for searching for an appropriate exact hitting set of the given hypergraph.

▶ **Theorem 3.** *Let $H = (\mathcal{V}, \mathcal{E})$ be a hypergraph. Let t be a representative function of H and let $\hat{G}$ be the conflict graph of H. Then, the set $S = \{(e, u) \mid (e, u) \in \hat{G}, t(e) = u\}$ is an exact hitting set of cliques in $\mathcal{Q}_1$, $\omega(\hat{G}[S]) \leq \omega(\Gamma_t)$, and $\chi(\hat{G}[S]) \leq \chi(\Gamma_t)$.*

We next state the following important structural properties of co-occurrence graphs and conflict graphs specific to interval hypergraphs in Section 3.

▶ **Theorem 4.** *Each co-occurrence graph of an interval hypergraph is perfect.*

▶ **Theorem 5.** *Let $H = (\mathcal{V}, \mathcal{I})$ be an interval hypergraph. Then the conflict graph $\hat{G}(H)$ is perfect.*

In order to find an optimal co-occurrence graph for interval hypergraphs, we formulate a Linear Program (LP) for a constrained exact hitting set of a set of cliques of $\hat{G}$ in Section 4. We show that the LP can be solved in polynomial time using the ellipsoid method. The ellipsoid method uses a separation oracle that we design specifically for interval hypergraphs, and this oracle crucially relies on the fact that the conflict graph is perfect. Further, an optimum fractional solution obtained from the LP is rounded to give an integer feasible solution. This integer feasible solution naturally gives a representative function for the given interval hypergraph, and we show that the corresponding co-occurrence graph has the minimum chromatic number over all co-occurrence graphs of the given interval hypergraph. The minimum colouring of the co-occurrence graph can also be computed in polynomial time using known algorithms for minimum colouring of perfect graphs. We believe that this technique can be used to design an optimal polynomial time algorithm for the k-SCF problem of Cheilaris et al.[4]. The perfectness of co-occurrence graphs and conflict graphs in the 1-SCF colouring are not dependent on k and hence we conjecture that the same approach will work for the k-SCF problem with minimal tuning to the rounding procedure for the LP. Finally, we have not been able to prove any other computationally useful structure on the co-occurrence graphs and conflict graphs of interval hypergraphs. In particular, we know that both these graphs can have induced cycles of length 4, and thus we cannot use techniques from chordal graph colouring or interval graph hitting set (which uses the consecutive ones property of the clique-vertex incidence matrix) algorithms.

1.1 Preliminaries

In an interval $I = \{i, i+1, \ldots, j\}$, i and j are the *left* and *right endpoints* of I respectively, denoted by $l(I)$ and $r(I)$, respectively. Since an interval is a finite set of consecutive integers, it follows that $|I|$ is well-defined. Throughout the paper, we assume that the hypergraph H has n vertices and m hyperedges.

If vertex $v \in e$ has been assigned a colour c that is different from the colour of all other vertices in e, then we say that e *is* 1-*SCF coloured* by vertex v and by colour c. Note that in our convention, we use colour 0 to indicate that a vertex with colour 0 does not 1-SCF colour any hyperedge.

For a set S of vertices in a simple graph G, $G[S]$ denotes the induced subgraph of G on S. Perfect graphs [9] are very well-studied and many hard problems are tractable on perfect graphs. We use four well known properties of perfect graphs.

P1 Let $G = (V, E)$ be a perfect graph. For a given subset $V' \subseteq V$, let $G[V'] = (V', E_{V'})$ be the subgraph induced by V', where $E_{V'} = \{uv \in E \mid u, v \in V'\}$. Then, it is known from [9] that $\omega(G[V']) = \chi(G[V'])$, where $\omega(G[V'])$ and $\chi(G[V'])$ are, respectively, the clique number and the chromatic number of $G[V']$. Recall that the clique number of a simple graph G is the size of its largest clique and the chromatic number of G is the number of colours needed in an optimal proper colouring of G.

P2 A *Berge graph* is a simple graph that has neither an *odd hole* nor an *odd anti-hole* as an induced subgraph [3, 6, 7, 9]. An odd hole is an induced cycle of odd length that has at least 5 vertices and an odd anti-hole is the complement of an odd hole. It is known from Theorem 1.2 in [7] that a graph is perfect if and only if it is Berge.

P3 The chromatic number of a perfect graph can be found in polynomial time [11].

P4 The maximum weighted clique problem can be solved in polynomial time in perfect graphs [10],[11].

We use linear programming and combinatorial optimization concepts from [12] and [10]. All other definitions and notations used in this paper are from West [19] and Smorodinsky [17].

2 Co-occurrence Graphs, Conflict Graphs and 1-SCF colouring

We define two types of simple graphs associated with a hypergraph $H = (\mathcal{V}, \mathcal{E})$. We also establish the relationship between a proper colouring of these graphs and a 1-SCF colouring of H. For a 1-SCF colouring function C defined on $\mathcal{V}$, let $t : \mathcal{E} \to \mathcal{V}$ be a function such that for each $e \in \mathcal{E}$, $t(e)$ is that vertex v in e such that the colour given to v by C is not given to any other vertex in e. We refer to t as a representative function obtained from the colouring C. Further, each function $t : \mathcal{E} \to \mathcal{V}$ such that for each edge e, $t(e) \in e$ is referred to as a representative function of H. We now define the *Co-occurrence Graph*, $\Gamma_t(H)$, given a representative function t of H. Let $R \subseteq \mathcal{V}$ denote the image of $\mathcal{E}$ under the function t. The vertex set of the co-occurrence graph $\Gamma_t(H)$ is R, and for $u, v \in R$, uv is an edge in $\Gamma_t(H)$ if and only if for some $e \in \mathcal{E}$, $u \in e$ and $v \in e$ and $t(e)$ is either u or v. Further, given a representative function t of H, a proper colouring of the graph $\Gamma_t(H)$ defines a 1-SCF colouring of H for which t is a representative function obtained from the 1-SCF colouring. Note that in this 1-SCF colouring, the vertices which are not present in $\Gamma_t(H)$ get the 0 colour. Wherever H is implied, we use Γ_t to denote $\Gamma_t(H)$. Define $\chi_{min}(H) = \min_t \chi(\Gamma_t)$ where $\chi(\Gamma_t)$ is the chromatic number of the co-occurrence graph Γ_t and the minimum is taken over all representative functions t of the hypergraph H.

We prove the equivalence stated in Theorem 2.

Proof of Theorem 2. Let t be a representative function such that $\chi(\Gamma_t) = \chi_{min}(H)$. We extend a proper colouring C of Γ_t to a vertex colouring function C' of $\mathcal{V}(H)$ by assigning the colour 0 to those vertices in $\mathcal{V}(H) \setminus R$. C' is a 1-SCF colouring of H since for each $e \in \mathcal{E}$, the colour assigned to the vertex $t(e)$ by C' is different from the colour assigned to every other vertex in e. The reason for this is as follows: let $v \in e$ be a vertex different from $t(e)$. If $C'(v) = 0$, then definitely its colour is different from $C'(t(e))$. On the other hand, if $C'(v) \neq 0$, then it implies that there is an e' such that $v = t(e')$. Consequently, $v \in V(\Gamma_t)$, and since $v \in e$, $\{v, t(e)\}$ is an edge in Γ_t by the definition of Γ_t. Further, since C' is obtained from a proper colouring C of Γ_t it follows that $C'(v)$ is different from $C'(t(e))$.

Thus $\chi_{cf} \leq \chi_{min}(H)$. We prove that $\chi_{min}(H) \leq \chi_{cf}(H)$ as follows: since a minimum 1-SCF colouring of H gives a representative function t as defined above, it follows that $\chi_{cf}(H) \geq \chi(\Gamma_t) \geq \chi_{min}(H)$. Therefore, it follows that $\chi_{cf}(H) = \chi_{min}(H)$. ◀

As a consequence of Theorem 2, to find a 1-SCF colouring with the least number of colours, we need to find a co-occurrence graph for which the chromatic number is the least over all co-occurrence graphs. This entails first computing the representative function corresponding to the co-occurrence graph which has the minimum chromatic number, and then computing a minimum colouring of the co-occurrence graph. We pose the problem of finding the candidate representative function as a hitting set problem on a graph called the *Conflict Graph* associated with a hypergraph. Given a hypergraph $H = (\mathcal{V}, \mathcal{E})$, we use $\hat{G}(H) = (V, E)$ to denote the conflict graph of H. Wherever H is implied, we use $\hat{G}$ to denote $\hat{G}(H)$. The elements of $V(\hat{G})$ and $\mathcal{V}(H)$ are referred to as *nodes* and *vertices*, respectively. The node set of $\hat{G}(H)$ is $V = \big\{(e, v) \mid e \in \mathcal{E}, v \in e\big\}$. In a node (e, v), we refer to e as the hyperedge coordinate and v as the vertex coordinate. Conceptually, a node (e, v) in $\hat{G}$ represents the logical proposition that hyperedge e is 1-SCF coloured by vertex $v \in e$. $E(\hat{G})$ is defined such that each edge encodes a constraint to be satisfied by any 1-SCF colouring of H. The edge set of $\hat{G}$ is $E = E_{edge} \cup E_{colour}$, where E_{edge} and E_{colour} are defined as follows:

1. $E_{edge} = \big\{\big((e, v), (e, u)\big) \mid \{v, u\} \subseteq e, u \neq v\big\}$. For each hyperedge e in H, the nodes in $\hat{G}$ with e as the hyperedge coordinate form a clique.
2. $E_{colour} = \big\{\big((e, v), (g, u)\big) \mid \{v, u\} \subseteq e \text{ or } \{v, u\} \subseteq g, u \neq v, e \neq g\big\}$. These edges encode the proposition that if two vertices co-occur in a hyperedge, they must get two different colours, irrespective of other hyperedges to which any of these vertex may belong to.

The following structural property of a conflict graph is crucial in the proof of Lemma 11.

▶ **Observation 6.** *Given a hypergraph $H = (\mathcal{V}, \mathcal{E})$, for each vertex $v \in \mathcal{V}$, the set of nodes $\{(e, v) \mid e \in \mathcal{E}, v \in e\}$ in $\hat{G}$ forms an independent set.*

We identify the following sets of "useful" cliques in a conflict graph.

▶ **Definition 7** (Hyperedge Cliques and Colour Cliques)**.** *Hyperedge Clique is a clique in a conflict graph formed by nodes having the same hyperedge coordinate. The set of hyperedge cliques in a conflict graph is denoted by $\mathcal{Q}_1$. Colour Clique is a maximal clique in a conflict graph that has at least one edge from E_{colour}. The set of colour cliques in a conflict graph is denoted by $\mathcal{Q}_2$.*

We now prove Theorem 3 that states the relationship between the clique sizes of the co-occurrence graph and the conflict graph.

Proof of Theorem 3. By our premise, t is a representative function and hence the set S, obtained from t as defined above, hits every hyperedge clique exactly once. Therefore, it follows from definition of an exact hitting set that S is indeed an exact hitting set of the set of hyperedge cliques. Now, we show that $\omega(\Gamma_t) \geq \omega(\hat{G}[S])$. Let $\big\{(e, u), (f, v)\big\}$ be an edge in $\hat{G}[S]$ such that $e \neq f$. By definition of set S, $t(e) = u$ and $t(f) = v$. Since $e \neq f$, the edge $\big\{(e, u), (f, v)\big\}$ belongs to E_{colour} of $\hat{G}$. Hence u and v are both present together in either e or f. Without loss of generality, let $u, v \in e$. Since $(e, u) \in \hat{G}[S]$, we have $t(e) = u$ by construction of S. Hence, $\{u, v\}$ is an edge in Γ_t.
Therefore, for every edge $\big\{(e, u), (f, v)\big\}$ in $\hat{G}[S]$, there exists an edge $\{u, v\}$ in Γ_t. It follows that for every clique in $\hat{G}[S]$, there exists a clique of same size in Γ_t. Hence, $\omega(\Gamma_t) \geq \omega(\hat{G}[S])$.
Given a proper colouring of Γ_t, let the colour given to the node (e, u) be the colour given to vertex u in the proper colouring of Γ_t. From Observation 6, we know that there are no edges

between two nodes with the same vertex coordinate. Further, for each edge $\{(e,u),(f,v)\}$ in $\hat{G}[S]$, the edge $\{u,v\}$ is in Γ_t, and hence it follows that the colouring of $\hat{G}[S]$ is a proper colouring. Thus $\chi(\Gamma_t) \geq \chi(\hat{G}[S])$. ◀

As a consequence of Theorem 3, it follows that a representative function for H could be computed by finding an exact hitting set S of hyperedge cliques in $\hat{G}$ such that the chromatic number of $\hat{G}[S]$ is the minimum over all such exact hitting sets. We apply this approach to find an optimum 1-SCF colouring in polynomial time for interval hypergraphs by showing that such a hitting set can be computed in polynomial time for intervals. We show that this hitting set indeed gives the representative function whose co-occurrence graph has the minimum chromatic number. Further for interval hypergraphs, we show that the minimum vertex colouring of the co-occurrence graph can be computed efficiently. These results rely on the results in Section 3 which show that the co-occurrence graphs and the conflict graph of interval hypergraphs are perfect graphs.

3 Perfectness of Co-occurrence graphs and Conflict graphs of Interval Hypergraphs

We now prove two perfectness results when the given hypergraph is an interval hypergraph. The perfectness of co-occurrence graphs proved in Theorem 4 enables us to find a proper colouring of Γ_t in polynomial time. The perfectness of conflict graphs proved in Theorem 5 is used to prove Lemma 10. Lemma 10 is crucial in finding a hitting set of hyperedge cliques in $\hat{G}$. We first prove Theorem 4.

Proof of Theorem 4. We use property P2 of perfect graphs given in Section 1.1 to prove this result. By property P2, we know that an induced odd cycle and its complement are forbidden induced subgraphs for perfect graphs. Given an interval hypergraph H, let t be a representative function and let Γ_t be the resulting co-occurrence graph. We first show that Γ_t does not have an induced cycle of length at least 5. Note that we prove a stronger statement than required by property P2 of perfect graphs which states that there are no induced odd cycles of length at least 5. Our proof is by contradiction. Assume that $F = \{p_1, p_2 \dots p_r\}$ is an induced C_r-cycle for $r \geq 5$. Let the sequence of nodes in F be $p_1, p_2 \dots p_r, p_1$. Let p_i be the rightmost point of F on the line. In what follows, the arithmetic among the indices of p is *mod* r. Observe that, due to cyclicity of C_r, if $i = 1$, then $i - 1 = r$. Similarly, if $i = r$, then $i + 1 = 1$ and $i + 2 = 2$. Without loss of generality, let us assume that $p_{i-1} < p_{i+1}$, which are the two neighbours of p_i in F. Therefore, $p_{i-1} < p_{i+1} < p_i$. Since edge $\{p_{i-1}, p_i\}$ is in F, it follows that there exists an interval I for which $t(I) = p_{i-1}$ or $t(I) = p_i$. We claim that $t(I)$ is p_i: if $t(I)$ is p_{i-1}, then $\{p_{i-1}, p_{i+1}\}$ is an edge in Γ_t by definition. Therefore, $\{p_{i-1}, p_{i+1}\}$ is a chord in F, a contradiction to the fact that F is an induced cycle. Therefore, $t(I) = p_i$. Further, we claim that the point $p_{i+2} < p_{i-1}$: if $p_{i+2} > p_{i-1}$, then p_{i+2} belongs to the interval I and by the definition of the edges in Γ_t, $\{p_i, p_{i+2}\}$ is an edge in Γ_t. Therefore, $\{p_i, p_{i+2}\}$ is a chord in F. This contradicts the fact that F is an induced cycle. Therefore, $p_{i+2} < p_{i-1}$. At this point in the proof we have concluded that $p_{i+2} < p_{i-1} < p_{i+1} < p_i$ and $t(I) = p_i$. Since $\{p_{i+1}, p_{i+2}\}$ is an edge in F, it follows that there exists an interval J such that both p_{i+1} and p_{i+2} belong to J and $t(J) = p_{i+1}$ or $t(J) = p_{i+2}$, and therefore $p_{i-1} \in J$. Since F is an induced cycle of length at least 5, $\{p_{i-1}, t(J)\}$ is an edge in Γ_t by definition. Therefore, $\{p_{i-1}, t(J)\}$ is a chord in either case, that is when $t(J) = p_{i+1}$ or $t(J) = p_{i+2}$. This contradicts the assumption that F is an induced cycle of length at least 5. Thus, Γ_t cannot have an induced cycle of size at least 5. Next, we show that Γ_t does not contain the

complement of an induced cycle of length ≥ 5, ($\overline{C_r}$, $r \geq 5$) as an induced subgraph. Again, our proof is by contradiction. Assume that F is an induced $\overline{C_r}$, $r \geq 5$ in Γ_t. Let $q_1, q_2, \ldots, q_r$ be the nodes of F. Also, let $q_1 < q_2 < \ldots < q_r$ be the left to right ordering of points on the line corresponding to vertices of F. Since $deg(q_i) = r - 3$ for all q_i in F, it follows that no interval I, such that $t(I) \in F$, contains more than $r - 2$ vertices from F. Otherwise, if there exists an interval I such that $t(I) \in F$ contains more than $r - 2$ vertices from F, then $deg(t(I)) \geq r - 2$ in F which is a contradiction. Therefore, there does not exist any interval that contains both q_1 and q_r. Similarly, there does not exist any interval that contains both q_1 and q_{r-1} and any interval that contains both q_2 and q_r. Since $deg(q_1) = r - 3$, it follows that q_1 must be adjacent to all vertices in $\{q_2, q_3, \ldots, q_{r-2}\}$. Similarly, q_r must be adjacent to all vertices in $\{q_3, q_4, \ldots, q_{r-1}\}$. Next, we consider the degrees of vertices q_2 and q_{r-1} in F. Since they are in F, q_2 is adjacent to q_1 and q_{r-1} is adjacent to q_r. Now, q_2 must be adjacent to $r - 4$ more vertices. We show that q_2 is not adjacent to q_{r-1}. Suppose not, that is, if q_2 is adjacent to q_{r-1}, then there exists an interval I that contains both q_2 and q_{r-1} and $t(I) = q_2$ or $t(I) = q_{r-1}$. Then $t(I)$ is adjacent to all points in the set $\{\{q_2, q_3, \ldots, q_{r-1}\} \setminus t(I)\}$. Thus, by considering the one additional edge incident on $t(I)$ depending on whether $t(I) = q_2$ or q_{r-1}, it follows that $deg(t(I)) \geq r - 2$, a contradiction to the fact that the degree of each vertex inside F is $r - 3$. Therefore, the edge $\{q_2, q_{r-1}\}$ does not exist in F. It follows that in $\overline{F}$, which we know is an induced cycle of length at least 5, there is an induced cycle $q_1, q_{r-1}, q_2, q_r, q_1$ of length 4. This contradicts the structure of an induced cycle of length at least 5. Hence, we conclude that Γ_t does not have an induced cycle of length 5 or more or its complement. Therefore Γ_t is a perfect graph. ◀

We now prove that for an interval hypergraph H, $\hat{G}(H)$ is perfect. In this proof, $\mu(H)$ denotes the number of vertices in $\hat{G}$. Note that $\mu(H) = \sum_{I \in \mathcal{I}} |I|$.

Proof of Theorem 5. By property P2 of perfect graphs given in Section 1.1, we know that for each $p > 1$, induced odd cycle C_{2p+1} and its complement denoted by $\overline{C_{2p+1}}$ are forbidden induced subgraphs for perfect graphs. We now show that for an interval hypergraph, the graph $\hat{G}$ is perfect. Our proof is by starting with the hypothesis that the claim is false and deriving a contradiction. Let $H = (\mathcal{V}, \mathcal{J})$ be an interval hypergraph for which $\hat{G}$ is not perfect, and among all such interval hypergraphs, H minimizes $\mu(H)$. Since $\hat{G}$ is not perfect, let us consider a minimal induced subgraph of $\hat{G}$, denoted by, say F for which $\omega(F) \neq \chi(F)$. We claim that for every interval $I \in \mathcal{J}$ such that $|I| > 1$, both the nodes $(I, l(I))$ and $(I, r(I))$ belong to F. The proof of this claim is by contradiction to the fact that H is an interval hypergraph that minimizes $\mu(H)$ and for which $\hat{G}$ is not perfect. Let I be an interval in $\mathcal{J}$ such that $|I| > 1$ and the node $(I, r(I)) \notin V(F)$. Consider the hypergraph $H' = (\mathcal{V}, \mathcal{J}')$ where $\mathcal{J}' = (\mathcal{J} \setminus I) \cup (I \setminus r(I))$. Let $\hat{G}'$ denote the conflict graph of H'. Observe that $V(\hat{G}') = V(\hat{G}) \setminus \{(I, r(I))\}$. Since $(I, r(I)) \notin V(F)$ and $(I, r(I)) \notin V(\hat{G}')$, it follows that F is an induced subgraph of $\hat{G}'$ also. Hence it follows that $\hat{G}'$ is imperfect. Further, $\mu(H') < \mu(H)$. This contradicts the hypothesis that H is the interval hypergraph with minimum $\mu(H)$ for which $\hat{G}$ is imperfect. Therefore, it follows that for each interval $I \in \mathcal{J}$, $(I, r(I))$ is a node in F. An identical argument shows that for each interval $I \in \mathcal{J}$, $(I, l(I))$ is also a node in F. Hence it follows that $\forall I \in \mathcal{J}$ such that $|I| > 1$, both the nodes $(I, l(I))$ and $(I, r(I))$ belong to F. We now consider two exhaustive cases to obtain a contradiction to the known structure of F which we know is either a C_{2p+1} or a $\overline{C_{2p+1}}$ for some $p > 1$.
Case 1- When F is an induced odd cycle $C_j, j \geq 5$: In the following proof we consider different cases, and in each case we conclude that three nodes of C_j form a K_3 in $\hat{G}$. This is a contradiction to the fact that induced cycles of length at least 4 do not have a K_3, and we refer to this as *a contradiction.*

We know that all the intervals I such that $|I| > 1$ have both the nodes $(I, l(I))$ and $(I, r(I))$ in F. Therefore, if (I'', q) is a node such that for some I', q is in interval I' and q is different from $r(I')$ and $l(I')$, then from the definition of E_{edge} and E_{colour}, we know that the 3 nodes $(I', l(I')), (I', r(I')), (I'', q)$ form a K_3, a contradiction. As a consequence of this observation, it also follows that for any two nodes (I_1, q_1) and (I_2, q_2) in C_j for which $|I_1| > 1$ and $|I_2| > 1$, $l(I_1)$ and $l(I_2)$ are different, and $r(I_1)$ and $r(I_2)$ are different. Therefore, for each node (I, q) in C_j, q is either $l(I)$ or $r(I)$ or $|I| = 1$, and q is the left end point (right end point) of at most one interval, and for each interval I', q is not an element of $I' \setminus \{l(I'), r(I')\}$ (we call this set as the strict interior of I').

From the conclusions above, the intervals of length more than 1 contribute an even number of distinct nodes to the cycle C_j. Since C_j is an induced odd cycle, it follows that in C_j there is at least one more node (I'', q) for which $|I''| = 1$. It follows that I'' contains only the point q. Let (I_1, q_1) and (I_2, q_2) be the two neighbours of (I'', q) in C_j. From Observation 6, it follows that q is different from q_1 and q_2. From the analysis above, it follows that q and q_1 are end points of I_1, and q and q_2 are end points of I_2. Again from the conclusions above, since $l(I_1)$ and $l(I_2)$ are different, and since $r(I_1)$ and $r(I_2)$ are different, without loss of generality, let us consider $q = l(I_1) = r(I_2)$ and $l(I_2) = q_2 < q < q_1 = r(I_1)$. Therefore, the three nodes $(I_1, r(I_1)), (I'', q), (I_2, l(I_2))$ form a path in F. We know that $(I_1, l(I_1))$ and $(I_2, r(I_2))$ are also vertices in C_j which is an induced (that is, chordless) cycle. Therefore, $(I_1, l(I_1)), (I_1, r(I_1)), (I'', q), (I_2, l(I_2)), (I_2, r(I_2))$ is a path in C_j. In other words, $(I_1, q), (I_1, q_1), (I'', q), (I_2, q_2), (I_2, q)$ is an induced path of length 5 in C_j, since (I_1, q) and (I_2, q) are not adjacent, by Observation 6. Since C_j is a cycle, it has at least one another node, say (I_3, q_3), which is the second neighbour of $(I_1, l(I_1))$ in C_j. We now show that all the points in I_3 are at least $r(I_1)$, and thus they are all larger than q. By the definition of $E(\hat{G})$ we know that $I_3 \cap I_1 \neq \emptyset$. Further, q_3 is an endpoint of I_3, and q_3 is not in the strict interior of I_1, and since q is in I_1 and I_2, and as per our convention each interval corresponds to a single hyperedge in H, it follows that q_3 is different from q. Consequently, it follows that $l(I_3) = r(I_1)$ and q_3 is $r(I_3)$. Note that this argument includes the case when $|I_3| = 1$, in which case $r(I_1) = q_3$. It follows that I_3 is an interval such that each point in I_3 is at least $r(I_1)$ which is larger than q.

Therefore from the conclusions made thus far, each node in C_j is one of two types: either the hyperedge coordinate is such that all the points in the corresponding interval are at most q or the hyperedge coordinate is such that all the points in the corresponding interval are more than q. In particular, I_2 is such that all the points are at most q and I_3 is such that all points are more than q. Since C_j is an induced cycle, it follows that there are two adjacent nodes (I_l, q_l) and (I_r, q_r) such that all points in I_l are at most q, and all points in I_r are more than q. In other words, I_l and I_r are two disjoint intervals, and we have concluded that (I_l, q_l) and (I_r, q_r) are adjacent. This is a contradiction to the definition of $E(\hat{G}) = E_{colour} \cup E_{edge}$. This contradiction has been arrived at due to the assumption that there is a C_j of odd length at least 5. Hence, in this case our hypothesis that there is a minimal H for which $\hat{G}$ is not perfect is wrong.

Case 2- When F is the complement of an odd cycle, say $\overline{C_j}, j \geq 5$: Here we order the nodes in non-decreasing order of their vertex coordinate. Let the order be $(I_1, p_1), (I_2, p_2), \ldots, (I_j, p_j)$. Since each node is adjacent to exactly $j - 3$ vertices in F, it follows that (I_1, p_1) is not adjacent to (I_{j-1}, p_{j-1}) and (I_j, p_j) in $\hat{G}$. Similarly, (I_j, p_j) is not adjacent to (I_1, p_1) and (I_2, p_2). The reason is that if there is an edge between (I_1, p_1) and (I_{j-1}, p_{j-1}) then one of the two nodes is adjacent to all the nodes whose vertex coordinates are between p_1 and p_{j-1}. Such a node will have degree $j - 2$ which contradicts the fact that all the nodes in F have

degree $j-3$. Since the degree of each vertex is $j-3$, it follows that (I_1, p_1) is adjacent to all nodes from (I_2, p_2) to (I_{j-2}, p_{j-2}). Similarly, (I_j, p_j) is adjacent to all nodes from (I_3, p_3) to (I_{j-1}, p_{j-1}). Now, let us consider (I_2, p_2) and (I_{j-1}, p_{j-1}). If these two nodes are adjacent in F, then one of the two will have degree at least $j-2$. Such a case cannot happen. Therefore, the nodes $(I_1, p_1), (I_j, p_j), (I_2, p_2), (I_{j-1}, p_{j-1}), (I_1, p_1)$ forms an induced 4-cycle in $\overline{F}$. $\overline{F}$ is an induced cycle $C_j, j \geq 5$, and by definition does not contain an induced cycle of length 4. Thus our hypothesis that $\hat{G}$ contains F is false.
In either case the assumption of the existence of a minimal H for which $\hat{G}$ is not perfect leads to a contradiction to the known structure of perfect graphs. Hence, it follows that for an interval hypergraph $\hat{G}$ is perfect. ◀

4 Computing the Optimal Co-occurrence Graph of Interval Hypergraphs using Conflict Graphs

We present the algorithm to compute an optimal co-occurrence graph of a given interval hypergraph from a constrained exact hitting set of hyperedge cliques in the conflict graph.

4.1 Representative Function from a Hitting Set of Hyperedge Cliques

In Lemma 8 below, we strengthen Theorem 3 for interval hypergraphs by proving the equality of the chromatic number of the conflict graph and the co-occurrence graph. This plays a crucial role in formulating an LP relaxation to compute a constrained exact hitting set of hyperedge cliques. Let $H = (\mathcal{V}, \mathcal{I})$ be the given interval hypergraph and let $\hat{G}$ be its conflict graph. Let q_{min} be the smallest positive integer such that there exists an exact hitting set of hyperedge cliques that hits every maximal clique in the colour cliques at most q_{min} times. Let $S_{min} \subseteq V(\hat{G})$ be such an exact hitting set of hyperedge cliques that hits every maximal clique in the colour cliques at most q_{min} times. Clearly, $|S_{min}| = m$ since there are m hyperedge cliques, each corresponding to an interval. Define the representative function $t : \mathcal{I} \to \mathcal{V}$ as follows: $t(I) = u$ if $(I, u) \in S_{min}$. We show that the chromatic number of the co-occurrence graph Γ_t is upper bounded by q_{min}. Note that Theorem 3 proves the opposite inequality for all hypergraphs.

▶ **Lemma 8.** *Let $t : \mathcal{I} \to \mathcal{V}$ be the function as defined above. Then t is a representative function obtained from some 1-SCF colouring and $\chi(\Gamma_t) \leq q_{min}$.*

Proof. Since S_{min} is an exact hitting set of hyperedge cliques, it follows that for every hyperedge $I \in \mathcal{I}$, there exists exactly one node in S_{min} whose hyperedge coordinate is I. Hence, t is indeed a function. Since every interval is assigned a unique representative by t, it follows from the proof of Theorem 2 that any proper colouring of Γ_t is a 1-SCF colouring of H. Therefore, t is a representative function obtained from such a 1-SCF colouring of H. Now, we show that $\chi(\Gamma_t) \leq q_{min}$. In Theorem 4, we show that Γ_t is a perfect graph. It follows from property P1 of perfect graphs in Section 1.1 that the clique number ω and the chromatic number χ of every induced subgraph of Γ_t are equal. Hence it suffices to show that $\omega(\Gamma_t) \leq q_{min}$. To prove this, we show that $\omega(\Gamma_t)$ is at most the size of the maximum clique in $\hat{G}[S_{min}]$, and by assumption $\omega(\hat{G}[S_{min}]) \leq q_{min}$. In particular, for each clique in Γ_t we identify a clique of the same size in $\hat{G}[S_{min}]$. The proof is by induction on the size of a clique in Γ_t. The base case is for a clique of size 1 in Γ_t. Clearly, there is a clique of size 1 in $\hat{G}[S_{min}]$. By the induction hypothesis, corresponding to a clique comprising of $u_1, u_2, \ldots, u_{q-1}$ in Γ_t, there is a clique containing nodes $(I_1, u_1), (I_2, u_2), \ldots, (I_{q-1}, u_{q-1})$ in $\hat{G}[S_{min}]$. Now, we prove the claim when there are q vertices in a clique in Γ_t. Let $u_1, u_2, \ldots, u_q$ be the set

of vertices in the clique. Without loss of generality, assume that $u_1, u_2, \ldots, u_{q-1}, u_q$ is the left to right ordering of points on the line. Since $\{u_1, u_q\} \in E(\Gamma_t)$, there exists an interval, say I' such that u_1 and u_q belong to I' and $t(I') \in \{u_1, u_q\}$. We prove the claim for the case when $t(I')$ is u_1. It follows that the node $(I', u_1) \in S_{min}$. Since both u_1 and u_q belong to the interval I', it follows that $u_2, \ldots, u_{q-1}$ also belong to interval I'. By the induction hypothesis, for the $q-1$ sized clique $u_2, u_3, \ldots, u_q$ in Γ_t, there is a clique containing the nodes $(I_2, u_2), (I_3, u_3), \ldots, (I_q, u_q)$ in $\hat{G}[S_{min}]$. Therefore, it follows that (I', u_1) is adjacent to all the nodes $(I_2, u_2), (I_3, u_3), \ldots, (I_q, u_q)$ in $\hat{G}$. It follows that corresponding to the clique $u_1, \ldots, u_q$ in Γ_t, there is a clique $(I', u_1), (I_2, u_2), \ldots, (I_q, u_q)$ in $\hat{G}[S_{min}]$. In case $t(I')$ is u_q, an identical argument is applied to the clique $u_1, \ldots, u_{q-1}$ in Γ_t to prove the claim. Hence the lemma is proved. ◀

We next show that finding a 1-SCF colouring using minimum colours is equivalent to finding an exact hitting set of hyperedge cliques such that colour cliques are hit as few times as possible.

▶ **Lemma 9.** *There exists a set $S \subseteq V(\hat{G})$ such that for each $Q \in \mathcal{Q}_1$, $|S \cap Q| = 1$ and for each $Q' \in \mathcal{Q}_2$, $|S \cap Q'| \leq q$ if and only if there is a 1-SCF colouring of H with q colours.*

Proof. Let S be a subset of $V(\hat{G})$ such that for each $Q \in \mathcal{Q}_1$, $|S \cap Q| = 1$ and for each $Q' \in \mathcal{Q}_2$, $|S \cap Q'| \leq q$. Then by Lemma 8, there exists a representative function t such that $\chi(\Gamma_t) \leq q$. It further follows from Theorem 2 that a proper colouring of Γ_t is a 1-SCF colouring of H using $\chi(\Gamma_t) \leq q$ colours. This completes the forward direction of the claim. Next we prove the reverse direction. Let C be a 1-SCF colouring of H using q colours. Then by Theorem 2, C gives a representative function t with the property $q \geq \chi(\Gamma_t)$. Since co-occurrence graphs are perfect by Theorem 4, it follows that $q \geq \chi(\Gamma_t) = \omega(\Gamma_t)$. Define $S \triangleq \{(I, u) \mid I \in \mathcal{I}, t(I) = u\}$. By Theorem 3, the set S is an exact hitting set of cliques in $\mathcal{Q}_1$ and $\omega(\hat{G}[S]) \leq \omega(\Gamma_t) \leq q$. Thus we conclude that if there is a 1-SCF colouring of H using q colours, then there exists an exact hitting set of cliques in $\mathcal{Q}_1$ that intersects every maximal clique in $\mathcal{Q}_2$ at most q times. ◀

4.2 Linear Program for Exact Hitting Sets of Hyperedge Cliques

From Lemma 9, it is clear that an optimal 1-SCF colouring of an interval hypergraph H can be found by computing an exact hitting set of hyperedge cliques of $\hat{G}(H)$ such that each colour clique is hit as few times as possible. We present an LP formulation for this exact hitting set problem. In this LP, there is one variable corresponding to each node of $\hat{G}$ and integer valued variable q. Define $X \triangleq \{x_{I,u} \mid (I, u) \in \hat{G}\}$ to be the set of variables in the LP, where

$$x_{I,u} = \begin{cases} 1, & \text{if node } (I, u) \text{ hits hyperedge clique corresponding to } I \\ 0, & \text{otherwise} \end{cases}$$

LP Formulation. Find values to variables $\{x_{I,u} \mid u \in I, I \in \mathcal{I}\}$ subject to

$$\text{(P.1)} \quad \begin{aligned} &\sum_{u \in I} x_{I,u} = 1, \forall I \in \mathcal{I} && (1) \\ &\sum_{(I,u) \in Q} x_{I,u} \leq q, \text{ for each maximal clique } Q \text{ in } \mathcal{Q}_2. && (2) \\ &x_{I,u} \leq 1, q \geq 0 \end{aligned}$$

The LP relaxation has a set of equations, which are given in (P.1):(1) and a set of inequalities, which are given in (P.1):(2). Logically, an equation corresponds to choosing exactly one vertex per interval; that is, each equation corresponds to choosing exactly one node from one hyperedge clique. On the other hand, an inequality corresponds to a maximal clique in the set of colour cliques. Logically, the inequality means that we pick at most q nodes from every maximal clique in $\mathcal{Q}_2$. Together, any integral solution to the LP relaxation is an exact hitting set of hyperedge cliques such that each maximal clique in the set of colour cliques is hit at most q times. This LP relaxation is solved using the ellipsoid method which uses a polynomial time separation oracle that we next design. Let x denote an optimum solution to the LP relaxation. In Section 4.4 we present a rounding technique that converts the fractional solution x to a feasible integer solution for the LP in polynomial time.

4.3 Separation Oracle based LP Algorithm `SPAlg`

A separation oracle is a polyomial time algorithm that given a point in $\mathbb{R}^d$, where d is the number of variables in a linear program relaxation, either confirms that this point is a feasible solution, or produces a violated constraint (See Section 12.3.1 in[18]). A polynomial time separation oracle is used by the ellipsoid method to give a P-time algorithm for finding a feasible solution to the LP. In this section, we describe a polynomial time separation oracle `SPMaxWtClique` for our LP for a fixed positive integer value q. For each $(I, v) \in V(\hat{G})$, let $x_{I,v}$ be a real value assigned to the corresponding variable in the LP relaxation. Given this as an input, for a fixed positive integer value q, the separation oracle `SPMaxWtClique` considers the vertex-weighted graph $\hat{G}^w$ corresponding to $\hat{G}$, where the weight of node (I, v) is $x_{I,v}$ for all $(I, v) \in V(\hat{G})$. The oracle then computes the maximum weight clique of $\hat{G}^w$. If the weight of the maximum weight clique of $\hat{G}^w$ exceeds q, then it follows that there is some maximal clique Q' whose weight is more than q. This implies that the given point violates the inequality corresponding to Q', and this inequality is returned by the oracle as the violated inequality. If the weight of the maximum weight clique is at most q, then the oracle checks if all equalities hold. If any equality is violated, then we have found a violated constraint, and the oracle returns the appropriate inequality as the violated inequality. If all the constraints are satisfied, then the oracle reports that the given point is feasible. This completes the description of the separation oracle `SPMaxWtClique`. We show in Lemma 10 that `SPMaxWtClique` runs in polynomial time.

▶ **Lemma 10.** *For an input interval hypergraph and for each integer value $q \geq 0$ the separation oracle `SPMaxWtClique` runs in polynomial time.*

Proof. For an interval hypergraph we know that $\hat{G}$ is perfect by Theorem 5. From property P4 of perfect graphs listed in Section 1.1, it is known that the maximum weighted clique problem in perfect graphs can be solved in polynomial time. Thus we can find the maximum weighted clique in the graph vertex-weighted graph $\hat{G}^w$. Thus, finding an inequality in the LP corresponding to a maximal clique whose weight exceeds q can be done in polynomial time. Also, since there are only a polynomial number of hyperedge cliques, it follows that the check of whether there is a violated equation can also be done in polynomial time. It follows that `SPMaxWtClique` runs in polynomial time. ◀

Let $\mathcal{B}$ be an instance of the given LP. Algorithm `SPAlg` takes as inputs the LP instance $\mathcal{B}$. It uses the separation oracle `SPMaxWtClique` and iterates over integer values of q and outputs a solution x and a value q that satisfies the system of equations and inequalities. Otherwise, it reports that the system is infeasible. Let q_{min} be the smallest value of q for which Algorithm

SPAlg finds a feasible solution of the instance $\mathcal{B}$ and let B_{opt} be the solution returned by Algorithm SPAlg for the integer q_{min}. If B_{opt} is an integral solution, then we have an integer solution in polynomial time. If B_{opt} is not integral, then we present steps in Section 4.4 to round the fractional values in B_{opt} that results in a feasible integral solution for the value q_{min}.

▶ **Lemma 11.** *For an input interval hypergraph, the algorithm* SPAlg *runs in polynomial time.*

Proof. We have shown in Lemma 10 that the separation oracle in SPMaxWtClique runs in polynomial time. Since there is a polynomial time separation oracle, using the ellipsoid method, a feasible solution in the polytope of $\mathcal{B}$ can be found in polynomial time for each q. The number of values of q is at most the number of points in the interval hypergraph H. This is because, from Observation 6, for each vertex $u \in \mathcal{V}$ each clique in $\hat{G}$ can contain at most one node whose vertex coordinate is u. Hence the lemma is proved. ◀

4.4 Rounding the LP solution

RoundingFrac is a recursive function which takes as input a fractional feasible solution of the LP $\mathcal{B}$ for the value q_{min} on the intervals $\mathcal{I}$ and returns a feasible integer solution for $\mathcal{B}$ for the value q_{min}.
In every iteration of the *while* loop in Algorithm 1, at least one variable in X is rounded to

Algorithm 1 RoundingFrac$(B_{opt}, \mathcal{I}')$.

Output: Fractional solution B_{opt} rounded to integer solution B_{optI}

$i \leftarrow 0$;
$B_{opt}(0) \leftarrow B_{opt}$;
while $\exists x_{I,v} \in B_{opt}(i)$ *that does not belong to* $\{0,1\}$ **do**
  $i \leftarrow i+1$;
  $B_{opt}(i) \leftarrow B_{opt}(i-1)$;
  $I_i \leftarrow$ Longest Interval in $\mathcal{I}'$ with the smallest left endpoint ;
  $r \leftarrow r(I_i)$;
  $r-1 \leftarrow$ vertex to the immediate left of $r(I_i)$ on the line ;
  for *each interval* I' *that contains* r *and* $r-1$ **do**
    $x_{I',r-1} \leftarrow x_{I',r-1} + x_{I_i,r}$;
    $x_{I',r} \leftarrow x_{I',r} - x_{I_i,r}$;
    Modify entries in $B_{opt}(i)$ corresponding to the values changed above ;
  end
  $\mathcal{I}' \leftarrow \mathcal{I}' \setminus I_i$;
  $I_i \leftarrow I_i \setminus r$; ▷ *Remove right end point of* I_i;
  $\mathcal{I}' \leftarrow \mathcal{I}' \cup I_i$;
end
$B_{optI} \leftarrow B_{opt}(i)$;
return B_{optI} ;

an integer value. In iteration i, let I_i be the interval with the smallest left end point among all intervals of maximum length. Let $l(I_i)$ and $r(I_i)$ denote the left and right endpoints of interval I_i respectively. Since $r(I_i)$ is removed during iteration i, it follows that the total number of points (in all the intervals) in iteration $i+1$ is one less than the total number of

points in iteration i. Hence the conflict graph corresponding to intervals in iteration $i+1$ has strictly fewer number of nodes than the conflict graph corresponding to intervals in iteration i. In Lemma 13, we show that for every $i \geq 0$, the solution $B_{opt}(i)$ is feasible for the linear program $\mathcal{B}$ for the value q_{min}. We show in Lemma 12 that for some positive integer j, $B_{opt}(j)$ will be an all integer solution for $\mathcal{B}$, at which time algorithm exits.

▶ **Lemma 12.** *Let B_{opt} be a fractional feasible solution returned by* `SPAlg`$(\mathcal{B}, q_{min})$*. Then,* `RoundingFrac` *returns an integer solution for $\mathcal{B}$ on the input B_{opt} in a polynomial number of steps.*

Proof. From the description of `RoundingFrac`, in each iteration i, $x_{I_i,r(I_i)}$ becomes zero and the variable $x_{I_i,r(I_i)}$ does not become non-zero in any subsequent iteration. Then the number of variables whose value is not 0 or 1 reduces in each iteration. Further, the rounding is such that if a variable $x_{I,r}$ is reduced by a certain value then $x_{I,r-1}$ is increased by the exact same value. This ensures that after each iteration the equations in (P.1):(1) are all satisfied, and in particular they add up to 1. Therefore, eventually in each equation there will be a variable which is 1 and all others are 0. Further, it follows from Lemma 13 that the inequalities in (P.1):(2) also hold after each iteration. It follows that the resulting values are indeed a solution of the given LP and will be integral in at most $\mu(H)$ iterations, where $\mu(H)$ is the number of nodes in $\hat{G}$. ◀

Let B_{optI} be the integer solution returned by `RoundingFrac`. We show in Lemma 13 that B_{optI} is feasible for the instance $\mathcal{B}$ for the value q_{min}. In other words, the values to the variables in each inequality corresponding to the colour cliques add up to at most the same value as it was adding up to in B_{opt}. We show that the solution returned on a smaller instance after every iteration is feasible for $\mathcal{B}$ for the value q_{min}. In the proof of Lemma 13 below, we use r to denote $r(I_i)$, where I_i is the longest interval with the smallest left endpoint in iteration i. Similarly, denote the point to the immediate left of r on the number line by $r-1$. For every other interval I', denote its right endpoint and the point immediately to the left of the right endpoint by $r(I')$ and $r(I')-1$, respectively.

▶ **Lemma 13.** *Let B_{opt} be a fractional feasible solution returned by* `SPAlg`$(\mathcal{B}, q_{min})$*. The solution B_{optI} returned by* `RoundingFrac` *is a feasible solution for the LP instance $\mathcal{B}$ for the value q_{min}.*

Proof. The proof of correctness is by induction on the iteration number. We know that B_{opt} is feasible for $\mathcal{B}$ for the value q_{min}. Let us assume that for an integer $i \geq 0$ $B_{opt}(i-1)$ is feasible for $\mathcal{B}$ for the value q_{min}. We show that $B_{opt}(i)$ is also feasible for $\mathcal{B}$ for the value q_{min}. From the description of the `RoundingFrac`, during iteration i, the value which is subtracted from one variable from $x_{I,r}$ is added to the variable $x_{I,r-1}$. This fact is crucial in the analysis below. Hence all equations in (P.1):(1) are satisfied by $B_{opt}(i)$. Now, we show that the inequalities in (P.1):(2) corresponding to the maximal cliques are also satisfied by $B_{opt}(i)$. Let I' be an interval that contains the point $r-1$ such that $x_{I',r-1}$ has increased due to step 10 in Algorithm 1. By the choice of I' for which $x_{I',r-1}$ is increased, it follows that $x_{I',r}$ is reduced and thus I' contains the point r. It follows from the definition of the edge set E_{colour} that there is an edge between $(I', r-1)$ and (I_i, r) in $\hat{G}$.

Let Q be a maximal clique that contains the node $(I', r-1)$. By Observation 6, all nodes with the same vertex coordinate form an independent set. Hence Q does not contain any node of the form $(I'', r-1)$, where $I'' \neq I'$. Further, for any clique Q, there is at most one node whose value increases. If Q contains the node (I', r), then $x_{I',r}$ has reduced and hence the inequality corresponding to Q is satisfied under $B_{opt}(i)$. If Q does not contain the node

(I', r) we now show that it must contain a node whose vertex coordinate is r. To prove this, among all nodes in Q, consider two nodes - one for which the vertex coordinate is leftmost and another for which the vertex coordinate is the rightmost on the line. We denote the leftmost coordinate by λ and the rightmost coordinate by ρ. Let (J, λ) and (J', ρ) be two nodes in Q.

First, we show that $\lambda \geq l(I_i)$. The proof is by contradiction. Suppose $\lambda < l(I_i)$. Due to the edge between nodes (J, λ) and $(I', r-1)$ in Q, it is clear that either J or I' contains both λ and $r-1$. Without loss of generality, assume that J contains both λ and $r-1$. Since by our assumption $\lambda < l(I_i)$, it follows that J is at least as long as I_i and $l(J) < l(I_i)$. This is a contradiction to our choice of I_i being the longest interval with the smallest left endpoint. It follows that $\lambda \geq l(I_i)$. We show using the following cases that the inequality corresponding to Q is still feasible.

1. Case $\rho < r-1$. We show that this case is not possible. Since $(I', r-1)$ belongs to Q, and ρ is the rightmost vertex coordinate among all nodes in Q, it follows that $\rho \geq r-1$.
2. Case $\rho = r-1$. Since $\lambda \geq l(I_i)$ and $\rho = r-1$, it follows that all points from λ to ρ belong to I_i. Therefore, by the definition of the edges of $\hat{G}$, (I_i, r) is adjacent to all the nodes of Q. This contradicts the premise that Q is a maximal clique that does not contain (I_i, r). Therefore $\rho = r-1$ is not possible.
3. Case $\rho = r$. Since (J', ρ), which is the same as (J', r) belongs to Q, it follows that the inequality corresponding to Q is still feasible. Since the decrease in $x_{J',r}$ is exactly the same as the increase in $x_{I_i,r-1}$.
4. Case $\rho > r$. Observe that there is an edge between nodes (J, λ) and (J', ρ) since they are both in Q. It follows that either J or J' both contain λ and ρ. Without loss of generality, let J be this interval. Since J contains all the points on the line from λ to ρ, both included, it follows that the interval J contains both points r and $r-1$. Further, by the definition of the graph $\hat{G}$, it follows that (J, r) is adjacent to all the nodes in Q whose vertex coordinates which are different from r and lie between λ and ρ, both included. Also, since there can be at most one node in a maximal clique with any particular vertex coordinate, and since Q is a maximal clique, it follows that either (J, r) belongs to Q or that Q contains a node (J'', r) where $J \neq J''$. Since $x_{I_i,r}$ is reduced in iteration i, follows that $x_{J,r}$ and $x_{J'',r}$ are also reduced. Therefore, in the maximal clique Q the increase in $x_{I',r-1}$ is compensated by a decrease in $x_{J,r}$ or $x_{J'',r}$ whichever is present in Q. Therefore, the inequality corresponding to Q is satisfied in by $B_{opt}(i)$.

Therefore, in all the cases we have concluded the $B_{opt}(i)$ satsfies $\mathcal{B}$. This completes the proof by induction. ◀

We show in Theorem 1 that the 1-SCF colouring problem in interval hypergraphs can be solved in polynomial time. Finally, we prove the main result in this paper.

Proof of Theorem 1. By Lemma 11 the LP returns a feasible solution in polynomial time using the separation oracle `SPMaxWtClique`. By Lemmas 12 and 13, a feasible integer solution can be obtained from the fractional feasible solution in polynomial time. Further, the representative function t and thereof, the co-occurrence graph Γ_t can also be obtained in polynomial time. By Theorem 4, the co-occurrence graph Γ_t is perfect. Since a proper colouring of a perfect graph can be found in polynomial time, it follows from Theorem 2 that an optimal 1-SCF colouring of an interval hypergraph can be found in polynomial time. ◀

References

1 Zachary Abel, Victor Alvarez, Erik D. Demaine, Sándor P. Fekete, Aman Gour, Adam Hesterberg, Phillip Keldenich, and Christian Scheffer. Three colors suffice: Conflict-free coloring of planar graphs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, page 1951–1963, USA, 2017. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611974782.127.

2 Pradeesha Ashok, Aditi Dudeja, and Sudeshna Kolay. Exact and FPT algorithms for max-conflict free coloring in hypergraphs. In *Algorithms and Computation*, pages 271–282, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. doi:10.1007/978-3-662-48971-0_24.

3 C. Berge. *Graphs and Hypergraphs*. Elsevier Science Ltd., Oxford, UK, 1985.

4 Panagiotis Cheilaris, Luisa Gargano, Adele A. Rescigno, and Shakhar Smorodinsky. Strong conflict-free coloring for intervals. *Algorithmica*, 70(4):732–749, December 2014. doi:10.1007/s00453-014-9929-x.

5 Ke Chen, Amos Fiat, Haim Kaplan, Meital Levy, Jirí Matoušek, Elchanan Mossel, János Pach, Micha Sharir, Shakhar Smorodinsky, Uli Wagner, et al. Online conflict-free coloring for intervals. *SIAM Journal on Computing*, 36(5):1342–1359, 2006. doi:10.1137/S0097539704446682.

6 Maria Chudnovsky, Gérard Cornuéjols, Xinming Liu, Paul Seymour, and Kristina Vušković. Recognizing Berge graphs. *Combinatorica*, 25(2):143–186, March 2005. doi:10.1007/s00493-005-0012-8.

7 Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The strong perfect graph theorem. *Annals of Mathematics*, 164:51–229, 2006.

8 Guy Even, Zvi Lotker, Dana Ron, and Shakhar Smorodinsky. Conflict-free colorings of simple geometric regions with applications to frequency assignment in cellular networks. *SIAM Journal on Computing*, 33(1):94–136, 2003. doi:10.1137/S0097539702431840.

9 Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57)*. North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands, 2004.

10 Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, June 1981. doi:10.1007/BF02579273.

11 Martin Grötschel, László Lovász, and Alexander Schrijver. Polynomial algorithms for perfect graphs. In C. Berge and V. Chvátal, editors, *Topics on Perfect Graphs*, volume 88 of *North-Holland Mathematics Studies*, pages 325–356. North-Holland, 1984. doi:10.1016/S0304-0208(08)72943-8.

12 Martin Grötschel, Lászlo Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988.

13 Matthew J Katz, Nissan Lev-Tov, and Gila Morgenstern. Conflict-free coloring of points on a line with respect to a set of intervals. *Computational Geometry*, 45(9):508–514, 2012. doi:10.1016/j.comgeo.2012.01.013.

14 Chaya Keller and Shakhar Smorodinsky. Conflict-free coloring of intersection graphs of geometric objects. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 2397–2411. Society for Industrial and Applied Mathematics, 2018. doi:10.1137/1.9781611975031.154.

15 János Pach and Gábor Tardos. Conflict-free colourings of graphs and hypergraphs. *Combinatorics, Probability and Computing*, 18(05):819–834, 2009. doi:10.1017/S0963548309990290.

16 Shakhar Smorodinsky. On the chromatic number of geometric hypergraphs. *SIAM Journal on Discrete Mathematics*, 21:676–687, 2007. doi:10.1137/050642368.

17 Shakhar Smorodinsky. Conflict-free coloring and its applications. In *Geometry – Intuitive, Discrete, and Convex*, pages 331–389. Springer Berlin Heidelberg, 2013. doi:10.1007_978-3-642-41498-5_12.

18 Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin, Heidelberg, 2001.

19 Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2 edition, September 2000.

Constant-Time Dynamic $(\Delta + 1)$-Coloring

Monika Henzinger
University of Vienna, Faculty of Computer Science, Vienna, Austria
monika.henzinger@univie.ac.at

Pan Peng
Department of Computer Science, University of Sheffield, Sheffield, UK
p.peng@sheffield.ac.uk

Abstract

We give a fully dynamic (Las-Vegas style) algorithm with *constant expected amortized* time per update that maintains a proper $(\Delta + 1)$-vertex coloring of a graph with maximum degree at most Δ. This improves upon the previous $O(\log \Delta)$-time algorithm by Bhattacharya et al. (SODA 2018). Our algorithm uses an approach based on assigning random ranks to vertices and does not need to maintain a hierarchical graph decomposition. We show that our result does not only have optimal running time, but is also optimal in the sense that already deciding whether a Δ-coloring exists in a dynamically changing graph with maximum degree at most Δ takes $\Omega(\log n)$ time per operation.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Dynamic graph algorithms, Graph coloring, Random sampling

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.53

Related Version A full version of the paper is available at `https://arxiv.org/abs/1907.04745`.

Funding 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 no. 340506.

1 Introduction

A (fully) dynamic graph algorithm is a data structure that provides information about a graph property while the graph is being modified by *edge updates* such as edge insertions or deletions. When designing a dynamic graph algorithm the goal is to minimize the time per update or query operation. The lower bounds of Patrascu and Demaine [24] showed that in the cell-probe model many fundamental graph properties, such as asking whether the graph is connected, require $\Omega(\log n)$ time per operation, where n is the number of nodes in the graph. Their lower bound technique also gives logarithmic time lower bounds for further dynamic problems such as higher types of connectivity, planarity and bipartiteness testing, and minimum spanning forest, and it is an open research question for which other dynamic graph problems non-constant time lower bounds exist.

Furthermore, there are only very few graph problems for which it is known that no such lower bounds can exist. These are the following problems, which all have constant-time, and thus optimal, algorithms: maintaining (a) a maximal matching (randomized) [25], (b) a $(2 + \varepsilon)$-approximate vertex cover (deterministic) [7], and (c) a $(2k - 1)$-stretch spanner of size $O(n^{1+\frac{1}{k}} \log^2 n)$ for *constant* k (randomized) [3]. All these are *amortized* time bounds and each of these algorithms maintains a dynamically-changing sophisticated hierarchical graph decomposition.

In this paper we present a dynamic algorithm with constant update time for a new graph problem, expanding the above list. Additionally, our algorithm does not rely on a dynamically changing hierarchical graph decomposition, making it (but not its analysis) simpler. Our new result is a dynamic algorithm for the following problem: We call a dynamic

graph Δ-*bounded* if throughout the updates, the graph has maximum degree at most Δ. A *proper coloring* assigns to each vertex an integer value, called *color*, such that the endpoints of every edge have a different color. A $(\Delta + 1)$-vertex coloring is a proper coloring that uses only colors from the range $[1, \ldots, \Delta + 1]$. Note that a proper $(\Delta + 1)$-vertex coloring in a (static) graph with maximum degree at most Δ always exists and can be found in linear time by a simple greedy algorithm [27]. A *fully dynamic* graph algorithm is a data structure that maintains a graph $G = (V, E)$ while it is undergoing an arbitrary sequence of the following operations: 1) **Insert**(u, v): insert the edge (u, v) in G; 2) **Delete**(u, v): delete the edge (u, v) from G. In the dynamic $(\Delta + 1)$-vertex coloring problem, the fully dynamic graph algorithm maintains after each update operation a proper $(\Delta + 1)$-vertex coloring of the current graph in a Δ-bounded dynamic graph. When asked to perform a **Query**(u) operation, the algorithm returns the color of the given vertex u.

Maintaining a proper $(\Delta + 1)$-vertex coloring in a Δ-bounded dynamic graph can be done trivially in $O(\Delta)$ worst-case update time: the algorithm does nothing after an edge deletion or an edge insertion between two nodes of different colors; once an edge is inserted between two nodes of the same color it scans the whole neighborhood of one of the nodes and chooses an unused color. Recently Bhattacharya et al. [5] presented a randomized $(\Delta+1)$-vertex coloring algorithm with $O(\log \Delta)$ expected amortized update time and a deterministic algorithm that maintains a $(\Delta + o(\Delta))$-vertex coloring with $O(\text{poly} \log \Delta)$ amortized time. Their randomized algorithm works against the *oblivious adversary*: It is assumed that the sequence of update operations is generated by an adversary whose goal is to maximize the running time, but has to fix the sequence *before* the algorithm starts to run. This guarantees that the adversary is *oblivious* to the random choices of the algorithm. Note that if Δ is polynomial in n, their algorithm takes time $O(\log n)$. In this paper, we improve upon this result as follows.

▶ **Theorem 1.** *There exists a fully dynamic algorithm for maintaining a proper* $(\Delta+1)$*-vertex coloring for a* Δ*-bounded graph against an oblivious adversary with* $O(1)$ *expected amortized update time.*

Unlike the algorithm in [5] our algorithm does not need to maintain a hierarchical graph decomposition. Furthermore, apart from having optimal running time, our result is also optimal in the sense that deciding whether a proper coloring with only Δ colors exists in a dynamically changing graph (with maximum degree at most Δ) takes at least $\Omega(\log n)$ time per operation, as we show in Theorem 2. More precisely, we define the *dynamic* Δ*-colorability testing problem* as follows: Besides operations **Insert**(u, v) and **Delete**(u, v), there is a **Query**() operation that returns *yes* if the graph is Δ-colorable and *no* otherwise, where Δ is the maximum degree in the current graph. We show the following theorem.

▶ **Theorem 2.** *Any data structure for dynamic* Δ*-colorability testing, where* Δ *is the maximum degree in the graph, must perform* $\Omega(\log n)$ *cell probes, where each cell has size* $O(\log n)$*.*

Our Techniques. We first give a brief overview of the algorithm in [5] that maintains a proper $(\Delta + 1)$-vertex coloring for a dynamic graph with maximum degree at most Δ. Let χ be the current proper $\Delta + 1$-coloring. First note that after an edge deletion and after an edge insertion (u, v) that does not cause a conflict, i.e., if $\chi(u) \neq \chi(v)$, then the coloring remains unchanged. If a conflict occurs (i.e., $\chi(u) = \chi(v)$), then one needs to fix the coloring by recoloring one vertex from $\{u, v\}$, say u. Instead of scanning the whole neighborhood of u to find the color (called a *blank* color) that has not been used by any of its neighbors, the algorithm in [5] tries to *sample* a color from a set S that contains only blank colors and colors (called *unique* colors) that have been used by exactly one neighbor of u. Note that S

has size $\Omega(\Delta)$, which guarantees that a future conflict edge incident to u occurs with low probability (i.e., with probability $O(1/\Delta)$). On the other hand, if a unique color is chosen, one needs to recolor the corresponding vertex w (which is a neighbor of u), again, using a new color sampled from the set of blank and unique colors for w. This procedure might cause a cascade and even not terminate at all. The dynamic $(\Delta+1)$-vertex coloring algorithm of [5] resolves this problem by maintaining a hierarchical graph decomposition, and when recoloring a node it picks a color randomly out of all colors that are either (i) used by none of the neighbors or (ii) used by at most one of the neighbors *on a lower level* in the graph hierarchy. The resulting algorithm is then shown to have $O(\log \Delta)$ amortized update time for maintaining a proper coloring. However, maintaining such a hierarchical partition is not only complicated, but also inefficient, as it alone already takes $O(\log \Delta)$ amortized update time.

Now we describe our main ideas which lead to a constant-time dynamic coloring algorithm. We show that an approach based on *assigning random ranks* to vertices outperforms the graph-hierarchy based algorithm: During preprocessing each node v is assigned a random rank $r(v)$ from $[0,1]$ and a random color (assuming as usual that the initial graph is empty). Let L_v denote the set of neighbors of a node v with rank lower than $r(v)$ and for any set S of neighbors of a node let $S^<$ denote the subset of S whose rank is at most the median rank of the nodes in S. When recoloring v, we pick a color randomly out of all colors that are either (i) used by none of its neighbors (called *blank* colors) or (ii) by at most one neighbor in L_v *and* this node belongs to $L_v^<$. (We show that there are always $\Omega(|L_v|)$ many such colors.) In case (ii) this neighbor w must be recolored. Due to the definition of $L_v^<$ it is guaranteed that $r(w)$ is at most the median rank of the lower-ranked neighbors of v. Recoloring w is done with a more refined recoloring procedure that additionally to the above information takes into account which nodes of L_w also belong to $N(v)$, the neighborhood of v. This is necessary since on the one side (a) we need to guarantee that the new color is chosen randomly from a set of $\Omega(|L_w|)$ colors and the other side (b) we have to apply a different analysis depending on whether the new color belongs to $N(v)$ or not.

More formally let $L_{w,\text{new}} := L_w \setminus N(v)$, let $L_{w,\text{old}} := L_w \cap N(v)$, and let L^* equal $L^<_{w,\text{new}}$ if $|L_{w,\text{new}}| > |L_w|/10$ and $L^<_{w,\text{old}}$ otherwise. The algorithm randomly samples a color out of the set which consists of (i) all blank colors and (ii) all colors which are used by exactly one node in L_w and are used by a node in L^*. If the color of a node y in L^* was chosen, y will be recolored recursively taking $N(x)$ for *all* previously visited nodes x into account. If y was chosen from $L^<_{w,\text{new}}$, y is called a *good* vertex, otherwise a *bad* vertex. This results in a recoloring of nodes along a random *recoloring path* P in the graph until a blank color is chosen. The latter is guaranteed to happen when a node y with $L_y = \emptyset$ is reached. We give a data structure that implements each *coloring step*, i.e., the selection of a new color of a vertex y on P, in time $O(|L_y|)$. Thus, the total time for recoloring P is $O(\sum_{y\in P} |L_y|)$.

This sampling routine guarantees that the rank of the next node is at most the median rank of the lower-ranked neighbors of the previous node. If there were no dependencies between the rank of the current node and the previous nodes on P, the *expected rank* would halve in this coloring step. These dependencies are exactly why we introduced $L_{y,\text{new}}$, $L_{y,\text{old}}$, and L^*, and labeled the vertices on P as good and bad. More specifically, we show that at every good vertex y the *expected rank* and the *expected size of* $L_{y,new}$ halves. This by itself would not be sufficient, since we need the expected size of L_y, and not only the expected size of $L_{y,\text{new}}$, to halve. Here we use the definition of L^* to show that the expected size of L_y decreases by *a constant factor* whenever $L_{y,\text{new}}$ halves. This then implies that the total expected time at the good vertices on P, i.e. $O(\sum_{y\in P, y:\text{good}} |L_y|)$, forms a geometric series adding up to $O(r(v)\Delta)$, where v is the initial vertex of P.

The main difficulty that the analysis still has to overcome is the fact that there might be bad vertices. To deal with this we introduce a novel potential function Φ based on the nodes on P, which allows us to bound the work, i.e., the number of ("standard" word) operations that the algorithm performs, done at *bad* vertices by the work done at *good* vertices. More specifically, we show that, when traversing P from an initial vertex v, at every bad vertex Φ drops. As (i) Φ is always non-negative, (ii) Φ only increases at good vertices, and (iii) the drop of Φ gives an upper bound of the time spent at bad vertices, we can *bound the total time for coloring all the vertices on P by the total time spent at the good vertices on P times a constant.* This allows us to prove that the total work done for recoloring all vertices on P is $O(r(v)\Delta)$, where v is the initial vertex of P (Lemma 4).

Finally, we combine this bound with the fact that (a) for many operations (such as all deletions and many insertions) no recoloring is necessary and (b) the color of each node y was picked uniformly at random from a set of $\Omega(|L_y|)$ many colors, to show that the expected amortized time per update operation is constant.

Note that the refined sampling routine as well as the analysis that combines a potential function analysis with a careful analysis of the expected size of the sets L_y along a random path P is novel. The technique has the advantage that, unlike in a hierarchical graph decomposition where the ordering of nodes by levels might change and needs to be updated, the ordering of nodes by ranks is static and does not create update costs. However, it has the disadvantage that, unlike in the hierarchical graph decomposition of [5], (1) we do not have a worst-case upper bound on the number of nodes that are "lower" in the ordering and (2) the length of P, which is limited by the longest strictly decreasing path in the ordering, might be $\Theta(n)$ and not $\Theta(\log \Delta)$ in the worst case, as in [5].

As we recently learnt, Bhattacharya et al. [6] achieved the same result as Theorem 1 independently.

Our proof of Theorem 2 follows from a simple reduction from dynamic connectivity, whose cell probe lower bound was known to be $\Omega(\log n)$ [24].

Other Related Work. Partially due to the $\Omega(\log n)$ lower bound for the fundamental problem of testing connectivity [24], a large amount of previous research on dynamic graph algorithms has focused on algorithms with polylogarithmic or super-polylogarithmic update time. Examples include testing k-edge (or vertex) connectivity (see e.g., [14, 18, 17]), maintaining minimum spanning tree (see e.g., [15, 14, 17, 16, 18, 19, 20, 28, 22, 23]), and graph coloring [2, 1, 5, 26, 13]. There are also studies on *incremental algorithms* that only allow edge insertions, and *decremental algorithms* that only allow edge deletions throughout all the updates. In contrast to such studies, our work is focusing on *fully dynamic* algorithms, in which both edge insertions and deletions are allowed.

The technique of maintaining random ranks for vertices was previously used for dynamic maximal independent sets in the distributed setting [10] and very recently in the centralized setting [11, 4]. However, our analysis is quite different from theirs.

2 Maintaining a Proper $(\Delta + 1)$-Vertex Coloring

In this section, we give our constant-time dynamic algorithm and its analysis for maintaining a proper $(\Delta+1)$-coloring in a dynamic Δ-bounded graph and present the proof of Theorem 1. In Section 4, we discuss how to extend our algorithm to handle the case that the maximum degree Δ also changes. Recall that a dynamic graph is said to be Δ-bounded if throughout the updates, it is Δ-bounded. Given Δ, let $\mathcal{C} := \{1, \cdots, \Delta + 1\}$ denote the set of *colors*. A coloring $\chi : V \to \mathcal{C}$ is *proper* if $\chi(u) \neq \chi(v)$ for any $(u, v) \in E$.

2.1 Data Structures and the Algorithm

Data structures. We use the following data structures.

(1) We maintain a vertex coloring χ as an array such that $\chi(v)$ denotes the color of the current graph and guarantee that χ is a proper $(\Delta+1)$-vertex coloring after each update.

(2) For each vertex $v \in V$ we maintain: (a) its rank $r(v)$ that is chosen uniformly at random from $[0,1]$ during preprocessing; (b) its degree $\deg(v)$; (c) the last time stamp, denoted by τ_v, at which v was recolored; (d) two sets $L_v := \{u : (u,v) \in E, r(u) < r(v)\}, H_v := \{u : (u,v) \in E, r(u) \geq r(v)\}$, which contain all neighbors of v with ranks less than v, and all neighbors of v with ranks at least v (including v itself), respectively; (e) the sizes of the previous two sets, i.e., $|L_v|$ and $|H_v|$. Note that $\deg(v) = |L_v \cup H_v| = |L_v| + |H_v|$. For each vertex $v \in V$ note that every color of $\mathcal{C}$ is either (i) used by no neighbor of v (and we call such color a *blank* color for v), (ii) used by a neighbor in H_v, or (iii) used by a neighbor in L_v and by *no* neighbor in H_v. We call the corresponding sets of colors (i) $\mathcal{B}_v$, (ii) $\mathcal{C}_v(H)$, and (iii) $\mathcal{C}_v(L)$. We further partition $\mathcal{C}_v(L)$ into (iii.1) $\mathcal{U}_v(L)$, which denotes the set of *unique* colors for v that have been used by exactly one vertex in L_v and (iii.2) $\mathcal{M}_v(L)$, which denotes the set of colors that have been used by at least two vertices in L_v. Thus, $\mathcal{C} = \mathcal{C}_v(H) \dot\cup \mathcal{B}_v \dot\cup \mathcal{U}_v(L) \dot\cup \mathcal{M}_v(L)$. As it will be useful in the description of the algorithm, we finally define $\mathcal{C}_v(\overline{H}) := \mathcal{B}_v \cup \mathcal{U}_v(L) \cup \mathcal{M}_v(L)$. Note that for any fixed v, a color c can appear in exactly one of the two sets $\mathcal{C}_v(H)$ and $\mathcal{C}_v(\overline{H})$.

(3) (i) For every vertex v, we maintain $\mathcal{C}_v(H)$ and $\mathcal{C}_v(\overline{H})$ in doubly linked lists. (ii) For each color $c \in \mathcal{C}$ and vertex $v \in V$, we keep the following information: (a) a pointer $p_{c,v}$ from c to its position in either $\mathcal{C}_v(H)$ or $\mathcal{C}_v(\overline{H})$, depending on which list it belongs to; (b) a counter $\mu_v^H(c)$ such that $\mu_v^H(c)$ equals the number of neighbors in H_v with color c if $c \in \mathcal{C}_v(H)$; or equals 0 if $c \in \mathcal{C}_v(\overline{H})$. (iii) For any vertex v and color $c \in \mathcal{C}$ we keep the pointer $p_{c,v}$ in a hash table $\mathcal{A}_v$ which is indexed by c. (iv) For any vertex v and color $c \in \mathcal{C}_v(H)$, we maintain the pairs $(c, \mu_v^H(c))$ in a hash table $\mathcal{A}_v^H$ which is indexed by the pair (v,c).

More precisely, we use the dynamic perfect hashing algorithm by Dietzfelbinger et al. [12], which takes amortized expected constant time per update and worst-case constant time for lookups. (Alternatively we can get constant worst-case time for updates and lookups by spending time $O(n\Delta)$ during preprocessing to initialize suitable arrays).

To simplify the presentation and since the randomness in the hash tables is independent of the randomness used by the algorithm otherwise, we will not mention the randomness introduced through the usage of hash tables in the following.

Initialization. As the initial graph G_0 is empty, we initialize as follows: (1) For each vertex $u \in V$, sample a random number (called *rank*) $r(u) \in [0,1]$. (2) Color each vertex u by a random color $\chi(u) \in \mathcal{C} := \{1, \cdots, \Delta+1\}$ and initialize all the data structures suitably. In particular, for each $u \in V$, we initialize $\mathcal{C}_u(H)$ to be the empty list and $\mathcal{C}_u(\overline{H})$ to be the doubly linked list containing all colors in $\mathcal{C}$. Note that the latter takes $O(n\Delta)$ time. We discuss how to reduce the initialization time to $O(n)$ while keeping constant expected amortized update time in Section 4.

Time stamp reduction. Our algorithm does not use the actual values of the time stamps, only their relative order. Thus, every $\text{poly}(n)$ (say, n^4) number of updates we determine the order of the vertices according to the time stamps and set the time stamps of every vertex to equal its position in the order and set the current time stamp to $n+1$. This guarantees that we only need to use $O(\log n)$ bits to store the time stamp τ_v for each vertex v and it does

not affect the ordering of the time stamps. The cost of the recomputation of time stamps is $O(n \log n)$ and can be amortized over all the operations that are performed between two updates, increasing their running time only by an additive constant.

Handling an edge deletion. As any edge deletion (u, v) does not lead to a violation of the current proper coloring, we do not need to recolor any vertex, except to update the data structures corresponding to u, v, the details of which are deferred to Section 2.1.1.

Handling an edge insertion. For an edge insertion (u, v), we note that if $\chi(u) \neq \chi(v)$ before the insertion, then we only need to update the basic data structures corresponding to the two endpoints. If $\chi(u) = \chi(v)$, i.e, the current coloring χ is not proper any more, then we need to recolor one vertex $w \in \{u, v\}$ as well as to update the relevant data structures. We always recolor the vertex that was colored last, i.e., the one with larger τ_w. W.l.o.g., we assume this vertex is v. Then we invoke a subroutine RECOLOR(v) to recolor v and potentially some other lower level vertices, and update the corresponding data structures. That is, we will first update H_u, L_u, H_v, L_v and their sizes trivially in constant time. Then if $\chi(u) \neq \chi(v)$, we update the data structures corresponding to u, v as described in Section 2.1.1.

If $\chi(u) = \chi(v)$, and w.l.o.g., suppose that $\tau_v > \tau_u$, then we recolor v by invoking the procedure RECOLOR(v) below, where $\mathcal{U}_v(L)$ denotes the set of colors that have been used by *exactly one* vertex in L_v.

RECOLOR(v)

1. Run SETCOLOR(v) and obtain a new color c (from $\mathcal{B}_v \cup \mathcal{U}_v(L)$).
2. Set $\chi(v) = c$. Update the data structures by the process $(*)$ described in Section 2.1.1.
3. If $c \in \mathcal{U}_v(L)$,
   a. Find the unique neighbor $w \in L_v$ with $\chi(w) = c$.
   b. RECOLOR(w).
4. If $c \in \mathcal{B}_v$, then remove all the **visited** marks generated from the calls to SETCOLOR.

Note that the recursive calls will eventually terminate as for every call RECOLOR(w) in Step 3 it holds that $r(w) < r(v)$. Furthermore, no recursive call will be performed when $L_v = \emptyset$ as it implies that $\mathcal{U}_v(L) = \emptyset$. The subroutine RECOLOR$(v)$ calls the following subroutine SETCOLOR(v).

2.1.1 Updating the Data Structures

Case I: an edge deletion (u, v). Whenever an edge (u, v) gets deleted, we update the data structures corresponding to u and v as follows. More precisely, we first update the sets H_u, L_u, H_v, L_v and their sizes trivially in constant time. The lists $\mathcal{C}_u(H), \mathcal{C}_u(\overline{H}), \mathcal{C}_v(H), \mathcal{C}_v(\overline{H})$ can be updated in constant worst-case time. The hash tables $\mathcal{A}_u^H, \mathcal{A}_v^H$ can also be maintained in constant amortized expected update time. More precisely, suppose w.l.o.g., $u \in L_v$, then we do the following:

1. Delete $(\chi(v), \mu_u^H(\chi(v)))$ from $\mathcal{A}_u^H$; $\mu_u^H(\chi(v)) \leftarrow \mu_u^H(\chi(v)) - 1$.
2. If $\mu_u^H(\chi(v)) = 0$, then $\mathcal{C}_u(H) \leftarrow \mathcal{C}_u(H) \setminus \{\chi(v)\}$, $\mathcal{C}_u(\overline{H}) \leftarrow \mathcal{C}_u(\overline{H}) \cup \{\chi(v)\}$.
3. Otherwise, insert $(\chi(v), \mu_u^H(\chi(v)))$ to $\mathcal{A}_u^H$.

SetColor(v)

1. Mark v as **visited**. Initialize sets $L_{v,\text{old}} := \{v\}$ and $L_{v,\text{new}} := \emptyset$.
   Scan the list L_v: for any $u \in L_v$, if it is marked as **visited**, then add u to $L_{v,\text{old}}$; otherwise (i.e., it is not marked), then add u to $L_{v,\text{new}}$ and mark u as **visited**.
2. If $|L_v| + |H_v| < \frac{\Delta}{2}$ (i.e., $\deg(v) < \frac{\Delta}{2}$), repeatedly sample a color uniformly at random from $[\Delta + 1]$ until we get a color c that is contained in $\mathcal{B}_v$, the set of *blank* colors for v that have not been used by any neighbor of v.
3. Otherwise, we let $L^{<}_{v,\text{new}}$ denote the subset of vertices in $L_{v,\text{new}}$ with ranks at most the median of all ranks of vertices in $L_{v,\text{new}}$. We let $\mathcal{U}_v(L^{<}_{\text{new}})$ denote the set of colors that each has been used by exactly one vertex in $L_{v,\text{new}}$ and additionally this vertex belongs to $L^{<}_{v,\text{new}}$. Define $L^{<}_{v,\text{old}}$ and $\mathcal{U}_v(L^{<}_{\text{old}})$ similarly.
   a. If $|L_{v,\text{new}}| \geq \frac{1}{10}|L_v|$ or $L_v = \emptyset$, then we sample a random color c from the set of the first $\min\{|\mathcal{B}_v \cup \mathcal{U}_v(L^{g}_{\text{new}})|, |L^{<}_{v,\text{new}}| + 1\}$ elements of $\mathcal{B}_v \cup \mathcal{U}_v(L^{<}_{\text{new}})$.
   b. Else (i.e., $|L_{v,\text{old}}| > \frac{9}{10}|L_v|$) we sample a random color c from the set of the first $\min\{|\mathcal{B}_v \cup \mathcal{U}_v(L^{<}_{\text{old}})|, |L^{<}_{v,\text{old}}| + 1\}$ elements of $\mathcal{B}_v \cup \mathcal{U}_v(L^{<}_{\text{old}})$.
4. Update the relevant data structures (i.e. of v and its neighbors in L_v) and **Return** c.

Case II: an edge insertion (u, v) such that $\chi(u) \neq \chi(v)$. In this case, w.l.o.g., suppose that $r(u) < r(v)$, we update the data structures as follows:

1. $\mathcal{C}_u(H) \leftarrow \mathcal{C}_u(H) \cup \{\chi(v)\}$, $\mathcal{C}_u(\overline{H}) \leftarrow \mathcal{C}_u(\overline{H}) \setminus \{\chi(v)\}$, $\mu^H_u(\chi(v)) \leftarrow \mu^H_u(\chi(v)) + 1$
2. Delete $(\chi(v), \mu^H_u(\chi(v)) - 1)$ from $\mathcal{A}^H_u$ if $\mu^H_u(\chi(v)) > 1$, insert $(\chi(v), \mu^H_u(\chi(v)))$ to $\mathcal{A}^H_u$.

Case III: procedure (⁎) in the subroutine Recolor(v). In the subroutine RecolorMis(v), if the color of v is changed from c' to c, then we update the relevant data structure as follows:

(⁎) For every $w \in L_v$:

1. $\mu^H_w(c') \leftarrow \mu^H_w(c') - 1$
2. If $\mu_w(c') = 0$, then $\mathcal{C}_w(H) \leftarrow \mathcal{C}_w(H) \setminus \{c'\}$, $\mathcal{C}_w(\overline{H}) \leftarrow \mathcal{C}_w(\overline{H}) \cup \{c'\}$,
3. $\mathcal{C}_w(H) \leftarrow \mathcal{C}_w(H) \cup \{c\}$, $\mathcal{C}_w(\overline{H}) \leftarrow \mathcal{C}_w(\overline{H}) \setminus \{c\}$, $\mu^H_w(c) \leftarrow \mu^H_w(c) + 1$.
4. Delete $(c, \mu^H_w(c))$ from $\mathcal{A}^H_w$ if $\mu^H_w(c) > 1$, and insert $(c, \mu^H_w(c))$ to $\mathcal{A}^H_w$.

2.2 The Analysis

Next we prove Theorem 1. Let $v_0 := v$ be the vertex that needs to be recolored after an insertion and let $v_1, v_2, \cdots, v_\ell$ denote the vertices on which the recursive calls of Recolor() were executed. We call $v_0, v_1, \cdots, v_\ell$ the *recoloring path* originated from v. In the following lemma, we show that the expected total time for all calls Recolor(v_i) is $O(1 + \sum_{i=0}^{\ell} |L_{v_i}|)$, where the expectation is *not* over the random choices of ranks or colors at Step 3, but comes from the use of hash tables and sampling colors at Step 2.

▶ **Lemma 3.** *Subroutine* SetColor(v) *can be implemented to run in $O(1 + |L_v|)$ expected time. For any recoloring path $v_0, v_1, \cdots, v_\ell$, the expected time for subroutine* Recolor(u) *for any $u \in \{v_1, \ldots, v_l\}$ excluding the recursive calls to* Recolor() *is $O(|L_u|)$ if $u \neq v_\ell$, and is $O(1 + \sum_{i=0}^{\ell} |L_{v_i}|)$ if $u = v_\ell$.*

Proof. Recall that we store L_v, $\mathcal{C}_v(H)$, and $\mathcal{C}_v(\overline{H})$ for every vertex v. We use them to build all the sets needed in SetColor(v). First we use an array $R_{v,L_{\text{new}}}$ (resp. $R_{v,L_{\text{old}}}$) to store ranks of vertices in $L_{v,\text{new}}$ (resp. $L_{v,\text{old}}$), and then find the median $m_{v,\text{new}}$ (resp. $m_{v,L_{\text{old}}}$) of the set of ranks of vertices in $L_{v,\text{new}}$ (resp. $L_{v,\text{old}}$) deterministically in $O(|R_{v,L_{\text{new}}}|) = O(|L_v|)$

time [8]. Traversing L_v again (and using an empty array of length Δ that we clean again after this step) we compute (1) the sets $\mathcal{U}_v(L^<_{\text{new}})$ and $\mathcal{U}_v(L^<_{\text{old}})$ of colors that contain all colors that have been used by *exactly one* vertex in $L^<_{v,\text{new}}$, and by *exactly one* vertex in $L^<_{v,\text{old}}$, respectively, and (2) the sets $\mathcal{M}_v(L)$ of colors that contain all colors that have been used by *at least two* vertices in L_v. Note that $\mathcal{U}_v(L) = \mathcal{U}_v(L^<_{\text{new}}) \cup \mathcal{U}_v(L^<_{\text{old}})$, and, thus, it can be computed by copying these lists. All these lists have size $O(|L_v|)$ and, thus, all these steps take time $O(|L_v|)$.

We will keep the sets $\mathcal{M}_v(L)$, $\mathcal{U}_v(L)$, $\mathcal{U}_v(L^<_{\text{new}})$, $\mathcal{U}_v(L^<_{\text{old}})$ in four separate lists and build hash tables for these sets with pointers to their positions in the lists. Next we delete all colors in $\mathcal{M}_v(L) \cup \mathcal{U}_v(L)$ from the list $\mathcal{C}_v(\overline{H})$ and the resulting list will be $\mathcal{B}_v$. Note that the hash tables can be implemented in time linear in the size of corresponding sets, and each lookup (i.e., check if an element is in the set) takes constant worst-case time [12]. This completes the building of the data structure before Step 1.

Recall that $|L_v| + |H_v| = \deg(v)$. Then for Step 2, if $\deg(v) < \frac{\Delta}{2}$, we know that $|\mathcal{B}_v| > \Delta - \frac{\Delta}{2} = \frac{\Delta}{2}$. Thus, a randomly sampled color from $[\Delta+1]$ belongs to $\mathcal{B}_v$ with probability at least $1/2$, which implies that in $O(1)$ expected time, we will sample a color c from $\mathcal{B}_v$. Note that a color c belongs to $\mathcal{B}_v$ if and only if c is not contained in $\mathcal{M}_v(L) \cup \mathcal{U}_v(L) \cup \mathcal{C}_v(H)$, which can be checked by using the hash tables for $\mathcal{M}_v(L)$, for $\mathcal{U}_v(L)$ and the hash table $\mathcal{A}_v^H$.

All the other steps only write, read and/or delete lists or hash tables of size proportional to $|L_v|$ or $|\mathcal{M}_v(L) \cup \mathcal{U}_v(L)|$, which is at most $|L_v|$. Though the list $\mathcal{B}_v \cup \mathcal{U}_v(L^<_{\text{new}})$ might have size much larger than $|L^<_{v,\text{new}}|$, it suffices to read at most $|L^<_{v,\text{new}}|$ elements from it in Step 3 (similar for $\mathcal{B}_v \cup \mathcal{U}_v(L^<_{\text{old}})$ versus $|L^<_{v,\text{old}}|$). In Step 4, to update the relevant data structures, we add all colors in $\mathcal{M}_v(L) \cup \mathcal{U}_v(L)$ back to the list $\mathcal{B}_v$ to construct $\mathcal{C}_v(\overline{H})$. Thus, SetColor($v$) takes $O(1 + |L_v|)$ expected time.

To analyze the running time of Recolor(u) (apart from the recursive calls), for any $u \in v_0, v_1, \ldots, v_\ell$, note that apart from calling SetColor(u), Recolor updates the data structures, determines the neighbor w that needs to be recolored next (if any) and if no such neighbor w exists, i.e. c is a blank color and u is the last vertex of the recoloring path, then it unmarks all vertices that were marked by all the calls to SetColor on the recoloring path. For this SetColor has stored all the marked vertices on a list, which it returns to Recolor. This list is then used by recolor to unmark these vertices. The time to update the data structures is constant expected time (the expectation arises due to the use of hash tables) to update its own data structure and $O(|L_u|)$ to update the data structures of its lower neighbors. Determining w requires $O(|L_u|)$ time, as all lower neighbors of u have to be checked. Finally, Recolor(u) for the last vertex $u = v_\ell$ on the recoloring path takes expected time $O(1 + \sum_i |L_{v_i}|)$ as it unmarks all vertices on the recoloring path and their neighbors. ◀

Throughout the process we have two different types of randomness: one for sampling the ranks for the vertices and the other for sampling the colors. These two types of randomness are independent. Furthermore, only the very last vertex v_ℓ on the recoloring path $P = v_0, v_1, \cdots, v_\ell$ can satisfy the condition of Step 2 in SetColor, as once the condition is satisfied, we will sample a blank color which will not cause any further recursive calls. Thus, for all vertices on P, with the possible exception of v_ℓ, Step 3 will be executed. We call a vertex w with $\deg(w) < \frac{\Delta}{2}$ a *low degree* vertex. Note that for a low degree vertex w, SetColor(w) executes Step 2 and takes $O(1)$ expected time, as with probability at least $1/2$ a randomly sampled color will be blank. In the following, we consider the expected time T_v of recoloring P that excludes the time of recoloring any low degree vertex (which, if

exists, must be the last vertex on P). We first present a key property regarding the expected running time for recoloring a vertex v. Let $N(v)$ denote the set of all neighbors of v in the current graph.

▶ **Lemma 4.** *Let G denote the current graph. For any vertex v with rank $r(v) \leq \alpha$, the expected running time T_v (over the randomness of choosing ranks of other vertices) is*

$$E[T_v | r(v) \leq \alpha] = O(\alpha\Delta) \tag{1}$$

Furthermore, conditioned on ranks of vertices in $N(v)$ and $r(v) \leq \alpha$, it holds that the expected running time T_v (over the randomness of sampling ranks of $V \setminus (N(v) \cup \{v\})$) is

$$E[T_v | r(v) \leq \alpha, r(w) \forall w \in N(v)] = O(|L_v|) + O(\alpha\Delta) \tag{2}$$

The proof of the above lemma is deferred to Section 2.2.1. We remark that Lemma 4 assumes that for each operation, it is executed in any possible current graph G with any proper $(\Delta + 1)$-coloring (i.e. worst-case analysis for graph and coloring) and that each rank is sampled uniformly at random from $[0, 1]$ in G. This is true as the adversary is assumed to be oblivious, i.e., the sequence of all updates has been written down before the algorithm starts to process the updates. That is, for any current graph G, the random ranks of vertices still follows from the same distribution as the one in the beginning. The above further implies that we can bound the work for recoloring a conflicting vertex v in G by a function that depends only on the randomness for sampling ranks (and *not* on the randomness for selecting colors in previous updates).

We will also need the following lemma regarding the size of the sampled color set. The proof of the lemma follows from a more refined analysis of the proof of Claim 3.1 in [5] and can be found in the full version of the paper.

▶ **Lemma 5.** *Let v be any vertex that needs to be recolored. Let s denote the size of the set of colors that the algorithm samples from in order to choose a new color for v. Then it holds that 1) if $|L_v| + |H_v| < \frac{\Delta}{2}$, then $s \geq \frac{\Delta}{2} + 1$; 2) otherwise, $s \geq \frac{1}{100}|L_v| + 1$.*

With the lemmas above, we are ready to prove Theorem 1.

Proof of Theorem 1. Note that an edge deletion does not lead to the recoloring of any vertex. Let us consider an insertion (u, v). If $\chi(u) \neq \chi(v)$, we do not need to recolor any vertex. Otherwise, we need to recolor one vertex from $\{u, v\}$. Suppose w.l.o.g. that $\tau_v > \tau_u$, where τ_u denotes the last time that u has been recolored. This implies that v is recolored at the current time step, which we denote by τ. We will invoke RECOLOR(v) to recolor v. Note that by definition, after calling subroutine RECOLOR, there will be no conflict in the resulting coloring. This proves the correctness of the algorithm. In the following, we analyze its running time.

Recall that we let T_v denote the running time of calling RECOLOR(v), including all the recursive calls to RECOLOR, while *excluding* the time of recoloring any low degree vertex (i.e. a vertex where SETCOLOR(w) executed Step 2) on the recoloring path originated from v (which, if exists, must be the last vertex on the path). If the last vertex is indeed a low degree vertex, then the expected total running time (over all sources of randomness) of RECOLOR(v) will be $\mathrm{E}[T_v] + O(1)$, where the expectation $\mathrm{E}[T_v]$ in turn is over the randomness of sampling ranks of all vertices; otherwise, the expected total running time (over all sources of randomness) of RECOLOR(v) will be $\mathrm{E}[T_v]$. Let $\alpha_0 = \frac{4C \log \Delta}{\Delta}$ for some constant $C \geq 1$. Now we consider two cases:

Case I: $r(v) \leq \alpha_0$. First we note that this case happens with probability at most α_0 as $r(v)$ is chosen uniformly at random from $[0,1]$. Furthermore, by Lemma 4, conditioned on the event that $r(v) \leq \alpha_0$, the expected time of the subroutine $\textsc{Recolor}(v)$ is $\mathrm{E}[T_v | r(v) \leq \alpha_0] = O(\alpha_0 \Delta)$, where the expectation is taken over the randomness of choosing ranks of all other vertices except v. Therefore, the expected time of $\textsc{Recolor}(v)$ (over the randomness of choosing ranks of all vertices) is at most $\alpha_0 \cdot O(\alpha_0\Delta) = O(\alpha_0^2\Delta) = O(\frac{\log^2 \Delta}{\Delta}) = O(1)$.

Case II: $r(v) > \alpha_0$. Let $r(v) = \alpha$. Conditioned on the event that $r(v) = \alpha$, by Lemma 4, the expected running time (over the randomness of choosing ranks of other vertices) of $\textsc{Recolor}(v)$ at time τ is $O(\alpha\Delta)$.

We let L_v and L'_v denote the set of neighbors of v with ranks lower than v in the graph at (current) time τ and at time τ_v, (the latest time that v was recolored), respectively. Note that $\tau_u < \tau_v$ implies that *neither $\chi(u)$ nor $\chi(v)$ changed between τ_v and τ*. We define H_v, H'_v similarly. We let $\deg(v) = |L_v \cup H_v|$ and $\deg'(v) = |L'_v \cup H'_v|$ denote the degree of v at time τ and τ_v, respectively.

Case (a): $\deg'(v) < \Delta/2$. In this case, we know that at time τ_v, we will sample a color from the set of blank colors $\mathcal{B}(v)$, which has size at least $\Delta/2$. Thus, the probability that we sampled *any fixed* color at time τ_v is at most $2/\Delta$. This also applies to the color $\chi(u)$. Thus, the probability that $\chi(v) = \chi(u)$ at time τ_v is at most $2/\Delta$. As neither $\chi(v)$ nor $\chi(u)$ have changed between τ_v and τ (which implies that the random choices of the algorithm between τ_v and τ have no influence on $\chi(v)$ or $\chi(u)$), the probability that $\chi(v) = \chi(u)$ at time τ is at most $2/\Delta$. On the other hand, at time τ, we will spend at most $O(\alpha\Delta) = O(\Delta)$ expected time (over the randomness of sampling ranks of vertices in $V \setminus \{v\}$). Thus, the expected time (over the randomness of sampling ranks and of sampling colors at time τ_v) we spent on recoloring v at time τ is $O(\frac{1}{\Delta} \cdot \Delta) = O(1)$.

Case (b): $\deg'(v) \geq \Delta/2$. We now consider two sub-cases.

Case (b1): If $\deg(v) < \Delta/4$, then there must have been at least $\deg'(v)/2 = \Omega(\Delta)$ deletions of edges incident to v between τ_v and τ. We can recolor v at time τ in expected $O(\alpha\Delta) = O(\Delta)$ time. We charge this time to the updates incident to v between τ_v and τ. Note that each update is only charged twice in this way, once from each endpoint, adding a constant amount of work to each deletion.

Case (b2): If $\deg(v) \geq \Delta/4$, then $\mathrm{E}[|L_v|] = \alpha \deg(v) \geq \alpha\Delta/4 \geq \alpha_0\Delta/4 \geq C \log \Delta$ for some constant $C \geq 1$ and $\mathrm{E}[|L_v|] = \alpha \deg(v) \leq \alpha\Delta$. Then over the randomness of sampling ranks for vertices in $N(v)$, it follows from a Chernoff bound that with probability at least $1 - \frac{1}{\Delta}$, $\frac{\mathrm{E}[|L_v|]}{2} \leq |L_v| \leq \frac{3\mathrm{E}[|L_v|]}{2}$, which implies that with probability at least $1 - \frac{1}{\Delta}$,

$$(\alpha\Delta)/8 \leq \mathrm{E}[|L_v|]/2 \leq |L_v| \leq (3\mathrm{E}[|L_v|])/2 \leq (3\alpha\Delta)/2 \tag{3}$$

By Ineq. (2) in Lemma 4, over the randomness of sampling ranks for $V \setminus (N(v) \cup \{v\})$, the expected work for recoloring v at time τ is $O(|L_v|) + O(\alpha\Delta) = O(\alpha\Delta)$. We first analyze the case that Ineq. (3) does not hold, which happens with probability at most $1/\Delta$. Then the work for recoloring is $O(\Delta)$ as $|L_v| \leq \Delta$. Thus the expected work of this case is $\frac{1}{\Delta} \cdot O(\Delta) = O(1)$.

Next we analyze the case that Ineq. (3) holds and further distinguish two sub-cases.

Case (b2-1): If $|L_v \triangle L'_v| > \frac{1}{10}|L_v|$, then there must have been at least $\frac{1}{10}|L_v| = \Theta(\alpha\Delta)$ edge updates incident to v between τ_v and τ. By the same argument as above we can amortize the expected work of $O(\alpha\Delta)$ over these edge updates, charging each edge update at most twice. This adds an expected amortized cost of $O(1)$ to each update.

Case (b2-2): If $|L_v \triangle L'_v| \leq \frac{1}{10}|L_v|$, then it holds that $|L'_v| \geq |L_v| - |L_v \triangle L'_v| \geq \frac{9}{10}|L_v|$. By Lemma 5, $\chi(v)$ was picked at time τ_v from a set of $\Omega(|L'_v|)$ many colors. By similar argument for the Case (a), the probability that we picked the color $\chi(u)$ at time τ_v is at most $O(\frac{1}{|L'_v|}) = O(\frac{1}{|L_v|})$. As the expected work at time τ is at most $O(\alpha\Delta) = O(|L_v|)$ (with the expectation over randomness of sampling ranks), the expected amortized update time is $O(\frac{1}{|L_v|}) \cdot O(|L_v|) = O(1)$.

This completes the proof of the theorem. ◀

2.2.1 Bounding the Expected Work per Recoloring: Proof of Lemma 4

Let $v_0, v_1, \cdots$ be the vertices on the recoloring path after an insertion. By Lemma 3 the total expected time for all calls RECOLOR(v_i) is $O(1 + \sum_{i \geq 0} |L_{v_i}|)$. Recall that the running time T_v excludes the time spent on recoloring a low degree vertex (and a low degree vertex can only be the last vertex of a recoloring path). Thus, for all vertices v_i that contribute to T_v only Step 3a or Step 3b of SETCOLOR can occur. Let $v_{i_0} = v_0, v_{i_1}, v_{v_2}, \cdots$ be the vertices for which Step 3a occurred during SETCOLOR(v), which we call *good* vertices. We bound the expected value of ranks of good vertices and the expected size of the lower-ranked neighborhood of these vertices in the following lemma. Note that the expectations are taken over the randomness for sampling ranks of vertices, whose ranks are *not* in the conditioned events.

▶ **Lemma 6.** *For any $j \geq 0$, it holds that*

$$E[r(v_{i_j+1})|r(v_0) \leq \alpha] \leq \alpha/2^j, \quad E[|L_{v_{i_j}}||r(v_0) \leq \alpha] \leq (10 \cdot \alpha \cdot \Delta)/2^{j-1}.$$

Furthermore, for any $j \geq 1$, it holds that

$$\begin{aligned} E[r(v_{i_j+1})|r(v_0) \leq \alpha, r(w) \forall w \in N(v_0)] &\leq \alpha/2^{j-1}, \\ E[|L_{v_{i_j}}||r(v_0) \leq \alpha, r(w) \forall w \in N(v_0)] &\leq (10 \cdot \alpha \cdot \Delta)/2^{j-2}. \end{aligned}$$

Proof. To prove the lemma, we use the principle of deferred decisions: Instead of sampling the ranks for all vertices (independently and uniformly at random from $[0, 1]$) at the very beginning, we sample the ranks of vertices sequentially by the following random process:

Starting from v_0 with rank $r(v_0)$, we sample all the ranks of vertices in $N(v_0)$. We will then choose v_1 as described in the algorithm RECOLOR (if a non blank color has been sampled). Now for each $i \geq 1$, we note that the ranks of all the vertices in $N_{\text{old}}(v_i) := N(v_i) \cap (\cup_{j<i} N(v_j) \cup \{v_0\})$ have already been sampled, and then we only need to sample (independently and uniformly at random from $[0, 1]$) the ranks for all vertices in $N_{\text{new}}(v_i) := N(v_i) \setminus N_{\text{old}}(v_i)$. In this case, we say that the ranks of vertices in $N_{\text{new}}(v_i)$ are sampled *when we are exploring* v_i. Then we will choose v_{i+1} in the algorithm RECOLOR (if a non blank color has been sampled). We iterate the above process until RECOLOR has sampled a blank color.

For any i, we call $N_{\text{new}}(v_i)$ the *free neighbors* of v_i with respect to $v_0, v_1, \cdots, v_{i-1}$. In particular, $N_{\text{new}}(v_0) = N(v_0)$ and $N(v_i) = N_{\text{new}}(v_i) \dot{\cup} N_{\text{old}}(v_i)$. Now a key observation is that

($\star$) for any vertex v_i, it holds that $L_{v_i,\text{new}}$ (as defined in the algorithm SETCOLOR(v_i)) is entirely determined by the ranks of the vertices $N_{\text{new}}(v_i)$ and is independent of the randomness for sampling ranks of $N_{\text{old}}(v_i)$.

This is true since $L_{v_i,\text{new}}$ contains all the neighbors of v_i with ranks less than $r(v_i)$ and have not been visited so far: for any vertex in $N_{\text{old}}(v_i)$, either its rank is higher than v_i, or its rank is less than v_i and it has been marked as **visited** before we invoke SetColor(v_i).

We first prove the first part of the lemma. We *assume for now that* $r(v_0)$ *is fixed* and we denote by $\mathcal{R}(i_j)$ the randomness of sampling ranks for vertices in $N_{\text{new}}(v_{i_j})$. We will prove by induction on the index j that

$$\mathrm{E}_{\mathcal{R}(i_j)}[r(v_{i_j+1})] \le r(v_0)/2^j \text{ and } \mathrm{E}_{\mathcal{R}(i_j)}[|L_{v_{i_j},\text{new}}|] \le (r(v_0)\cdot\Delta)/2^{j-1}. \tag{4}$$

Note that this holds for $j=0$ since $i_0=0$, $r(v_1)\le r(v_0)$, $L_{v_{i_0},\text{new}} = L_{v_0}$, and $\mathrm{E}_{\mathcal{R}(0)}[|L_{v_0}|] = r(v_0)\cdot|N(v_0)| \le r(v_0)\cdot\Delta$. Next we assume it holds for $j-1$, and prove it also holds for j. By the definition of the good vertex v_{i_j}, we know that $v_{i_j+1}\in L_{v_{i_j}}$, and that the rank of v_{i_j+1} is at most the median, denoted by $m_{v_{i_j},\text{new}}$, of all the ranks of vertices in $L_{v_{i_j},\text{new}}$, which in turn consists of all vertices in $N_{\text{new}}(v_{i_j})$ with rank not larger than $r(v_{i_j})$. Furthermore, by the observation $(\star)$, the rank of $r(v_{i_j+1})$ depends only on $r(v_{i_j})$ and the ranks in $N_{\text{new}}(v_{i_j})$. This implies that

$$\mathrm{E}_{\mathcal{R}(i_j)}[r(v_{i_j+1})|r(v_{i_j})] \le \mathrm{E}_{\mathcal{R}(i_j)}[m_{v_{i_j},\text{new}}|r(v_{i_j})] \le r(v_{i_j})/2,$$

where the last inequality follows from the fact that $m_{v_{i_j},\text{new}}$ is the median of a set of numbers chosen independently and uniformly at random from $[0,1]$, conditioned on that they are at most $r(v_{i_j})$ (see e.g., Lemma 8.2 and 8.3 in [21]). Since $r(v_{i_j}) \le r(v_{(i_{j-1})+1})$ in all cases and, by the induction assumption, $\mathrm{E}_{\mathcal{R}(i_{j-1})}[r(v_{(i_{j-1})+1})] \le \frac{r(v_0)}{2^{j-1}}$, it holds that

$$\begin{aligned}
&\mathrm{E}_{\mathcal{R}(i_j)}[r(v_{i_j+1})] \le \mathrm{E}_{r(v_{i_j})}[\mathrm{E}_{\mathcal{R}(i_j)}[r(v_{i_j+1})|r(v_{i_j})]] \le \frac{1}{2}\mathrm{E}_{r(v_{i_j})}[r(v_{i_j})] \\
\le\ & \frac{1}{2}\mathrm{E}_{\mathcal{R}(i_{j-1})}[\mathrm{E}_{r(v_{i_j})}[r(v_{i_j})|r(v_{(i_{j-1})+1})]] \le \frac{1}{2}\mathrm{E}_{\mathcal{R}(i_{j-1})}[r(v_{(i_{j-1})+1})] \le \frac{r(v_0)}{2^j}.
\end{aligned}$$

Furthermore, for any $j\ge 0$, by the observation $(\star)$, $L_{v_{i_j},\text{new}}$ depends only on $r(v_{i_j})$ and ranks in $N_{\text{new}}(v_{i_j})$. Thus

$$\mathrm{E}_{\mathcal{R}(i_j)}[|L_{v_{i_j},\text{new}}|\ |r(v_{i_j})] \le r(v_{i_j})\cdot|N_{\text{new}}(v_{i_j})| \le r(v_{i_j})\cdot\Delta.$$

This further implies that

$$\mathrm{E}_{\mathcal{R}(i_j)}[|L_{v_{i_j},\text{new}}|] = \mathrm{E}_{r(v_{i_j})}[\mathrm{E}_{\mathcal{R}(i_j)}[|L_{v_{i_j},\text{new}}|\ |r(v_{i_j})]] \le \mathrm{E}_{r(v_{i_j})}[r(v_{i_j})]\cdot\Delta \le \frac{r(v_0)\cdot\Delta}{2^{j-1}}.$$

Now let us no longer assume that $r(v_0)$ is fixed, but instead condition on the event that $r(v_0)\le\alpha$. Then it follows that $\mathrm{E}_{\mathcal{R}(i_j)}[r(v_{i_j+1})|r(v_0)\le\alpha] \le \frac{\alpha}{2^j}$ and $\mathrm{E}_{\mathcal{R}(i_j)}[|L_{v_{i_j},\text{new}}|\ |r(v_0)\le\alpha] \le \frac{\alpha\cdot\Delta}{2^{j-1}}$.

Now by the definition of good vertices, we have $|L_{v_{i_j},\text{new}}| \ge \frac{1}{10}|L_{v_{i_j}}|$. This implies that

$$\mathrm{E}_{\mathcal{R}(i_j)}[|L_{v_{i_j}}|\ |r(v_0)\le\alpha] \le 10\cdot\mathrm{E}_{\mathcal{R}(i_j)}[|L_{v_{i_j},\text{new}}|\ |r(v_0)\le\alpha] \le 10\cdot(\alpha\cdot\Delta)/(2^{j-1}).$$

This completes the proof of the first part of the lemma.

For the "Furthermore" part of the lemma, the analysis is similar as above. Now we start with the assumption that $r(v_0), r(w)\forall w\in N(v_0)$ are fixed. Note that $v_{i_1}\in N(v_0)$, which implies that $r(v_{i_1})$ is also fixed. We will then prove by induction on the index j that

$$\mathrm{E}_{\mathcal{R}(i_j)}[r(v_{i_j+1})] \le (r(v_{i_1}))/(2^{j-1}) \text{ and } \mathrm{E}_{\mathcal{R}(i_j)}[|L_{v_{i_j},\text{new}}|] \le (r(v_{i_1})\cdot\Delta)/(2^{j-2}).$$

In the case $j=1$, the above two inequalities hold as $r(v_{i_1+1}) \le r(v_{i_1})$ and $\mathrm{E}_{\mathcal{R}(i_1)}[|L_{v_{i_1},\text{new}}|] = r(v_{i_1})\cdot|N_{\text{new}}(v_{i_1})| \le r(v_{i_1})\cdot\Delta$. The inductive step from case $j-1$ to j can be then proven in the same way as we proved Inequalities (4). Then instead of assuming that

$r(v_0), r(w)\forall w \in N(v_0)$, we condition on the event that $r(v_0) \leq \alpha, r(w)\forall w \in N(v_0)$, which directly implies that $r(v_{i_1}) \leq \alpha$. Then it follows that $\mathrm{E}_{\mathcal{R}(i_j)}[r(v_{i_j+1})|r(v_0) \leq \alpha, r(w)\forall w \in N(v_0)] \leq \frac{\alpha}{2^{j-1}}$ and $\mathrm{E}_{\mathcal{R}(i_j)}[|L_{v_{i_j},\mathrm{new}}| \ |r(v_0) \leq \alpha, r(w)\forall w \in N(v_0)] \leq \frac{\alpha\cdot\Delta}{2^{j-2}}$. Finally, by the definition of good vertices, $|L_{v_{i_j},\mathrm{new}}| \geq \frac{1}{10}|L_{v_{i_j}}|$, which implies that $\mathrm{E}_{\mathcal{R}(i_j)}[|L_{v_{i_j}}| \ |r(v_0) \leq \alpha, r(w)\forall w \in N(v_0)] \leq 10 \cdot \mathrm{E}_{\mathcal{R}(i_j)}[|L_{v_{i_j},\mathrm{new}}| \ |r(v_0) \leq \alpha, r(w)\forall w \in N(v_0)] \leq \frac{10\alpha\cdot\Delta}{2^{j-2}}$. This completes the "Furthermore" part of the lemma. ◀

Now we relate the total work to the work incurred by Step 3a. Note that the total work T_v is proportional to the sum of sizes of all lower-ranked neighborhoods of $v_0, v_1, \ldots$. We will prove the following lemma, which implies that the total work of recoloring v is at most a constant factor of the total work for recoloring all the *good vertices* on the recoloring path.

▶ **Lemma 7.** *It holds that* $\sum_i |L_{v_i}| \leq 3\sum_{i\,:\,v_i \text{ is good}} |L_{v_i}| = 3\sum_j |L_{v_{i_j}}|$.

Proof. We first introduce the following definition. For any i and $k < i$, we let $\mathcal{F}(v_k, v_i)$ denote the set of vertices whose ranks are less than $r(v_i)$, and are sampled when we are exploring v_k, i.e., $\mathcal{F}(v_k, v_i) = \{w : w \in N_{\mathrm{new}}(v_k), r(w) < r(v_i)\}$. Note that as $r(v_{i+1}) < r(v_i)$, it always holds that for any $0 \leq k < i$, $\mathcal{F}(v_k, v_{i+1}) \subseteq \mathcal{F}(v_k, v_i)$. Now we define the following potential function Φ:

$$\Phi(-1) := 0 \text{ and } \ \Phi(i) := \sum_{k:k\leq i} |\mathcal{F}(v_k, v_{i+1})| \ \ \forall i \geq 0, \tag{5}$$

We have the following claim regarding the potential functions.

▷ Claim 8. For any $i \leq 0$, $\Phi(i) \geq 0$. Furthermore, if v_i is a good vertex, then $\Phi(i)-\Phi(i-1) \leq |L_{v_i}|/2$, otherwise $\Phi(i) - \Phi(i-1) \leq -7|L_{v_i}|/20$.

Proof. Note that if Step 3a in subroutine SetColor is executed at vertex v_i, i.e., v_i is good, then the potential $\Phi(i)$ might be larger or smaller than $\Phi(i-1)$. If v_i is good then $|\mathcal{F}(v_i, v_{1+i})| \leq \frac{|L^{<}_{v_i,\mathrm{new}}|}{2}$ by the fact that $r(v_{1+i})$ is at most the median rank in $L^{<}_{v_i,\mathrm{new}}$. Furthermore, it holds that

$$\begin{aligned}\Phi(i) = \sum_{k:k\leq i} |\mathcal{F}(v_k, v_{1+i})| \ &\leq \sum_{k:k\leq i-1} |\mathcal{F}(v_k, v_i)| + |\mathcal{F}(v_i, v_{i+1})| \\ &\leq \Phi(i-1) + |L^{<}_{v_i,\mathrm{new}}|/2 \leq \Phi(i-1) + |L_{v_i}|/2\end{aligned}$$

Now suppose that Step 3b is executed at vertex v_i, i.e., v_i is not good. Since v_{1+i} is a vertex from the lower half of the old lower neighbors of v_i (i.e., $v_{1+i} \in L^{<}_{v_i,\mathrm{old}} \subseteq \cup_{k<i}\mathcal{F}(v_k, v_i) \cap L_{v_i,\mathrm{old}}$), we have that to obtain the set $\cup_{k<i}\mathcal{F}(v_k, v_{1+i})$ from the set $\cup_{k<i}\mathcal{F}(v_k, v_i)$, we need to remove at least $\frac{1}{2}|L_{v_i,\mathrm{old}}| \geq \frac{1}{2}(1-\frac{1}{10})|L_{v_i}|$ vertices. Furthermore, $\mathcal{F}(v_i, v_{1+i})$ can contain at most $|L_{v_i,\mathrm{new}}| \leq \frac{1}{10}|L_{v_i}|$ vertices. This implies that

$$\begin{aligned}\Phi(i) &= \sum_{k:k\leq i} |\mathcal{F}(v_k, v_{1+i})| = \sum_{k:k\leq i-1} |\mathcal{F}(v_k, v_{1+i})| + |\mathcal{F}(v_i, v_{1+i})| \\ &\leq \sum_{k:k\leq i-1} |\mathcal{F}(v_k, v_i)| - \frac{1}{2}(1-\frac{1}{10})|L_{v_i}| + \frac{1}{10}|L_{v_i}| = \Phi(i-1) - \frac{7}{20}\cdot|L_{v_i}|\end{aligned}$$ ◁

Now we distinguish three types of indices. We call an index i, a *type I* index, if Step 3a occurred during SetColor(v) *and* the $\Phi(i) - \Phi(i-1) \geq 0$. By Claim 8 it holds that for such an index i, $|L_{v_i}| \geq 2(\Phi(i) - \Phi(i-1))$. We call i a *type II* index, if Step 3a occurred during SetColor(v) *and* the $\Phi(i) - \Phi(i-1) \leq 0$. It holds that for such an index i (as for any index), $|L_{v_i}| \geq 0$. We call i a *type III* index, if Step 3boccurred during SetColor(v), i.e. v_i is not a good vertex. By Claim 8 it holds that for such an index i, Φ decreases and

$$|L_{v_i}| \leq (\Phi(i-1) - \Phi(i)) \cdot \frac{20}{7} < 3 \cdot (\Phi(i-1) - \Phi(i)).$$

Now we bound the sum of sizes of lower-ranked neighborhoods of vertices corresponding to Step 3b. It holds that

$$\begin{aligned}\sum_{i:\text{ Step 3b}} |L_{v_i}| \leq \sum_{i:\text{ type III}} 3(\Phi(i-1)-\Phi(i)) &\leq \sum_{i:\text{ type II or III}} 3(\Phi(i-1)-\Phi(i)) \\ \leq \sum_{i:\text{ type I}} 3(\Phi(i)-\Phi(i-1)) &\leq \sum_{i:\text{ type I}} 3\cdot\frac{1}{2}|L_{v_i}| < \sum_{i:\text{ type I}} 2|L_{v_i}|\end{aligned}$$

where the third inequality follows from the fact that Φ starts at 0 and is non-negative at the end, and, thus, the total decrease of Φ is at most its total increase. Thus, it follows that

$$\sum_i |L_{v_i}| = \sum_{i:\text{ type I or II}} |L_{v_i}| + \sum_{i:\text{ type III}} |L_{v_i}| \leq 3 \sum_{i:\text{ type I or II}} |L_{v_i}| = 3\sum_j |L_{v_{i_j}}| \qquad \blacktriangleleft$$

Now we finish the proof of Lemma 4. By Lemma 7 and Lemma 6, it holds that

$$\mathrm{E}[\sum_i |L_{v_i}| \mid r(v)\leq\alpha] \leq 3\cdot\mathrm{E}[\sum_j |L_{v_{i_j}}| \mid r(v)\leq\alpha] = O(\alpha\cdot\Delta\cdot\sum_j \frac{1}{2^j}) = O(\alpha\Delta).$$

Since the expected work T_v satisfies that $T_v = O(\sum_i |L_{v_i}|)$, the first part of the lemma follows. By the "Furthermore" part of Lemma 6, it holds that

$$\begin{aligned}&\mathrm{E}[\sum_i |L_{v_i}| \,|\, r(v)\leq\alpha, r(w)\forall w\in N(v)] \\ \leq& 3\cdot|L_v| + 3\cdot\mathrm{E}[\sum_{j\geq 1} |L_{v_{i_j}}| \,|\, r(v)\leq\alpha, r(w)\forall w\in N(v)] \\ \leq& 3\cdot|L_v| + 3\cdot 10\cdot\alpha\cdot\Delta\cdot\sum_j \frac{1}{2^{j-2}} = 3\cdot|L_v| + O(\alpha\cdot\Delta\cdot\sum_j\frac{1}{2^j}) = O(|L_v|)+O(\alpha\Delta).\end{aligned}$$

Then the "Furthermore" part of Lemma 4 follows from the fact that $T_v = O(\sum_i |L_{v_i}|)$.

3 Lower Bound for Dynamic Δ-Colorability Testing: Proof of Theorem 2

In [24] Patrascu and Demaine construct an n-node graph and show that there exists a sequence $\mathcal{S}$ of T edge insertion, edge deletion, and query operations such that any data structure for dynamic connectivity must perform $\Omega(T\log n)$ cell probes to process the sequence, where each cell has size $O(\log n)$. This shows that the amortized number of cell probes per operation is $\Omega(\log n)$.

We now show how to use this result to get a lower bound for the dynamic Δ-colorability testing problem with $\Delta = 2$.

The graph G in the proof of [24] consists of a $\sqrt{n}\times\sqrt{n}$ grid, where each node in column 1 has exactly 1 edge to a node of column 2 and no other edges, each node in column i, with $1 < i < \sqrt{n}$ has exactly 1 edge to a node of column $i-1$ and 1 edge to a node of column $i+1$ and no other edges, and each node in column $\sqrt{n}$ has exactly 1 edge to a node of column $\sqrt{n}-1$ and no other edges. Thus, the graph consists of $\sqrt{n}$ paths of length $\sqrt{n}-1$ and the edges between column i and $i+1$ for any $1\leq i<\sqrt{n}$ represent a permutation of the $\sqrt{n}$ rows. The sequence $\mathcal{S}$ consists of "batches" of $O(\sqrt{n})$ edge updates, replacing the permutation of some column i by a new permutation for column i. Between the batches of updates are "batches" of connectivity queries, each consisting of $\sqrt{n}$ connectivity queries and a parameter $1\leq k\leq\sqrt{n}$, where the j-th query for $1\leq j\leq\sqrt{n}$ of each batch tests whether the j-th vertex of column 1 is connected with a specific vertex of column k.

Note that the maximum degree Δ is 2. We now show how to modify each connectivity query (u, v) such that it consists of a constant number of edge updates and one query whether the resulting graph is Δ-colorable. The answer will be *no* iff u and v are connected. Thus, in the resulting sequence $\mathcal{S}'$ the number of query operations equals the number of query operations in $\mathcal{S}$ and the number of update operations is linear in the number of update and query operations in $\mathcal{S}$. Thus the total number of operations in $\mathcal{S}'$ is only a constant factor larger than the number of operations in $\mathcal{S}$, which, together with the result of [24], implies that the amortized number of cell probes per operation is $\Omega(\log n)$.

We now show how to simulate a connectivity query(u, v), where u is in column 1 and v is in column k for some $1 \leq k\sqrt{n}$. We assume that k is even and explain below how to deal with the case that k is odd. The instance for the dynamic Δ-colorability testing consists of G with an additional node s added. To simulate a connectivity query(u, v) we (1) remove the edge from v to its neighbor in column $k+1$ if $k < \sqrt{n}$, (2) add the edges (u, s) and (v, s) and then (3) ask a Δ-colorability query. Note that the resulting graph still has maximum degree 2. Furthermore, if u and v are connected in G then there exists a unique path of odd length $k-1$ between them. Together with the edges (u, s) and (v, s) and the assumption that k is even, this results in an odd length cycle, so that the answer to the 2-colorability query is *no*. If, however, u and v are not connected in G, then adding the edges (u, s) and (v, s) creates a path of length $2 + \sqrt{n} - 1 + k - 1 = \sqrt{n} + k$, but no cycle. Thus, the 2-colorability query returns *yes*. Thus u and v are connected in G iff the 2-colorability query in the modified graph returns *no*. Afterwards we remove the edges (u, s) and (v, s). Finally if k is odd, we do not add a vertex s to G and to simulate the connectivity query(u, v) we simply insert the edge (u, v). As before there exists an odd length cycle in the graph iff u and v are connected. The rest of the proof remains unchanged.

This finishes the proof of Theorem 2.

▶ Remark 9. Let us recall Brooks' theorem [9]: every *connected* graph admits a Δ-coloring, except that it is an odd cycle or a complete graph. This implies that if the dynamic graph is guaranteed to be connected, then we can answer Δ-colorability in constant time for $\Delta \geq 3$ by checking if the graph is complete. However, since the graph is not necessarily connected, it is unclear if the query can be answered in constant time for $\Delta \geq 3$. In particular, testing whether a dynamic graph is connected or not requires $\Omega(\log n)$ time per operation [24].

4 Further Discussions

Initialization in $O(n)$ Time. Now we describe how we can reduce the initialization time from $O(n\Delta)$ to $O(n)$. Note that the only part that takes $O(n\Delta)$ time is to initialize $\mathcal{C}_u(\overline{H})$ for each vertex u, and the rest part of initialization already only takes $O(n)$ time. The main observation is that $\mathcal{C}_u(\overline{H})$ is only needed in the sampling subroutine of SetColor(u) and even there only once the degree of a vertex is at least $\Delta/2$. Since we make the standard assumption that we start with an empty graph, this means that $\Omega(\Delta)$ insertions incident to u must have happened. Thus, we build $\mathcal{C}_u(\overline{H})$ only once this is the case and amortize the cost of building it over these previous $\Omega(\Delta)$ insertions.

To be more precise, we change the initialization phase as follows: We do not build $\mathcal{C}_u(\overline{H})$ for any vertex u. Note that all other data structure are built as before, but they only have size $O(n)$ and only take time $O(n)$ to build.

When an edge (u, v) is inserted, we check whether one of the endpoints, say u, of the newly inserted edge reaches the degree $\Delta/2$ and does not yet have the data structure $\mathcal{C}_u(\overline{H})$. If so, we build $\mathcal{C}_u(\overline{H})$ and its hash table at this point in time $O(\Delta)$. We amortize this cost

over the $\Delta/2$ updates that increased the degree of u to $\Delta/2$, adding a constant amortized cost to each of them. (If the other endpoint v also reaches the degree $\Delta/2$, we handle it analogously.)

Note that this does not affect the SetColor algorithm: as long as the degree of a vertex u is less than $\Delta/2$, SetColor(u) selects a new color by sampling in Step 2 from $\mathcal{B}_u$. To do so $\mathcal{C}_u(\overline{H})$ is not needed: In time $O(|L_u|)$ time we build the lists and corresponding hash tables for $\mathcal{M}_u(L) \cup \mathcal{U}_u(L)$, which together with the maintained list and hash table for $\mathcal{C}_u(H)$ suffice for us to sample a color from $\mathcal{B}_u$ in $O(1)$ time: We pick a random color from $\mathcal{C}$ and test whether it belongs to $\mathcal{B}_u$ by making sure that it does not belong to $\mathcal{M}_u(L) \cup \mathcal{U}_u(L)$ or $\mathcal{C}_u(H)$. The fact that the degree of u is at most $\Delta/2$ implies that in expectation the second randomly chosen color will belong to $\mathcal{B}_u$.

Once $\mathcal{C}_u(\overline{H})$ and its hash table has been built, it is used in the way as we described before and updated as in Section 2.1.

Extension to Work for Changing Δ. As we mentioned, we can extend our algorithm to work with changing Δ. (A similar extension was also done in [5]). For any time stamp $t \geq 0$, we will maintain a global value $\Delta_t := \max_{j=1}^{t} \max_{v \in V} \deg_j(v)$, where $\deg_j(v)$ denotes the degree of v in the graph after j edge updates, that is, Δ is the maximum degree seen so far (till time t). Then we have a randomized algorithm for maintaining a $(\Delta_t + 1)$-coloring. More precisely, for any time stamp j, for each vertex v, we only need to guarantee that the color $\chi(v)$ is chosen from $\{1, \ldots, \deg_j(v) + 1\}$. Then for each vertex $v \in V$, we let $\mathcal{C}_v(\overline{H}) \subseteq \mathcal{C}$ consist of all the colors in $\{1, \ldots, \deg_j(v) + 1\}$ that have not been assigned to any neighbor u of v for $u \in H_v$. It is easy to see that Lemma 3, 4 and 5 still hold, and our randomized dynamic coloring algorithm maintains a proper $(\Delta_t + 1)$-coloring of the graph G_t at time t with constant amortized update time, for any $t \geq 0$.

Additionally we can keep a variable Δ such that we rebuild the data structure every Δn operations as follows: We determine the list of current edges and set Δ to be the maximum degree of the current graph. Then we build the data structure for an empty graph and insert all edges using the insert operation. This increases the running time by an amortized constant factor and guarantees that Δ is the maximum degree in the graph *within the last* Δn *updates.*

References

1. Luis Barba, Jean Cardinal, Matias Korman, Stefan Langerman, André van Renssen, Marcel Roeloffzen, and Sander Verdonschot. Dynamic graph coloring. In *Workshop on Algorithms and Data Structures*, pages 97–108. Springer, 2017.
2. Leonid Barenboim and Tzalik Maimon. Fully-dynamic graph algorithms with sublinear time inspired by distributed computing. *Procedia Computer Science*, 108:89–98, 2017.
3. Surender Baswana, Sumeet Khurana, and Soumojit Sarkar. Fully dynamic randomized algorithms for graph spanners. *ACM Transactions on Algorithms (TALG)*, 8(4):35, 2012.
4. Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully dynamic maximal independent set with polylogarithmic update time. In *IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS), 2019*. IEEE, 2019.
5. Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–20. SIAM, 2018.
6. Sayan Bhattacharya, Fabrizio Grandoni, Janardhan Kulkarni, Quanquan C. Liu, and Shay Solomon. Fully dynamic $(\delta + 1)$ coloring in constant update time. Private communication.

7 Sayan Bhattacharya and Janardhan Kulkarni. Deterministically maintaining a $(2+\varepsilon)$-approximate minimum vertex cover in $O(1/\varepsilon^2)$ amortized update time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1872–1885. SIAM, 2019.

8 Manuel Blum, Robert W Floyd, Vaughan Pratt, Ronald L Rivest, and Robert E Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.

9 R. L. Brooks. On colouring the nodes of a network. *Mathematical Proceedings of the Cambridge Philosophical Society*, 37(2):194–197, 1941.

10 Keren Censor-Hillel, Elad Haramaty, and Zohar Karnin. Optimal dynamic distributed mis. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 217–226. ACM, 2016.

11 Shiri Chechik and Tianyi Zhang. Fully dynamic maximal independent set in expected poly-log update time. In *IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS), 2019*. IEEE, 2019.

12 Martin Dietzfelbinger, Anna R. Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert, and Robert Endre Tarjan. Dynamic perfect hashing: Upper and lower bounds. *SIAM J. Comput.*, 23(4):738–761, 1994.

13 Ran Duan, Haoqing He, and Tianyi Zhang. Dynamic edge coloring with improved approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1937–1945. SIAM, 2019.

14 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997.

15 Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees (preliminary version). In *STOC*, pages 252–257, 1983.

16 Monika R Henzinger and Valerie King. Maintaining minimum spanning trees in dynamic graphs. In *International Colloquium on Automata, Languages, and Programming*, pages 594–604. Springer, 1997.

17 Monika Rauch Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999. doi:10.1145/320211.320215.

18 Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM (JACM)*, 48(4):723–760, 2001.

19 Jacob Holm, Eva Rotenberg, and Christian Wulff-Nilsen. Faster fully-dynamic minimum spanning forest. In *Algorithms-ESA 2015*, pages 742–753. Springer, 2015.

20 Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *SODA*, pages 1131–1141, 2013.

21 Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge university press, 2005.

22 Danupon Nanongkai and Thatchaphol Saranurak. Dynamic spanning forest with worst-case update time: adaptive, Las Vegas, and $O(n^{1/2-\varepsilon})$-time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1122–1129. ACM, 2017.

23 Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on*, pages 950–961. IEEE, 2017.

24 Mihai Patrascu and Erik D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM J. Comput.*, 35(4):932–963, 2006. doi:10.1137/S0097539705447256.

25 Shay Solomon. Fully dynamic maximal matching in constant update time. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 325–334. IEEE, 2016.

26 Shay Solomon and Nicole Wein. Improved dynamic graph coloring. In *26th Annual European Symposium on Algorithms*, 2018.

27 Dominic JA Welsh and Martin B Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1):85–86, 1967.

28 Christian Wulff-Nilsen. Fully-dynamic minimum spanning forest with improved worst-case update time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1130–1143. ACM, 2017.

Cryptocurrency Mining Games with Economic Discount and Decreasing Rewards

Marcelo Arenas
PUC Chile & IMFD Chile, Santigo, Chile
marenas@ing.puc.cl

Juan Reutter
PUC Chile & IMFD Chile, Santigo, Chile
jreutter@ing.puc.cl

Etienne Toussaint
University of Edinburgh, Edinburgh, UK
etienne.toussaint@ed.ac.uk

Martín Ugarte
PUC Chile & IMFD Chile, Santigo, Chile
martin@martinugarte.com

Francisco Vial
ProtonMail & IMFD Chile, Santiago, Chile
fvial@pm.me

Domagoj Vrgoč
PUC Chile & IMFD Chile, Santigo, Chile
dvrgoc@ing.puc.cl

Abstract

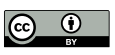
In the consensus protocols used in most cryptocurrencies, participants called *miners* must find *valid blocks* of transactions and append them to a shared tree-like data structure. Ideally, the rules of the protocol should ensure that miners maximize their gains if they follow a default strategy, which consists on appending blocks only to the longest branch of the tree, called the *blockchain*. Our goal is to understand under which circumstances are miners encouraged to follow the default strategy. Unfortunately, most of the existing models work with simplified payoff functions, without considering the possibility that rewards decrease over time because of the game rules (like in Bitcoin), nor integrating the fact that a miner naturally prefers to be paid earlier than later (the economic concept of discount). In order to integrate these factors, we consider a more general model where issues such as economic discount and decreasing rewards can be set as parameters of an infinite stochastic game. In this model, we study the limit situation in which a miner does not receive a full reward for a block if it stops being in the blockchain. We show that if rewards are not decreasing, then miners do not have incentives to create new branches, no matter how high their computational power is. On the other hand, when working with decreasing rewards similar to those in Bitcoin, we show that miners have an incentive to create such branches. Nevertheless, this incentive only occurs when a miner controls a proportion of the computational power which is close to half of the computational power of the entire network.

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

Keywords and phrases cryptocurrency, game theory, cryptomining, economic discount, decreasing rewards

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.54

Funding Millennium Institute for Foundational Research on Data, Chile



37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020).
Editors: Christophe Paul and Markus Bläser; Article No. 54; pp. 54:1–54:16

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

1 Introduction

The Bitcoin Protocol [14, 15, 16], or Nakamoto Protocol, introduces a novel decentralized network-consensus mechanism that is trustless and open for anyone connected to the Internet. This open and dynamic topology is supported by means of an underlying currency (a so-called *cryptocurrency* [16]), to encourage/discourage participants to/from taking certain actions. The largest network running this protocol at the time of writing is the Bitcoin network, and its underlying cryptocurrency is Bitcoin (BTC). The success of Bitcoin lead the way for several other cryptocurrencies; some of them are replicas of Bitcoin with slight modifications (e.g. Litecoin [27] or Bitcoin Cash [25]), while others introduce more involved modifications (e.g. Ethereum [26, 22] or Monero [28]).

The data structure used in these protocols is an append-only record of transactions, which are assembled into *blocks*, and appended to the record once they are marked as valid. The incentive to generate valid new blocks is an amount of currency, which is known as the *block reward*. In order to give value to the currencies, the proof-of-work framework mandates that participants generating new blocks are required to solve some computationally hard problem per each new block. This is known as *mining*, and the number of potential solutions that a miner can generate per second is referred to as her *hash power* (or *computational power*). Agents who participate in the generation of blocks are called *miners*. In Bitcoin, for example, the hard problem corresponds to finding blocks with a *header* whose hash value, when interpreted as a number, is less than a certain threshold. Since hash functions are pseudo-random, the only way to generate a valid block is to try with several different blocks, until one of them has a header with a hash value below the established threshold.

Miners are not told where to append the new blocks they produce. The only requirement is that new blocks must include a pointer to a previous block in the data structure, which then naturally forms a tree of blocks. The consensus data structure is generally defined as the longest branch of such a tree, also known as the *blockchain*. In terms of cryptocurrencies, this means that the only valid currency should be the one that originates from a transaction contained in a block of the blockchain. Miners looking to maximise their rewards may then attempt to create new branches out of the blockchain, to produce a longer branch that contains more of their blocks (and earn more block rewards) or to produce a branch that contains less blocks of a user they are trying to harm. This opens up several interesting questions: under what circumstances are miners encouraged to produce a new branch in the blockchain? What is the optimal strategy of miners assuming they have a rational behaviour? Finally, how can we design new protocols where miners do not have incentives to deviate from the main branch?

Our goal is to provide a model of mining that can incorporate different types of block rewards (including the decreasing rewards used in e.g. Bitcoin, where rewards for block decrease after a certain amount of time), as well as the economic concept of *discount*, i.e. the fact that miners prefer to be rewarded sooner than later, and that can help in answering the previous questions. Since mining protocols vary with each cryptocurrency, distilling a clean model that can answer these questions while simultaneously covering all practical nuances of currencies is far from trivial [10]. Instead, we abstract from these rules and focus on the limit situation in which a miner does not receive the full reward for a block if it stops being in the blockchain. More precisely, the reward for a block b is divided into an infinite number of payments, and the miner loses some of them whenever b does not belong to the blockchain. This limit situation represents miners with a strong incentive to put–and maintain–their blocks in the blockchain, and is relevant when studying cryptocurrencies as a closed system,

where miners do not wish to spend money right away but rather be able to cash-out their wealth at any point in time. In terms of how mining is performed, we consider these two simple rules: each player i is associated a fixed value h_i specifying her proportion of the hash power against the total hash power, and she tries in each step to append a new block somewhere in the tree of blocks, being h_i her probability of succeeding.

The last two rules mentioned above are the standard way of formalizing mining in a cryptocurrency. On the other hand, the way a miner is rewarded for a block in our model takes us on a different path from most of current literature, wherein agents typically mine with the objective of cashing-out as soon as possible or after an amount of time chosen a priori [10, 2]. Far from being orthogonal, our framework is complementary with these studies, as it allows to validate some of the assumptions and results obtained in these articles with miners who have stronger motives to mine and keep their blocks in the blockchain.

Contributions. Our first contribution is a model for mining, given as an infinite stochastic game in which maximising the utility corresponds to both putting blocks in the blockchain and maintaining them there for as long as possible. A benefit of our model is that using few basic design parameters we can accommodate different cryptocurrencies, and not focus solely on Bitcoin, while also allowing us to account for fundamental factors such as so-called *deflation*, or discount in the block reward. The second contribution of our work is a set of results about strategies in two different scenarios. First, we study mining under the assumption that block rewards are constant (as it will eventually be in cryptocurrencies with tail-emission such as Monero or Ethereum), and secondly, assuming that per-block reward decreases over time (a continuous approximation to Bitcoin rewards).

In the first scenario of constant rewards, we show that the default strategy of always mining on the latest block of the blockchain is indeed a Nash equilibrium and, in fact, provides the highest possible utility for all players. Therefore with constant reward, we prove that long forks should not happen, as it is not an optimal strategy. On the other hand, if block reward decreases over time, we prove that strategies that involve *forking* the blockchain can be a better option than the default strategy, and thus we study what is the best strategy for miners when assuming everyone else is playing the default strategy. We provide different strategies that involve branching at certain points of the blockchain, and show how to compute their utility. When we analyse which one of these strategies is the best, we see that the choice depends on the hash power, the rate at which block rewards decrease over time, and the usual financial discount rate. We confirm the commonly held belief that players should start deviating from the default strategy when they approach 50% of the network's hash power (also known as 51% attack), but we go further: there are more complex strategies that prove better than default even with less than 50% of the hash power. Further investigation is needed but these results complement and improve our current understanding of mining strategies and tend to show that even with decreasing reward long forks should not happen if no miner is holding close to 50% of the hash power, therefore validate the assumption used in most previous works (see e.g. [10, 2]).

Related work. Our framework takes us on a different path that most of current literature offering a game-theoretic characterisation for mining [10, 2, 11], which typically model the reward of players as the proportion of their blocks with respect to the total number of blocks (we pay for each block). Each choice has its own benefits; our choice allows us to analyse different forms of rewards and also introduce a discount factor on the utility, which we view as one of the main advantages of our model. It is also common to introduce assumptions

that limit the set of strategies. For instance, Kiayias et.al. [10] assume that only one block per depth generates reward, which is natural in their framework but limits the set of valid strategies they consider. Moreover, Biais et.al.[2] assume that the reward of a block depends on the proportion of hash-power dedicated to blockchains containing it at a time chosen a priori. These assumptions do not take into account every potential forking strategies, or the fact that a miner may want to adapt his cash-out strategy based on the situation. Lastly, our framework cannot deal with strategies that feature a tactical release of blocks often referred as selfish mining, in which miners opt not to release new blocks in hope that these will give them a future advantage [19, 6, 8, 18, 17]. Our model can be extended to account for most of those strategies, for example by defining states as a tuple of trees, one for each player. However work studying precise problems and taking into account the intrinsic cost of mining like electricity [23, 2] cannot easily be added to our framework, because it requires a continuous time-based model for mining.

Among other works that approach cryptocurrency mining from a game-theoretical point of view, we mention [12, 3], noting that these differ from our work either in the choice of a reward function, the space of mining strategies considered, or both. As far as we are aware, our work is the first to provide a model that can account for multiple choices in the reward function (say, constant reward or decreasing reward), and without any assumption on the set of strategies. Recently, the perks of adding new functionalities to bitcoin's mining protocol have been studied: In [11], it is shown that a pay-forward option would ensure optimality of the default behaviour, even when miner rewards are mainly given as transaction fees. There is also interesting work regarding mining strategies in multi-cryptocurrency markets [5, 20], and a number of articles on network properties of the Bitcoin protocol, as well as technical considerations regarding its security and privacy (see e.g. the survey by Conti et al. [4]). Interestingly, some network settings can inflict undesired mining behaviour [1, 9, 24].

Proviso. Due to the lack of space, some proofs are deferred to the full version.

2 A Game-theoretic Formalisation of Crypto-Mining

The mining game is played by a set $\mathbf{P} = \{0, 1, , \ldots, m-1\}$ of players, with $m \geq 2$. In this game, each player gains some reward depending on the number of blocks she owns. Every block must point to a previous block, except for the first block which is called the *genesis block*. Thus, the game defines a tree of blocks. Each block is put by one player, called the *owner* of this block. Each such tree is called a *state of the game*, or just *state*, and represents the knowledge that each player has about the blocks that have been mined thus far.

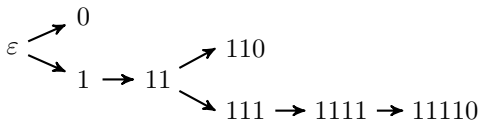
The key question for each player is, then, where do I put my next block? The general rule in cryptocurrencies is that miners are only allowed to spend their reward if their blocks belongs to the *blockchain*, which in this paper is simply the longest chain of blocks in the current state (the model is general enough to consider other forms of blockchain such as Ethereum's notion, but some of the results may change with this other definition). Thus, players face essentially two possibilities: put their blocks right after the end of the blockchain, or try to *fork*, betting that a smaller chain will eventually become the blockchain. As the likelihood of mining the next block is directly related to the comparative hash power of a player, we model mining as an infinite stochastic game, in which the probability of executing the action of a player p is given by her comparative hash power.

In what follows we define the components of the game considered in this paper. Our formalisation is similar to others in the literature [10, 11], except for the way in which miners are rewarded and the way in which these rewards are accumulated in the utility function. As these elements are fundamental for our model, we analyse them in detail in Section 2.1.

Blocks, states and the notion of blockchain. In a game played by m players, a block is defined as a string b over the alphabet $\{0, 1, \ldots, m-1\}$. We denote by $\mathbf{B}$ the set of all blocks, that is, $\mathbf{B} = \{0, 1, \ldots, m-1\}^*$. Each block apart from ε has a unique owner, defined by the function owner : $(\mathbf{B} \setminus \{\varepsilon\}) \to \{0, 1, \ldots, m-1\}$ such that owner(b) is equal to the last symbol of b. As in [10], a state of the game is defined as a tree of blocks. More precisely, a state of the game, or just state, is a finite and nonempty set of blocks $q \subseteq \mathbf{B}$ that is prefix-closed. That is, q is a set of strings over the alphabet $\{0, 1, \ldots, m-1\}$ such that if $b \in q$, then every prefix of b (including the empty word ε) also belongs to q. Note that a prefix closed subset of $\mathbf{B}$ uniquely defines a tree with ε as the root. The intuition here is that each element of q corresponds to a block that was put into the state q by some player. The genesis block corresponds to ε. When a player p decides to mine on top of a block b, she puts another block into the state defined by the string $b \cdot p$, where we use notation $b_1 \cdot b_2$ for the concatenation of two strings b_1 and b_2. Notice that with this terminology, given $b_1, b_2 \in q$, we have that b_2 is a descendant of b_1 in q if b_1 is a prefix of b_2, which is denoted by $b_1 \preceq b_2$. Moreover, a path in q is a nonempty set π of blocks from q for which there exist blocks b_1, b_2 such that $\pi = \{b \mid b_1 \preceq b \text{ and } b \preceq b_2\}$; in particular, b_2 is a descendant of b_1 and π is said to be a path from b_1 to b_2. Finally, let $\mathbf{Q}$ be the set of all possible states in a game played by m players, and for a state $q \in \mathbf{Q}$, let $|q|$ be its size, measured as the cardinality of the set q of strings (or blocks).

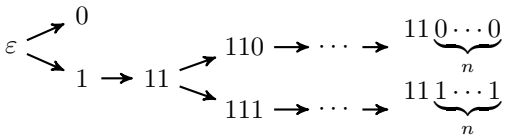
The *blockchain* of a state q, denoted by bc(q), is the path π in q of largest length, in the case this path is unique. If two or more paths in q are tied for the longest, then we say that the blockchain in q does not exist, and we assume that bc(q) is not defined (so that bc$(\cdot)$ is a partial function).

▶ **Example 2.1.** Consider the following state q of the game with players $\mathbf{P} = \{0, 1\}$:



In this case, we have that $q = \{\varepsilon, 0, 1, 11, 110, 111, 1111, 11110\}$, so q is a finite and prefix-closed subset of $\mathbf{B} = \{0, 1\}^*$. The owner of each block $b \in q \setminus \{\varepsilon\}$ is given by the the last symbol of b; for instance, we have that owner$(11) = 1$ and owner$(11110) = 0$. Moreover, the longest path in q is $\pi = \{\varepsilon, 1, 11, 111, 1111, 11110\}$, so that the blockchain of q is π (in symbols, bc$(q) = \pi$). Finally, $|q| = 8$, as q is a set consisting of eight blocks (including the genesis block ε).

Assume now that q' is the following state of the game:



We have that bc(q') is not defined since the paths $\pi_1 = \{\varepsilon, 1, 11, 110, \cdots, 110^n\}$ and $\pi_2 = \{\varepsilon, 1, 11, 111, \cdots, 111^n\}$ are tied for the longest path in q'.

Actions of a miner. On each step, each miner chooses a block in the current state, and attempts to mine on top of this block. Thus, in each step, each of the players race to place the next block in the state, and only one of them succeeds. The probability of succeeding is directly related to the comparative amount of hash power available to this player, the more hash power the more likely it is that she will mine the next block. Once a player places a block, this block is added to the current state, obtaining a different state from which the game continues.

Let $p \in \mathbf{P}$. Given a block $b \in \mathbf{B}$ and a state $q \in \mathbf{Q}$, we denote by $\text{mine}(p, b, q)$ an action in the mining game in which player p decides to mine on top of block b. Such an action $\text{mine}(p, b, q)$ is considered to be valid if $b \in q$ and $b \cdot p \notin q$. The set of valid actions for player p is collected in the set:

$$\mathbf{A}_p \quad = \quad \{\text{mine}(p, b, q) \mid b \in \mathbf{B}, q \in \mathbf{Q} \text{ and } \text{mine}(p, b, q) \text{ is a valid action}\}.$$

Moreover, given $a \in \mathbf{A}_p$ with $a = \text{mine}(p, b, q)$, the result of applying a to q, denoted by $a(q)$, is defined as the state $q \cup \{b \cdot p\}$. Finally, we denote by $\mathbf{A}$ the set of combined actions for the m players, that is, $\mathbf{A} = \mathbf{A}_0 \times \mathbf{A}_1 \times \cdots \times \mathbf{A}_{m-1}$.

Miner's Pay-off. Most cryptocurrencies follow these rules for miner's payment: (1) Miners receive a possibly delayed one-time reward per each block they mine. (2) The only blocks that are valid are those in the blockchain; if a block is not in the blockchain then the reward given for mining this block cannot be spent.

The second rule enforces that we cannot just give miners the full block reward when they put a block at the top of the current blockchain or after a delay, because blocks out of the blockchain may eventually give the same reward as valid ones. To illustrate this, consider the state q' in Example 2.1, where we have two paths (π_1 and π_2) competing to be the blockchain, and consider π_2 to be the latest path to be mined upon to reach q'. If player 0 had already been fully paid for any blocks 110^i, where $i \leq n$, then if π_2 wins the race and becomes the blockchain, such block 110^i would not be part of the blockchain anymore, but still would have given the full reward to player 0. To the best of our knowledge other attempts to formalize mining, especially bitcoin's mining, partially emancipate from this rule: only the first block to be confirmed will be paid, artificially nullifying the incentive to engage in long races.

In the following sections we will show how different reward functions can be used to understand different mining scenarios that arise in different cryptocurrencies. For now we assume, for each player $p \in \mathbf{P}$, the existence of a reward function $r_p : \mathbf{Q} \to \mathbb{R}$ such that the reward of p in a state q is given by $r_p(q)$. Moreover, the combined reward function of the game is $\mathbf{R} = (r_0, r_1, \ldots, r_{m-1})$. In Section 2.1 we provide a detailed explanation of how our pay-off model can be used to pay for blocks and at the same time to ensure that players try to maintain their blocks in the blockchain.

Transition probability function. As a last component of the game, we assume that $\mathbf{Pr} : \mathbf{Q} \times \mathbf{A} \times \mathbf{Q} \to [0, 1]$ is a transition probability function satisfying that for every state $q \in \mathbf{Q}$ and combined action $\mathbf{a} = (a_0, a_1, \ldots, a_{m-1})$ in $\mathbf{A}$, we have that $\sum_{p=0}^{m-1} \mathbf{Pr}(q, \mathbf{a}, a_p(q)) = 1$.

Notice that if p_1 and p_2 are two different players, then for every action $a_1 \in \mathbf{A}_{p_1}$, every action $a_2 \in \mathbf{A}_{p_2}$ and every state $q \in \mathbf{Q}$, it holds that $a_1(q) \neq a_2(q)$. Thus, we can think of $\mathbf{Pr}(q, \mathbf{a}, a_p(q))$ as the probability that player p places the next block, which will generate the state $a_p(q)$. As we have mentioned, such a probability is directly related to the hash power of player p, the more hash power the more likely it is that action a_p is executed and

p mines the next block before the rest of the players. In what follows, we assume that the hash power of each player does not change during the mining game, which is captured by the following condition: for each player $p \in \mathbf{P}$, we have that $\mathbf{Pr}(q, \mathbf{a}, a_p(q)) = h_p$ for every $q \in \mathbf{Q}$ and $\mathbf{a} \in \mathbf{A}$ with $\mathbf{a} = (a_0, a_1, \ldots, a_{m-1})$. We refer to such a fixed value h_p as the hash power of player p. Moreover, we assume that $h_p > 0$ for every player $p \in \mathbf{P}$, as if this is not the case then p can be removed from the game.

The mining game: definition, strategy and utility. Putting together all the components, a mining game is a tuple $(\mathbf{P}, \mathbf{Q}, \mathbf{A}, \mathbf{R}, \mathbf{Pr})$, where $\mathbf{P}$ is the set of players, $\mathbf{Q}$ is the set of states, $\mathbf{A}$ is the set of combined actions, $\mathbf{R}$ is the combined pay-off function and $\mathbf{Pr}$ is the transition probability function.

A strategy for a player $p \in \mathbf{P}$ is a function $s : \mathbf{Q} \to \mathbf{A}_p$. We define $\mathbf{S}_p$ as the set of all strategies for player p, and $\mathbf{S} = \mathbf{S}_0 \times \mathbf{S}_1 \times \cdots \times \mathbf{S}_{m-1}$ as the set of combined strategies for the game (recall that $\mathbf{P} = \{0, \ldots, m-1\}$ is the set of players). To define the notions of utility and equilibrium, we need some additional notation. Let $\mathbf{s} = (s_0, \ldots, s_{m-1})$ be a combined strategy. Given $q \in \mathbf{Q}$, define $\mathbf{s}(q)$ as the combined action $(s_0(q), \ldots, s_{m-1}(q))$. Moreover, given an initial state $q_0 \in \mathbf{Q}$, define the probability of reaching state $q \in \mathbf{Q}$, denoted by $\mathbf{Pr}^{\mathbf{s}}(q \mid q_0)$, as 0 if $q_0 \not\subseteq q$ (that is, if q is not reachable from q_0), and otherwise it is recursively defined following Bayes' rule: if $q = q_0$, then $\mathbf{Pr}^{\mathbf{s}}(q \mid q_0) = 1$; otherwise, we have that $|q| - |q_0| = k$, with $k \geq 1$, and

$$\mathbf{Pr}^{\mathbf{s}}(q \mid q_0) = \sum_{\substack{q' \in \mathbf{Q}: \\ q_0 \subseteq q' \text{ and } |q'| - |q_0| = k-1}} \mathbf{Pr}^{\mathbf{s}}(q' \mid q_0) \cdot \mathbf{Pr}(q', \mathbf{s}(q'), q).$$

In this definition, if for a player p we have that $s_p(q') = a$ and $a(q') = q$, then $\mathbf{Pr}(q', \mathbf{s}(q'), q) = h_p$. Otherwise, we have that $\mathbf{Pr}(q', \mathbf{s}(q'), q) = 0$ (this is well defined since there can be at most one player p whose action in the state q' leads us to the state q). For readability we write $\mathbf{Pr}^{\mathbf{s}}(q)$ instead of $\mathbf{Pr}^{\mathbf{s}}(q \mid \{\varepsilon\})$ to denote the probability of reaching state q from the intial state $\{\varepsilon\}$ that contains only the genesis block ε. The framework just described corresponds to a Markov Decision Process [13], but we do not explore this connection in this paper because we are not interested in the steady distributions of these processes.

Finally, we define the utility of players given a particular strategy. As is common when looking at personal utilities, we define it as the summation of the expected rewards, where future rewards are discounted by a factor of $\beta \in (0, 1)$ which is used to model the fact that money in the present is worth more than money in the future.

▶ **Definition 2.1.** *The β–discounted utility of a player p for a strategy $\mathbf{s}$ from a state q_0 in the mining game, denoted by $u_p(\mathbf{s} \mid q_0)$, is defined as:*

$$u_p(\mathbf{s} \mid q_0) \quad = \quad (1 - \beta) \cdot \sum_{q \in \mathbf{Q} \,:\, q_0 \subseteq q} \beta^{|q| - |q_0|} \cdot r_p(q) \cdot \mathbf{Pr}^{\mathbf{s}}(q \mid q_0).$$

Notice that the value $u_p(\mathbf{s} \mid q_0)$ may not be defined if this series diverges. To avoid this problem, from now on we assume that for every pay-off function $\mathbf{R} = (r_0, \ldots, r_{m-1})$, there exists a polynomial P such that $|r_p(q)| \leq P(|q|)$ for every player $p \in \mathbf{P}$ and state $q \in \mathbf{Q}$. Under this simple yet general condition, which is satisfied by the pay-off functions considered in this paper and in other game-theoretical formalisation's of Bitcoin mining [10], we can show that $u_p(\mathbf{s} \mid q_0)$ is a real number. Moreover, as for the definition of the probability of reaching a state from the initial state $\{\varepsilon\}$, we use notation $u_p(\mathbf{s})$ for the β–discounted utility of player p for the strategy $\mathbf{s}$ from $\{\varepsilon\}$, instead of $u_p(\mathbf{s} \mid \{\varepsilon\})$.

2.1 On the pay-off and utility of a miner

As mentioned earlier, we design our pay-off model with the goal of incentivising players to mine on the blockchain, and to keep their blocks in the blockchain. In this sense, the payment of a miner for a block b should be proportional to the amount of time b has been in the blockchain; in particular, the miner should be penalised if b ceases to be in the blockchain, and this penalty should decrease with time. In what follows, we explain how our pay-off model meets this goal.

Given a player p and a state q, for every block $b \in q$ assume that the reward obtained by p for the block b in q is given by $r_p(b, q)$, so that $r_p(q) = \sum_{b \in q} r_p(b, q)$. This decomposition can be done in a natural and straightforward way for the pay-off functions considered in this paper and in other game-theoretical formalisations of cryptomining [10, 11]. The reward for a mined block b is not granted immediately according to Definition 2.1, instead, a portion of $r_p(b, q)$ is paid in each state q where b is in. In other words, if a miner owns a block, then she will be rewarded for this block in every state where this block is part of the blockchain, in which case $r_p(b, q) > 0$.

Hence, in our model, a miner is payed a portion of a block's reward each time it is included in the blockchain, and even though she gets payed infinitely many times for each block, the discount factor in the definition of utility ensures that there is no overpay. In other words, when a player mines a new block, she will receive the full amount for this block only if she manages to maintain the block in the blockchain up to infinity. Otherwise, if the block ceases to be in the blockchain even for a short period, the miner receives only a fraction of the full amount. Formally, given a combined strategy $\mathbf{s}$, we can define the utility of a block b for a player p, denoted by $u_p^b(\mathbf{s})$, as follows:

$$u_p^b(\mathbf{s}) \quad = \quad (1-\beta) \cdot \sum_{q \in \mathbf{Q} \,:\, b \in \mathrm{bc}(q)} \beta^{|q|-1} \cdot r_p(q, b) \cdot \mathbf{Pr}^{\mathbf{s}}(q).$$

For simplicity, here we assume that the game starts in the genesis block ε, and not in an arbitrary state q_0. The discount factor in this case is $\beta^{|q|-1}$, since $|\{\varepsilon\}| = 1$.

To see that we pay the correct amount for each block, assume that there is a maximum value for the reward of a block b for player p, which is denoted by $M_p(b)$. Thus, we have that there exists $q_1 \in \mathbf{Q}$ such that $b \in q_1$ and $M_p(b) = r_p(b, q_1)$, and for every $q_2 \in \mathbf{Q}$ such that $b \in q_2$, it holds that $r_p(b, q_2) \leq M_p(b)$. Again, such an assumption is satisfied by most currently circulating cryptocurrencies, by the pay-off functions considered in this paper, and by other game-theoretical formalisations of cryptomining [10, 11]. Then we have that:

▶ **Proposition 2.2.** *For every player $p \in \mathbf{P}$, block $b \in \mathbf{B}$ and combined strategy $\mathbf{s} \in \mathbf{S}$, it holds that:* $u_p^b(\mathbf{s}) \leq \beta^{|b|} \cdot M_p(b)$.

Thus, the utility obtained by player p for a block b is at most $\beta^{|b|} \cdot M_p(b)$, that is, the maximum reward that she can obtained for the block b in a state multiplied by the discount factor $\beta^{|b|}$, where $|b|$ is the minimum number of steps that has to be performed to reach a state containing b from the initial state $\{\varepsilon\}$. Moreover, a miner can only aspire to get the maximum utility for a block b if once b is included in the blockchain, it stays in the blockchain in every future state. This tells us that our framework puts a strong incentive for each player in maintaining her blocks in the blockchain.

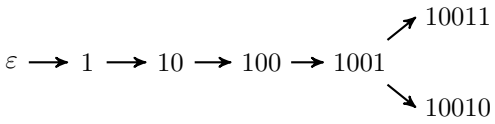


Figure 1 Although two paths are competing to become the blockchain, the blocks up to 1001 will contribute to the reward in both paths.

3 Equilibria with constant reward

The first version of the game we analyse is when the reward function $r_p(q)$ pays each block in the blockchain the same amount c. This is important for understanding what happens when currencies such as Ethereum or Monero switch to tail-emission, changing from a decreased reward scheme to a constant reward scheme. Further, it also helps to establish the main techniques used in this paper.

3.1 Defining constant reward

When considering the constant reward c for each block, $r_p(q)$ will equal c times the number of blocks owned by p in the blockchain $\mathrm{bc}(q)$ of q, when the latter is defined. On the other hand, when $\mathrm{bc}(q)$ is not defined it might seem tempting to simply define $r_p(q) = 0$. However, even if there is more than one longest path from the root of q to its leaves, it is often the case that all such paths share a common subpath (for instance, when two competing blocks are produced with a small time delay). While in this situation the blockchain is not defined, the miners know that they will at least be able to collect their reward on the portion of the state these two paths agree on. Figure 1 illustrates this situation.

Recall that a block b is a string over the alphabet $\mathbf{P}$, and we use notation $|b|$ for the length of b as a string. Moreover, given blocks b_1, b_2, we use $b_1 \preceq b_2$ to indicate that b_1 is a prefix of b_2 when considered as strings. Then we define:

$$\begin{aligned} \mathrm{longest}(q) &= \{b \in q \mid \text{for every } b' \in q : |b'| \leq |b|\} \\ \mathrm{meet}(q) &= \{b \in q \mid \text{for every } b' \in \mathrm{longest}(q) : b \preceq b'\}. \end{aligned}$$

Intuitively, $\mathrm{longest}(q)$ contains the leaves of all longest paths in the state q, and $\mathrm{meet}(q)$ is the path from the genesis block to the last block for which all these paths agree on. For instance, if q is the state from Figure 1, then we have that $\mathrm{longest}(q) = \{10011, 10010\}$, and $\mathrm{meet}(q) = \{\varepsilon, 1, 10, 100, 1001\}$. Notice that $\mathrm{meet}(q)$ is well defined as $\preceq$ is a linear order on the finite and non-empty set $\{b \in q \mid \text{for every } b' \in \mathrm{longest}(q) : b \preceq b'\}$. Also notice that $\mathrm{meet}(q) = \mathrm{bc}(q)$, whenever $\mathrm{bc}(q)$ is defined.

The reward function we consider in this section, which is called **constant reward**, is then defined for a player p as follows :

$$r_p(q) = c \cdot \sum_{b \in \mathrm{meet}(q)} \chi_p(b),$$

where c is a positive real number, $\chi_p(b) = 1$ if $\mathrm{owner}(b) = p$, and $\chi(p) = 0$ otherwise. Notice that this function is well defined since $\mathrm{meet}(q)$ always exists. Moreover, if q has a blockchain, then we have that $\mathrm{meet}(q) = \mathrm{bc}(q)$ and, hence, the reward function is defined for the blockchain of q.

3.2 The default strategy maximizes the utility

Let us start with analysing the simplest strategy, which we call the *default* strategy: regardless of what everyone else does, keep mining on the blockchain. More precisely, a player following the default strategy tries to mine upon the final block that appears in the blockchain of a state q. If the blockchain in q does not exist, meaning that there are at least two longest paths from the genesis block, then the player tries to mine on the final block of the path that maximizes her reward, which in the case of constant reward corresponds to the path containing the largest number of blocks belonging to her (if there is more than one of these paths, then between the final blocks of these paths she chooses the first according to a lexicographic order on the strings in $\{0, \ldots, m-1\}^*$). Notice that this is called the default strategy as it reflects the desired behaviour of the miners participating in the Bitcoin network. For a player p, let us denote this strategy by DF_p, and consider the combined strategy $\mathbf{DF} = (\mathrm{DF}_0, \mathrm{DF}_1, \ldots, \mathrm{DF}_{m-1})$.

We can easily calculate the utility of player p under $\mathbf{DF}$. Intuitively, a player p will receive a fraction h_p of the next block that is being placed in the blockchain, corresponding to her hash power. Therefore, at stage i of the mining game, the blockchain defined by the game will have i blocks, and the expected amount of blocks owned by the player p will be $h_p \cdot i$. The total utility for player p is then

$$u_p(\mathbf{DF}) \quad = \quad (1-\beta) \cdot h_p \cdot c \cdot \sum_{i=0}^{\infty} i \cdot \beta^i \quad = \quad h_p \cdot c \cdot \frac{\beta}{(1-\beta)}.$$

The question then is: can any player do better? As we show in the following theorem, the answer is no.

▶ **Theorem 3.1.** *Let p be a player, β be a discount factor in $(0,1)$ and u_p be the utility function defined in terms of β. Then for every combined strategy $\mathbf{s}$: $u_p(\mathbf{s}) \leq u_p(\mathbf{DF})$.*

The proof of this theorem relies on the fact that, under constant rewards, forking becomes less profitable because all blocks are worth the same amount of money, regardless of their position. This fact, combined with the economic discount, provides little incentives for players to sacrifice some time in order to fight for a longer blockchain: their reward is higher if instead of fighting they just keep mining on the blockchain.

A strategy $\mathbf{s}$ is a Nash equilibrium from a state q_0 in the mining game for m players if for every player $p \in \mathbf{P}$ and every strategy s for player p ($s \in \mathbf{S}_p$), it holds that $u_p(\mathbf{s} \mid q_0) \geq u_p((\mathbf{s}_{-p}, s) \mid q_0)$ (here as usual we use $(\mathbf{s}_{-p}, s)$ to denote the strategy $(s_0, s_1, \ldots s_{p-1}, s, s_{p+1}, \ldots, s_{m-1})$). As a corollary of Theorem 3.1, we obtain

▶ **Corollary 3.2.** *For every $\beta \in (0,1)$, the strategy $\mathbf{DF}$ is a Nash equilibrium.*

Hence, miners looking to maximise their wealth are better off with the default strategy. Especially this results prove that long forks should not happen and therefore validate the underlying assumption of other models [10]. Interestingly, previous work shows that under a setting in which miners are rewarded for the fraction of blocks they own against the total number of blocks, and no financial discount is assumed, then default strategy may not be an optimal strategy [10]. This suggests that miner's behaviour can really deviate depending on what are their short and long term goals, and we believe this is an interesting direction for future work.

4 Decreasing Reward

Miner's fees in many cryptocurrencies, including Bitcoin and Monero, are not constant, but decrease over time. We model such fees as a constant factor $\alpha \in [0, 1]$ that is lowered after every new block in the blockchain. That is, we use the following reward function r_p for all players $p \in \mathbf{P}$, denoted as the α**-discounted reward**:

$$r_p(q) \quad = \quad c \cdot \sum_{b \in \text{meet}(q)} \alpha^{|b|} \cdot \chi_p(b).$$

In this section, we show that forking can be a good strategy when miner's fees decrease over time. Not only we confirm the folklore fact that it is profitable to fork with more than half of the hash power, but our exploration gives us a concrete strategy that beats the default with less than half of the hash power.

4.1 When is forking a good strategy?

To understand when forking is a viable option, we consider a scenario when one of our m players decides to deviate from the default strategy, while the remaining players all follow the default strategy. In this case we can reduce the m player game to a two player game, where all the players following the default strategy are represented by a single player with the combined hash power of all these players. Therefore in this section we will consider that the mining game is played by two players 0 and 1, where 0 represents the miners behaving according to the default strategy, and 1 the miner trying to determine whether forking is economically more viable than mining on the existing blockchain. We always assume that player 1 has hash power h, while player 0 has hash power $1 - h$.

Let us first show the utility for player 1 when she uses the default strategy $\mathbf{DF} = (\text{DF}_0, \text{DF}_1)$.

▶ **Lemma 4.1.** *If h is the hash power of player 1, then*

$$u_1(\mathbf{DF}) \quad = \quad h \cdot c \cdot \frac{\alpha \cdot \beta}{(1 - \alpha \cdot \beta)}.$$

As in the case of constant reward, this corresponds to h times the utility of winning all the blocks in the single blockchain generated by the default strategy.

Now suppose that player 1 deviates from the default strategy, and considers a strategy based on forking the blockchain once player 0 mines a block. How would this new strategy look? In this section we consider the strategy AF (for *always fork*), where player 1 forks as soon as player 0 mines a block in the blockchain, and she continues mining on the new branch until it becomes the blockchain. Here player 1 is willing to fork every time player 0 produces a block in the blockchain. In other words, in AF, player 1 tries to have all the blocks in the blockchain. This strategy is depicted in Figure 2.

The utility of always forking. We want to answer two questions. On the one hand, we want to know whether AF is a better strategy than DF_1 for player 1, under the assumption that player 0 uses DF_0, and under some specific values of α, β and h. On the other hand, and perhaps more interestingly, we can also answer a more analytical question: given realistic values of α and β, how much hash power does player 1 need to consider following AF instead of DF_1? Answering both questions requires us to compute the utility for the strategy $\mathbf{AF} = (\text{DF}_0, \text{AF})$.

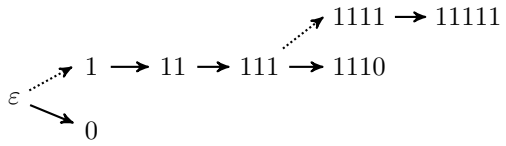


Figure 2 Dashed arrows indicate when player 1 does a fork. The first block (block 0) is mined by player 0. At this point, player 1 decides to fork (mining the block 1), and successfully mines the blocks 11 and 111 on this branch. When player 0 mines the block 1110, player 1 decides to fork again, mining the blocks 1111 and 11111.

▶ **Theorem 4.2.** *Let* $\mathbf{C}(x) = \frac{1-\sqrt{1-4x}}{2x}$ *denote the generating function of Catalan numbers. If* h *is the hash power of player 1, then*

$$u_1(\mathbf{AF}) \;=\; \frac{\Phi}{1-\Gamma}, \text{ where } \Phi \text{ and } \Gamma \text{ are defined as:}$$

$$\Phi \;=\; \frac{\alpha\cdot\beta\cdot h\cdot c}{(1-\alpha)}\cdot\left[\mathbf{C}(\beta^2\cdot h\cdot(1-h)) - \alpha\cdot\mathbf{C}(\alpha\cdot\beta^2\cdot h\cdot(1-h))\right],$$
$$\Gamma \;=\; \alpha\cdot\beta\cdot h\cdot\mathbf{C}(\alpha\cdot\beta^2\cdot h\cdot(1-h))$$

Let us give some intuition on this result. Player 1, adopting the AF strategy, will always start the game mining on ε, regardless of how many blocks player 0 manages to append, and continues until her branch is the longest. Therefore, the only states that contribute to player 1's utility are those in where she made at least one successful fork (all others states give zero reward to her). Having player 1 achieved the longest branch once, say, at block b, both players will now mine on b and the situation repeats as if b were ε, with proper shifting in the reward and β-discount. In other words, we have $u_1(\mathbf{AF}) = \Phi + \Gamma\cdot u_1(\mathbf{AF})$, where Φ is the contribution of a single successful fork, and Γ is the shifting factor, from which we obtain the expression for $u_1(\mathbf{AF})$ given before.

Now, in order to quantify the contribution of Φ on successful forks, we need to sum over all possible moments of time in which this fork was finally made, weighted by the possibility that such a fork was actually made. However, this is not direct because there may be different paths leading to the same state, and therefore the probability of forking at a certain stage depends on the length and the form of the state. We quantify these by bringing out an analogy between Dyck words [21] and paths leading to states in which player 1 forks successfully for the first time. Then the theorem uses the fact that the number of Dyck words of length $2m$ is the m-th Catalan number.

When is AF better than DF? With the closed forms for $u_1(\mathbf{DF})$ and $u_1(\mathbf{AF})$, we can compare the utilities of these strategies for player 1 for fixed and realistic values of α and β, but varying her hash power. For α we calculate the compound version of the discount in Bitcoin, that is, a value of α that would divide the reward by half every 210.000 blocks, i.e. $\alpha = 0.9999966993$. For β we calculate the 10-minute rate that is equivalent to the US real interest rate in the last few years, which is approximately 2%. This gives us a value of $\beta = 0.9999996156$.

Figure 4a shows the value of the utility of player 1 for the combined strategies **AF** and **DF** (this figure also includes two other strategies that will be explained in the next section). The plot data was generated using GMP C++ multi precision library [7]. The point where

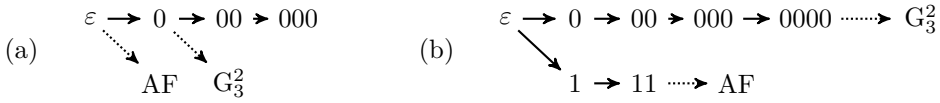


Figure 3 Difference between AF and G_3^2 in terms of actions in two states.

the utility for **AF** and **DF** meet is $h = 0.499805 \pm 0.000001$, which means that player 1 should use **AF** as soon as she controls more than this proportion of the hash power (a similar result was obtained in [10], although in a model without discounted reward).

4.2 Giving up for more utility

By adding a little more flexibility to the strategy of always forking, we can identify approaches that make a fork profitable with less hash power. The families of strategies that we study in this section involve two parameters. The first parameter, denoted by k, regulates how far back the miner will fork, when confronted with a chain of blocks she does not own. The second parameter, called the *give-up* time, and denoted by ℓ, tells us the maximum number of blocks that the player's opponent is allowed to extend the current blockchain with before the player gives up mining on the forking branch. If the player does not manage to transform her fork into the new blockchain before her opponent mines more than ℓ blocks, she will restart the strategy treating the tail of the current blockchain as the new genesis block. We denote these strategies by G_ℓ^k.

▶ **Example 4.3.** Let us compare G_3^2 and AF. Since $k = 2$, both strategies take the same action when the state is $\{\varepsilon\}, \{\varepsilon, 0\}$, and $\{\varepsilon, 0, 00\}$, namely, mining at ε. Hence, assume that both strategies are facing a chain of three blocks owned by player 0, as shown in Figure 3(a). In this case, AF would again try to do a fork from the genesis block as no block belongs to player 1. On the other hand, G_3^2 would try to fork on the dotted line, that is, the second block that does not belong to her. The second difference is provided by the give-up time, which is shown in Figure 3(b). Normally, AF is willing to continue forking regardless of the hope of winning, therefore the move for the state in Figure 3(b) would still be to mine upon her own block 11. On the other hand, G_3^2 has now seen 4 blocks from the start of the fork (one more than the maximum $\ell = 3$), so with this strategy player 1 instead gives up and tries to mine upon 0000, rebooting the strategy as if 0000 was the genesis block. Note also that $\text{AF} = G_\infty^\infty$.

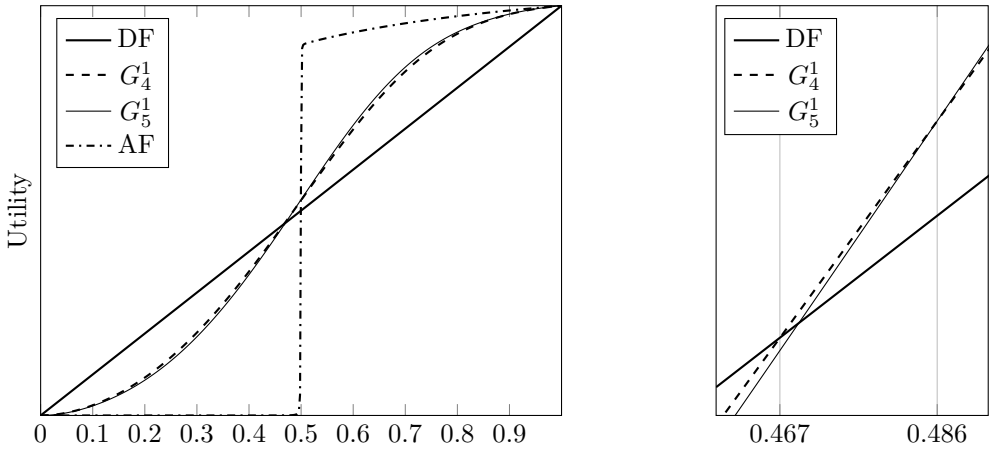
Define $\mathbf{G}_\ell^k$ as the combined strategy (DF_0, G_ℓ^k). We obtain an analytical form similar to that of Theorem 4.2, except in this case the set of paths leading to winning states has a more complex combinatorial nature, as expected when taking into account the parameters k and ℓ.

▶ **Theorem 4.4.** *For every pair of positive integers ℓ, k with $k < \ell$, we have that:*

$$u_1(\boldsymbol{G}_\ell^k) = \frac{\Phi_{\ell,k}}{1 - \Gamma_{\ell,k}},$$

where $\Phi_{\ell,k}$ and $\Gamma_{\ell,k}$ are rational functions of α, β and h.

In the proof of this theorem, available in the full version of this article, we develop precise expressions for $\Phi_{\ell,k}, \Gamma_{\ell,k}$. The proof extends the techniques used to show Theorem 4.2, where we again look to compute the weighted sum of all states where player 1 manages to fork. This weighted sum, however, requires much more involved computation; we use a new combinatorial result that involves two sets of polynomials related to Dyck words.



(a) Utilities for player 1 of combined strategies, **DF**, $\mathbf{G}_4^1$, $\mathbf{G}_5^1$ and **AF**.

(b) Zoom in around the intersection.

Figure 4 Comparing the utilities of three forking strategies against the default strategy.

We use Theorem 4.4 to analyse these strategies, plotting them, as we did before, for $\alpha = 0.9999966993$ and $\beta = 0.9999996156$. Figures 4a and 4b give interesting information about the advantages of these strategies. We fix $k = 1$, and plot in Figure 4a the utilities of combined strategies **DF**, $\mathbf{G}_4^1$, $\mathbf{G}_5^1$ and **AF**. In Figure 4b, we zoom in around the values of the hash power where **DF** intersects with $\mathbf{G}_4^1$ and $\mathbf{G}_5^1$. As we see in the figures, for a fork window $k = 1$, the optimal amount time player 1 should be willing to fight for a branch before giving up depends on the hash power. With little hash power the likelihood of winning a branch is small, so player 1 should give up as early as possible. However, the more hash power she obtains, the better it is to wait more. Interestingly, with more than 46.7% of the hash power, player 1 already should start using strategy G_4^1 to defeat the default strategy, and with more than 48.6% hash power, she should adopt G_5^1. We know that player 1 should use AF not before around $h = 0.499805$, so this gives us a lot of extra room to look for optimal strategies if we are willing to fork (especially considering that every percentage of hash power in popular cryptocurrencies may cost millions).

Plots for strategies with $k > 1$ present a similar behaviour: the more hash power we have, the more we should be willing to fight for our forks. The strategies we include in Figure 4 beat the default strategy under the least amount of hash power amongst any combination of values for k and ℓ with $k < \ell \leq 100$. The comparison is much less straightforward when looking at varying values of both k and ℓ, but in general, the more hash power the bigger the window of blocks one should aim to do a fork, and the more one should wait before giving up.

5 Concluding remarks

Our model of mining via a stochastic game allows for an intuitive representation of miners' actions as strategies, and gives us a way of understanding the rational behaviour of miners looking to accumulate cryptocurrency wealth. As it is the first model to provide payoff to miners for every branching strategy we can validate the commonly accepted assumption that long forks are not a viable strategy. In this respect, we would like to identify strategies that are a Nash equilibrium for the case of decreasing rewards. However, this has proven to be a difficult task. In particular, one can show that the default strategy can never be part of such

an equilibrium, no matter how small the hash power is for one of the players, if the strategy of another player involves forks of any length. This means that one must look for much more complex strategies to find such an equilibrium.

One of the advantages of our model is its generality: it can be adapted to specify more complex actions, study other forms of reward and include cooperation between miners. For example, we are currently looking at strategies that involve withholding a mined block to the rest of the network, for which we need a slight extension of the notions of action and state. An interesting venue for future work is to study how this model and previous work combine into a model where miner's behavior can deviate depending on both their short-and long-term goals. We would also like to study incentives under different models of cooperation between miners, and also other forms of equilibria in a dynamic setting.

References

1 Lear Bahack. Theoretical bitcoin attacks with less than half of the computational power (draft). *CoRR*, 2013. `arXiv:1312.7013`.

2 Bruno Biais, Christophe Bisière, Matthieu Bouvard, and Catherine Casamatta. The Blockchain Folk Theorem. *The Review of Financial Studies*, 32(5):1662–1715, 04 2019. `doi:10.1093/rfs/hhy095`.

3 Miles Carlsten, Harry Kalodner, S. Matthew Weinberg, and Arvind Narayanan. On the instability of bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

4 Mauro Conti, Sandeep Kumar, Chhagan Lal, and Sushmita Ruj. A survey on security and privacy issues of bitcoin. *IEEE Communications Surveys & Tutorials*, 2018.

5 Swapnil Dhamal, Tijani Chahed, Walid Ben-Ameur, Eitan Altman, Albert Sunny, and Sudheer Poojary. A stochastic game framework for analyzing computational investment strategies in distributed computing with application to blockchain mining. *arXiv preprint*, 2018. `arXiv:1809.03143`.

6 Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, 2014.

7 GNU. The gnu multiple precision arithmetic library (v. 6.1.2), 2018. URL: `https://gmplib.org/`.

8 Ethan Heilman. One weird trick to stop selfish miners: Fresh bitcoins, a solution for the honest miner (poster abstract). In *Financial Cryptography and Data Security*, 2014.

9 Benjamin Johnson, Aron Laszka, Jens Grossklags, Marie Vasek, and Tyler Moore. Game-theoretic analysis of ddos attacks against bitcoin mining pools. In *Financial Cryptography and Data Security*, 2014.

10 Aggelos Kiayias, Elias Koutsoupias, Maria Kyropoulou, and Yiannis Tselekounis. Blockchain mining games. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, EC '16, pages 365–382, 2016.

11 Elias Koutsoupias, Philip Lazos, Foluso Ogunlana, and Paolo Serafino. Blockchain mining games with pay-forward, 2019. *To appear in The Web Conference*.

12 Joshua A. Kroll, Ian C. Davey, and Edward W. Felten. The economics of bitcoin mining, or bitcoin in the presence of adversaries. In *The Twelfth Workshop on the Economics of Information Security (WEIS 2013)*, 2013.

13 Michael Mitzenmacher and Eli Upfal. *Probability and computing - randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

14 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `http://bitcoin.org/bitcoin.pdf`, 2008.

15 Arvind Narayanan, Joseph Bonneau, Edward W. Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and Cryptocurrency Technologies - A Comprehensive Introduction*. Princeton University Press, 2016.

16 Arvind Narayanan and Jeremy Clark. Bitcoin's academic pedigree. *Commun. ACM*, 60(12):36–45, 2017.

17 Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. *IACR Cryptology ePrint Archive*, 2015:796, 2015.

18 Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016*, pages 305–320, 2016.

19 Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. In *Financial Cryptography and Data Security*, pages 515–532, 2017.

20 Alexander Spiegelman, Idit Keidar, and Moshe Tennenholtz. Game of coins. *arXiv preprint*, 2018. `arXiv:1805.08979`.

21 R.P. Stanley. *Catalan Numbers*. Cambridge University Press, 2015.

22 Christopher P Thompson. *Ethereum - Distributed Consensus (A Concise Ethereum History Book)*. CreateSpace Independent Publishing Platform, 2017.

23 Itay Tsabary and Ittay Eyal. The gap game. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 713–728. ACM, 2018.

24 Marie Vasek, Micah Thornton, and Tyler Moore. Empirical analysis of denial-of-service attacks in the bitcoin ecosystem. In *Financial Cryptography and Data Security*, 2014.

25 Bitcoin Cash Website. `https://www.bitcoincash.org`, 2018.

26 Ethereum Website. `https://ethereum.org`, 2018.

27 Litecoin Website. `https://litecoin.org`, 2018.

28 Monero Website. `https://getmonero.org`, 2018.

Randomness and Initial Segment Complexity for Probability Measures

André Nies
School of Computer Science, The University of Auckland, New Zealand
https://www.cs.auckland.ac.nz/~nies/
andre@cs.auckland.ac.nz

Frank Stephan
Department of Mathematics and Department of Computer Science,
National University of Singapore, Singapore
fstephan@comp.nus.edu.sg

Abstract

We study algorithmic randomness properties for probability measures on Cantor space. We say that a measure μ on the space of infinite bit sequences is Martin-Löf absolutely continuous if the non-Martin-Löf random bit sequences form a null set with respect to μ. We think of this as a weak randomness notion for measures. We begin with examples, and a robustness property related to Solovay tests. Our main work connects our property to the growth of the initial segment complexity for measures μ; the latter is defined as a μ-average over the complexity of strings of the same length. We show that a maximal growth implies our weak randomness property, but also that both implications of the Levin-Schnorr theorem fail. We briefly discuss K-triviality for measures, which means that the growth of initial segment complexity is as slow as possible. We show that full Martin-Löf randomness of a measure implies Martin-Löf absolute continuity; the converse fails because only the latter property is compatible with having atoms. In a final section we consider weak randomness relative to a general ergodic computable measure. We seek appropriate effective versions of the Shannon-McMillan-Breiman theorem and the Brudno theorem where the bit sequences are replaced by measures.

2012 ACM Subject Classification Theory of computation; Theory of computation → Quantum information theory

Keywords and phrases algorithmic randomness, probability measure on Cantor space, Kolmogorov complexity, statistical superposition, quantum states

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.55

Funding *André Nies*: Partially supported by the Marsden Fund of the Royal Society of New Zealand, UOA 13-184.
Frank Stephan: Partially supported by the Singapore Ministry of Education Academic Research Fund Tier 2 grant MOE2016-T2-1-019 / R146-000-234-112.

1 Introduction

The theory of algorithmic randomness is usually developed for bit sequences. A central randomness notion based on algorithmic tests is the one due to Martin-Löf [11].

Let $\{0,1\}^{\mathbb{N}}$ denote the topological space of infinite bit sequences. A probability measure μ on $\{0,1\}^{\mathbb{N}}$ can be seen as a statistical superposition of bit sequences. The bit sequences Z form an extreme case: the corresponding measure μ is the Dirac measure δ_Z, i.e., μ is concentrated on $\{Z\}$. The opposite extreme is the uniform measure λ which independently gives each bit value the probability 1/2. The uniform measure represents the maximum disorder as no bit sequence is preferred over any other.

Recall that a measure μ on $\{0,1\}^{\mathbb{N}}$ is called *absolutely continuous* if each λ-null set is a μ-null set. Our main concept is an algorithmic randomness notion for probability measures that is a weakening of absolute continuity: we require that the λ-null set in the hypothesis be effective in the sense of Martin-Löf. Given that there is a universal Martin-Löf test, and hence a largest effective null set, all we have to require is that $\mu(\mathcal{C}) = 0$ where $\mathcal{C}$ is the class of bit sequences that are not Martin-Löf random.

Our research is partly motivated by a recent definition of Martin-Löf randomness for quantum states corresponding to infinitely many qubits, due to the first author and Scholz [18]. Using the terminology there, probability measures correspond to the quantum states ρ where the matrix $\rho\restriction_{M_n}$ is diagonal for each n, where M_n is the algebra of complex $2^n \times 2^n$ matrices. Subsequent work of Tejas Bhojraj has shown that for measures, the randomness notion defined there is equivalent to the one proposed here. So the measures form a useful intermediate case to test conjectures in the subtler setting of quantum states. This applies, for instance, to the SMB theorem discussed at the end of this section, which is studied in the setting of quantum states by the first author and Tomamichel (see the post in [20]).

Growth of initial segment complexity. Given a binary string x, by $C(x)$ one denotes its plain descriptive complexity, and by $K(x)$ its prefix-free descriptive complexity. Our main motivation is derived from the classical theory. Randomness of infinite bit sequences is linked to the growth of the descriptive complexity of their initial segments. For instance, the Levin-Schnorr theorem intuitively says that randomness of Z means incompressibility (up to the same constant b) of all the initial segments of Z. We want to study how much of this is retained in the setting of measures μ. One now takes the μ-average over the complexity of all strings of a given length n. It turns out that interesting new growth behaviour is possible, such as having maximal growth of C-complexity on all initial segments. This growth rate is ruled out for bit sequences by a result of Katseff. However, using that "most strings are incompressible" we verify in Fact 9 that the uniform measure λ has this growth behaviour, namely $C(\lambda\restriction_n) \geq^+ n$. On the other hand, we show that this type of fast growth implies our weak randomness notion.

The formal growth condition in the Levin-Schnorr theorem says that $K(x) \geq |x| - b$ for each initial segment x of Z, where $K(x)$ is the prefix free version of Kolmogorov complexity of a string x. The "n-th initial segment" of a measure μ is given by its values $\mu[x]$ for all strings x of length n, where $[x]$ denotes the set of infinite sequences extending x. As mentioned, it is natural to define the initial segment complexities $C(\mu\restriction_n)$ and $K(\mu\restriction_n)$ of this initial segment as the μ-average of the individual complexities of those strings. With this definition, in Section 3 we show that both implications of the analog of the Levin-Schnorr theorem fail. However, we also show in Proposition 25 that for measures that are random in our weak sense, $C(\mu\restriction_n)/n$, or equivalently $K(\mu\restriction_n)/n$, converges to 1. Thus, such measures have effective dimension 1; see Downey and Hirschfeldt [5, Section 12.3] on effective dimension.

Further results and potential avenues for future research. Opposite to random bit sequences are the K-trivial sequences, where the initial segment complexity grows no faster than that of a computable set; for background see e.g. Nies [17, Section 5.3]. In Section 4 we briefly extend this notion to statistical superpositions of bit sequences: we introduce K-trivial measures and show that they have countable support. This means that they are countable combinations of Dirac measures.

Measures can be viewed points in a canonical computable probability space, in the sense of [8]. This yields a notion of Martin-Löf randomness for measures. Culver [4, Th. 2.7.1] has shown that no measure μ that is Martin-Löf random is absolutely continuous; in fact μ

is orthogonal to λ in the sense that some null set is co-null with respect to μ. In contrast, in Section 5 we show that this notion implies our weak notion of randomness, ML-absolute continuity. The stronger randomness notion forces the measure to be atomless, so the converse implication fails. Further questions can be asked about the relationships between the different randomness notions for measures we have discussed. For instance, does the strong notion imply maximal growth of initial segment complexity for the measure (in the sense of C or of K)? We plan to address such questions in the upcoming journal version of the paper.

The Shannon-McMillan-Breiman Theorem from the 1950s (see [24], where it is called the Entropy Theorem) says informally that for an ergodic measure ρ on $\{0,1\}^{\mathbb{N}}$, outside a null set every bit sequence Z reflects the entropy of the measure ρ by the limiting weighted information content on its sufficiently large initial segments. In the final Section 6 we study what happens when Z is replaced by a measure μ that is Martin-Löf a.c. with respect to ρ and we take the μ-average of the information contents at the same length. Here we only obtain a partial result. However, in a similar vein, we establish an analog for measures of the effective Brudno's theorem [6, 7] that the entropy of ρ is given as the limit of $K(Z\upharpoonright_n)/n$, for any Z that is ρ-ML random. Obtaining a measure version of the effective Birkhoff ergodic theorem would be interesting as well.

For general background on recursion theory and algorithmic randomness we refer the readers to the textbooks of Calude [2], Downey and Hirschfeldt [5], Li and Vitányi [10], Nies [17], Odifreddi [21, 22], and Soare [25]. Lecture notes on recursion theory are also available online, e.g. [26].

2 Measures and Randomness

In this section we formally define our main notion (Definition 3), and collect some basic facts concerning it. In particular, we verify that the well-known equivalence of Martin-Löf test and Solovay tests extends to measures. We begin by briefly discussing algorithmic randomness for bit sequences [5, 17]. We use standard notation: letters $Z, X, \ldots$ denote elements of the space of infinite bit sequences $\{0,1\}^{\mathbb{N}}$, σ, τ denote finite bit strings, and $[\sigma] = \{Z\colon\ Z \succ \sigma\}$ is the set of infinite bit sequences extending σ. $Z\upharpoonright_n$ denotes the string consisting of the first n bits of Z. For quantities r, s depending on the same parameters, one writes $r \leq^+ s$ for $r \leq s + O(1)$. A subset G of $\{0,1\}^{\mathbb{N}}$ is called *effectively open* if $G = \bigcup_i [\sigma_i]$ for a computable sequence $\langle \sigma_i \rangle_{i\in\mathbb{N}}$ of strings. A measure ρ on $\{0,1\}^{\mathbb{N}}$ is computable if the map $\{0,1\}^{<\omega} \to \mathbb{R}$ given by $\sigma \mapsto \rho[\sigma]$ is computable.

▶ **Definition 1.** *Let ρ be a computable measure on $\{0,1\}^{\mathbb{N}}$. A ρ-Martin-Löf test (ρ-ML-test, for short) is a sequence $\langle G_m \rangle$ of uniformly effectively open sets such that $\rho G_m \leq 2^{-m}$ for each m. A bit sequence Z* fails *the test if $Z \in \bigcap_m G_m$, otherwise it* passes *the test. A bit sequence Z is ρ- Martin-Löf random (ML-random) if Z passes each ρ-ML-test.*

By λ one denotes the uniform measure on $\{0,1\}^{\mathbb{N}}$. So $\lambda[\sigma] = 2^{-|\sigma|}$ for each string σ. If no measure ρ is provided it will be tacitly assumed that $\rho = \lambda$, and we will use the term ML-random instead of λ-ML-random etc. Let $K(x)$ denote the prefix free version of descriptive (i.e., Kolmogorov) complexity of a bit string x.

▶ **Theorem 2** (Levin [9], Schnorr [23]). *Z is ML-random $\Leftrightarrow \exists b \forall n K(Z\upharpoonright_n) \geq n - b$.*

Using the notation of [17, Ch. 3], let $\mathcal{R}_b$ denote the set of bit sequences Z such that $K(Z\upharpoonright_n) < n - b$ for some n. It is easy to see that $\langle \mathcal{R}_b \rangle_{b\in\mathbb{N}}$ forms a Martin-Löf test. The Levin-Schnorr theorem says that this test is universal: Z is ML-random iff it passes the test.

Unless otherwise stated, all measures will be probability measures. We use the letters μ, ν, ρ for probability measures (and recall that λ denotes the uniform measure). We now provide the formal definition of our weak randomness notion for measures.

▶ **Definition 3** (Main). *A measure μ is called* Martin-Löf absolutely continuous *in ρ (ρ-ML a.c., for short) if $\inf_m \mu(G_m) = 0$ for each ρ-Martin-Löf test $\langle G_m \rangle_{m\in\mathbb{N}}$. We write $\mu \ll_{ML} \rho$.*

If $\inf_m \mu(G_m) = 0$ we say that μ *passes* the test. If $\inf_m \mu(G_m) \geq \delta$ where $\delta > 0$ we say μ *fails the test at level δ*.

Martin-Löf absolute continuity is a *weakening* of the usual notion of absolute continuity $\mu \ll \rho$. In fact, $\mu \ll \rho$ iff μ is ρ-ML^X-a.c. for each oracle X.

In the definition it suffices to consider ρ-ML tests $\langle G_m \rangle$ such that $G_m \supseteq G_{m+1}$ for each m, because we can replace $\langle G_m \rangle$ by the ρ-ML test $\widehat{G}_m = \bigcup_{k>m} G_k$, and of course $\inf_m \mu(\widehat{G}_m) = 0$ implies $\inf_m \mu(G_m) = 0$. So we can change the definition above, replacing the condition $\inf_m G_m = 0$ by the only apparently stronger condition $\lim_m G_m = 0$.

The intersection of a universal ρ-ML test consists of the non-ML random sequences. Since such a test exists, we have:

▶ **Fact 4.** *$\mu \ll_{ML} \rho$ iff the sequences which are not ρ-ML random form a μ-null set.*

We have already mentioned the two diametrically opposite types of examples:

▶ **Example 5.** (a) $\rho \ll_{ML} \rho$.

(b) For a Dirac measure δ_Z, we have $\delta_Z \ll_{ML} \rho$ iff Z is ρ-ML random.

For $p \neq 1/2$, a Bernoulli measure on $\{0,1\}^{\mathbb{N}}$, that independently gives probability p to a 0 in each position, is not Martin-Löf a.c. To see this, note that each ML-random sequence Z satisfies the law of large numbers

$$\lim_n \frac{1}{n} |\{i < n \colon Z(i) = 1\}| = 1/2;$$

see e.g. [17, Prop. 3.2.19]. So if μ is Martin-Löf a.c., then μ-almost surely, Z satisfies the law of large numbers. This is not the case for such Bernoulli measures.

For a measure ν and string σ with $\nu[\sigma] > 0$ let ν_σ be the localisation:

$$\nu_\sigma(A) = \nu(A \cap [\sigma])/\nu[\sigma].$$

Clearly if ν is Martin-Löf a.c. then so is ν_σ.

A set S of probability measures is called *convex* if $\mu_i \in S$ for $i \leq k$ implies that the convex combination $\mu = \sum_i \alpha_i \mu_i$ is in S, where the α_i are reals in $[0,1]$ and $\sum_i \alpha_i = 1$. The extreme points of S are the ones that can only be written as convex combinations of length 1 of elements of S.

▶ **Proposition 6.** *The Martin-Löf a.c. probability measures form a convex set. Its extreme points are the Martin-Löf a.c. Dirac measures, i.e. the measures δ_Z where Z is a ML-random bit sequence.*

Proof. Let $\mu = \sum_i \alpha_i \mu_i$ as above where the μ_i are Martin-Löf a.c. measures. Suppose $\langle G_m \rangle$ is a Martin-Löf test. Then $\lim_m \mu_i(G_m) = 0$ for each i, and hence $\lim_m \mu(G_m) = 0$.

Suppose that μ is Martin-Löf a.c. If μ is a Dirac measure then it is an extreme point of the Martin-Löf a.c. measures. Conversely, if μ is not Dirac, there is a least number t such that the decomposition

$$\mu = \sum_{|\sigma|=t, \mu[\sigma]>0} \mu[\sigma] \cdot \mu_\sigma$$

is nontrivial. Hence μ is not an extreme point. ◀

Recall that a *Solovay test* is a sequence $\langle S_k\rangle_{k\in\mathbb{N}}$ of uniformly Σ^0_1 sets such that $\sum_k \lambda(S_k) < \infty$. A bit sequence Z passes such a test if $Z \notin S_k$ for almost every k. (Each ML-test is a Solovay test, but the passing condition is stronger for Solovay tests). A basic fact from the theory of algorithmic randomness (e.g. [17, 3.2.9]) states that Z is ML-random iff Z passes each Solovay test.

The following characterises the Martin-Löf a.c. measures with countable support.

▶ **Fact 7.** *Let $\mu = \sum_k c_k\delta_{Z_k}$ where $\forall k\,[0 < c_k \leq 1]$ and $\sum_k c_k = 1$. Then μ is Martin-Löf a.c. iff all the Z_k are Martin-Löf random.*

Proof. The implication from left to right is immediate. For the converse implication, given a Martin-Löf test $\langle G_m\rangle$, note that the Z_k pass this test as a Solovay test. Hence for each r, there is M such that $Z_k \notin G_m$ for each $k \leq r$ and each $m \geq M$. This implies that $\mu(G_m) \leq \sum_{k>r} c_k$ for each $m \geq M$. So $\lim_m \mu(G_m) = 0$. ◀

We say that a measure μ *passes* a Solovay test $\langle S_k\rangle_{k\in\mathbb{N}}$ if $\lim_k \mu(S_k) = 0$. The fact that passing all Martin-Löf tests is equivalent to passing all Solovay tests generalises from bit sequences to measures. We note that Tejas Bhojraj (in preparation) proved such a result in even greater generality in the setting of quantum states, where the proof is more involved.

▶ **Proposition 8.** *A measure μ is Martin-Löf a.c. iff μ passes each Solovay test.*

Proof. Each Martin-Löf test is a Solovay test, and the passing condition $\lim_m \mu(G_m) = 0$ works for both types of tests by the remark after Definition 3. This yields the implication from right to left.

For the implication from left to right, let μ be Martin-Löf a.c. and let $\langle S_k\rangle_{k\in\mathbb{N}}$ be a Solovay test. By $\limsup_k S_k$ one denotes the set of bit sequences Z such that $\exists^\infty k\, Z \in S_k$, that is, the sequences that fail the test. By the basic fact (e.g. [17, 3.2.9]) mentioned above, the set of ML-random sequences is disjoint from $\limsup_k S_k$. By hypothesis on μ we have $\mu(\limsup_k S_k) = 0$. By Fatou's Lemma, $\limsup_k \mu(S_k) \leq \mu(\limsup_k S_k)$. So μ passes the Solovay test. ◀

3 Initial segment complexity of a measure μ

Let $K(\mu\restriction_n) = \sum_{|x|=n} K(x)\mu[x]$ be the μ-average of all the $K(x)$ over all strings x of length n. In a similar way we define $C(\mu\restriction_n)$. Note that for a Dirac measure δ_Z, we have $K(\delta_Z\restriction_n) = K(Z\restriction_n)$.

In this section we use standard inequalities such as $C(x) \leq^+ K(x)$, $K(x) \leq^+ |x| + 2\log|x|$ and $K(0^n) \leq^+ 2\log n$. We also use that for each r there are at most $2^r - 1$ strings such that $C(x) < r$. See e.g. [17, Ch. 2]. Recall that λ denotes the uniform measure on $\{0,1\}^{\mathbb{N}}$.

3.1 A fast growing initial segment complexity implies being ML-a.c.

Recall that $C(x) \leq^+ |x|$ and $K(x) \leq^+ |x| + K(|x|)$. The following says that the uniform measure λ has the fastest growing initial segment complexity that is possible in both sense of C and of K.

▶ **Fact 9.**

(a) $C(\lambda\restriction_n) \geq^+ n$.

(b) $K(\lambda\restriction_n) \geq^+ n + K(n)$.

Proof. Chaitin [3] showed that there is a constant c such that, for all d, there are at most 2^{n+c-d} strings $x \in \{0,1\}^n$ with $C(x) \leq n-d$. Similarly, among the strings of length n, there are at most 2^{n+c-d} strings with $K(x) \leq n + K(n) - d$. In other words, the fraction of strings of length n where, for (a), $C(x) \leq n-d$, and, for (b), $K(x) \leq n + K(n) - d$, respectively, is in each case at most 2^{c-d}. Now for each d, from the estimated lower bound n and $n + K(n)$, respectively, one subtracts the fraction of the strings of length n for which the Kolmogorov complexity is at least d below the average in order to correct the lower bound. For, if x is a string of length n such that $C(x) \leq n - r$ (resp, $K(x) \leq n + K(n) - r$), then in computing $C(\lambda\restriction_n)$ (resp, $K(\lambda\restriction_n)$) a correction of 2^{-n} has to be subtracted r times, for $d = 1, \ldots, r$.

Let c_d be the fraction of strings of length n with $C(x) \leq n-d$, and let k_d be the fraction of strings with $K(x) \leq n + K(n) - d$. Then as argued above,

$$C(\lambda\restriction_n) \geq n - \sum_{d\geq 0} c_d \text{ and } K(\lambda\restriction_n) \geq n + K(n) - \sum_{d\geq 0} k_d.$$

Using Chaitin's bounds gives then the corrected estimates on the averages of

$$C(\lambda\restriction_n) \geq n - \sum_{d\geq 0} 2^{c-d} \text{ and } K(\lambda\restriction_n) \geq n + K(n) - \sum_{d\geq 0} 2^{c-d}.$$

Now one uses that $\sum_{d\geq 0} 2^{c-d} \leq 2^{c+1}$ and that 2^{c+1} is a constant independent of n and only dependent on the universal machine in order to get that $C(\lambda\restriction_n) \geq^+ n$ and $K(\lambda\restriction_n) \geq^+ n + K(n)$. ◀

Recall [10] that a bit sequence $Z \in \{0,1\}^{\mathbb{N}}$ is called Kolmogorov random if there is r such that $C(Z\restriction_n) \geq n - r$ for infinitely many n; Z is strongly Chaitin random if there is r such that $K(Z\restriction_n) \geq n + K(n) - r$ for infinitely many n. For bit sequences these notions are equivalent to 2-randomness by [19] and [13], respectively; also see [17, 8.1.14] or [5].

We may extend these notion to measures. We show that a measure satisfying either of the notions is Martin-Löf a.c.:

▶ **Theorem 10.**

(a) *Suppose that μ is a measure such that there is an r with $C(\mu\restriction_n) \geq n - r$ for infinitely many n. Then μ is Martin-Löf a.c.*

(b) *The same conclusion holds under the hypothesis that $K(\mu\restriction_n) \geq n + K(n) - r$ for infinitely many n.*

Proof. Suppose that μ is not Martin-Löf a.c. So there is a Martin-Löf test $\langle G_d\rangle_{d\in\mathbb{N}}$ and $\delta > 0$ such that $\mu(G_d) \geq \delta$ for each d. We view G_d as given by an enumeration of strings, uniformly in d; thus $G_d = \bigcup_i [\sigma_i]$ for a sequence $\langle\sigma_i\rangle_{i\in\mathbb{N}}$ that is computable uniformly in d. Let $G_d^{\leq n}$ denote the clopen set generated by the strings in this enumeration of length at most n. (Note that this set is not effectively given as a clopen set, but we effectively have a description of it as a Σ^0_1 set). Let c be a constant such that, for each x of length n, one has $C(x) \leq n + c$ in case (a), and $K(x) \leq n + K(n) + c$ in case (b).

▶ **Lemma 11.** *If x is a string of length n such that $[x] \subseteq G_d^{\leq n}$ then $C(x \mid d) \leq^+ n - d$ and $K(x \mid n, d) \leq^+ n - d$.*

To verify this, we first consider the case of plain complexity C. Let N be a fixed plain machine that on input y and auxiliary input d prints out the y-th string x of length $n = |y| + d$ such that our enumeration of $G_d^{\leq n}$ asserts that $[x] \subseteq G_d^{\leq n}$. (Here we view y as the binary representation of a number, with leading zeros allowed.) Since $\lambda G_d \leq 2^{-d}$, sufficiently many strings are available to print all such x. This machine shows that $C(x \mid d) \leq^+ n - d$ for any x such that $[x] \subseteq G_d^{\leq n}$, as required.

For the case of prefix free complexity K, let N' be the slightly modified machine where both n and d are auxiliary inputs. The machine N provides for a string x of length n a description of length $n-d$. So for N', for the same pair n, d, the descriptions of different strings form a prefix free set. This verifies the lemma.

Now such x satisfy (after increaing x, if necessary) that $C(x) \leq n - d + 2\log d + c$ and $K(x) \leq n + K(n) - d + 2\log d + c$. We complete the proof separately for (a) and (b).

(a) For each d, n, letting x range over strings of length n, we have

$$C(\mu\restriction_n) = \sum_{|x|=n} C(x)\mu[x] = \sum_{[x]\subseteq G_d^{\leq n}} C(x)\mu[x] + \sum_{[x]\not\subseteq G_d^{\leq n}} C(x)\mu[x].$$

The first summand is bounded above by $\mu(G_d^{\leq n})(n - d + 2\log d + c)$ via the lemma, the second by $(1 - \mu(G_d^{\leq n}))(n + c)$. We obtain

$$C(\mu\restriction_n) \leq n + c - \mu(G_d^{\leq n})d/2.$$

Now for each d, for sufficiently large n we have $\mu(G_d^{\leq n}) \geq \delta$. So given r let $d = 2r/\delta$; then for large enough n we have $C(\mu\restriction_n) \leq n + c - r$.

(b) For each d, n, letting x range over strings of length n, we have

$$K(\mu\restriction_n) = \sum_{|x|=n} K(x)\mu[x] = \sum_{[x]\subseteq G_d^{\leq n}} K(x)\mu[x] + \sum_{[x]\not\subseteq G_d^{\leq n}} K(x)\mu[x].$$

The first summand is bounded above by $\mu(G_d^{\leq n})(n + K(n) - d + 2\log d + c)$, the second by $(1 - \mu(G_d^{\leq n}))(n + K(n) + c)$. We obtain

$$K(\mu\restriction_n) \leq n + K(n) + c - \mu(G_d^{\leq n})d/2.$$

Now for each d, for sufficiently large n we have $\mu(G_d^{\leq n}) \geq \delta$. As before, given r let $d = 2r/\delta$; then for large enough n we have $K(\mu\restriction_n) \leq n + K(n) + c - r$. ◀

It would be interesting to know whether the above-mentioned coincidences of randomness notions for bit sequences lift to measures; for instance, do the conditions in the theorem above actually imply that the measure is ML-a.c. relative to the halting problem $\emptyset'$?

3.2 Both implications of the Levin-Schnorr Theorem fail for measures

We will show that both implications of the analog of the Levin-Schnorr Theorem 2 fail for measures. One implication would say that a Martin-Löf a.c. measure cannot have initial segment complexity in the sense of K growing slower than $n - O(1)$. This can be disproved by a simple example of a measure with countable support. On the other hand, by Proposition 25 below, we have $\lim_n K(\mu\restriction_n)/n = 1$ for each Martin-Löf a.c. measure μ, which provides a lower bound on the growth.

▶ **Example 12.** Let $\theta \in (0,1)$. There is a Martin-Löf a.c. measure μ such that $K(\mu\restriction_n) \leq^+ n - n^\theta$.

Proof. We let $\mu = \sum c_k \delta_{Z_k}$ where Z_k is Martin-Löf random and $0^{n_k} \prec Z_k$ for a sequence $\langle c_k\rangle$ of reals in $[0,1]$ that add up to 1, and a sufficiently fast growing computable sequence $\langle n_k\rangle$ to be determined below. Then μ is Martin-Löf a.c. by Fact 7.

For n such that $n_k \leq n < n_{k+1}$ we have

$$\begin{aligned} K(\mu\upharpoonright_n) &\leq^+ (\sum_{l=0}^{k} c_l)\cdot(n+2\log n) + (\sum_{l=k+1}^{\infty} c_l)\cdot 2\log n \\ &\leq^+ (1-c_{k+1})n + 2\log n. \end{aligned}$$

Hence, to achieve $K(\mu\upharpoonright_n) \leq^+ n - n^\theta$ it suffices to ensure that $c_{k+1}n_k \geq n_{k+1}^\theta + 2\log n_{k+1}$ for almost all k. For instance, we can let $c_k = \frac{1}{(k+1)(k+2)}$ and $n_k = 2^{k+4}$. ◀

To disprove the converse implication, we need to provide a measure μ such that $K(\mu\upharpoonright_n) \geq^+ n$ yet μ is not Martin-Löf a.c. This will be immediate from the following fact on the growth of initial segment complexity for certain bit sequences.

▶ **Theorem 13.** *There are a Martin-Löf random bit sequence X and a non-Martin-Löf random bit sequence Y such that, for all n, $K(X\upharpoonright_n) + K(Y\upharpoonright_n) \geq^+ 2n$.*

Proof. Let X be a low Martin-Löf random set (i.e., $X' \equiv_T \emptyset'$). We claim that there is a strictly increasing function f such that the complement of the range of f is a recursively enumerable set E, and $K(X\upharpoonright_m) \geq m + 3n$ for all $m \geq f(n)$. To see this, recall that $\lim_n K(X\upharpoonright_n) - n = \infty$. Since X is low there is a computable function p such that for all n, $\lim_s p(n,s)$ is the maximal m such that $K(X\upharpoonright_m) \leq m + 3n$.

Define $f(n,s)$ for $n \leq s$ as follows. $f(n,0) = n$; for $s > 0$ let n be least such that $p(n,s) \geq f(n,s-1)$ or $n = s$. If $m \geq n$ and $n < s$ then let $f(m,s) = s + m - n$ else let $f(m,s) = f(m,s-1)$.

Note that for each n there are only finitely many $s > 0$ with $f(n,s) \neq f(n,s-1)$ and that almost all s satisfy $f(n,s) > p(n,s)$, as otherwise $f(n,s)$ would be modified either at n or some smaller value. Furthermore, $f(n,s) \neq f(n,s-1)$ can only happen if there is an $m \leq n$ with $f(m,s-1) \leq p(m,s)$ and that happens only finitely often, as all the $p(m,s)$ converge to a fixed value and every change of an $f(m,s)$ at some time s leads to a value above s. Furthermore, once an element is outside the range of f, it will never return, and so the complement of the range of f is recursively enumerable. So $f(n) = \lim_s f(n,s)$ is a function as required, which verifies the claim. (The complement E of the range of f is called a Dekker deficiency set in the literature [21].)

Now let $g(n) = \max\{m : f(m) \leq n\}$ (with the convention that $\max(\emptyset) = 0$). Since g is unbounded, by a result of Miller and Yu [15, Cor. 3.2] there is a Martin-Löf random Z such that there exist infinitely many n with $K(Z\upharpoonright_n) \leq n + g(n)/2$; note that the result of Miller and Yu does not make any effectivity requirements on g. Let

$$Y = \{n + g(n) : n \in Z\}.$$

Note that $K(Z\upharpoonright_n) \leq K(Y\upharpoonright_n) + g(n) + K(g(n))$, as one can enumerate the set E until there are, up to n, only $g(n)$ many places not enumerated and then one can reconstruct $Z\upharpoonright_n$ from $Y\upharpoonright_n$ and $g(n)$ and the last $g(n)$ bits of $Z\upharpoonright_n$. As Z is Martin-Löf random, $K(Z\upharpoonright_n) \geq^+ n$, so

$$K(Y\upharpoonright_n) \geq^+ n - g(n) - K(g(n)) \geq^+ n - 2g(n).$$

The definitions of X, f, g give $K(X\upharpoonright_n) \geq n + 3g(n)$. This shows that $K(X\upharpoonright_n) + K(Y\upharpoonright_n) \geq 2n$ for almost all n.

However, the set Y is not Martin-Löf random, as there are infinitely many n such that $K(Z\upharpoonright_n) \leq^+ n + g(n)/2$. Now $Y\upharpoonright_{n+g(n)}$ can be computed from $Z\upharpoonright_n$ and $g(n)$, as one needs only to enumerate E until the $g(n)$ nonelements of E below n are found and they allow to see

where the zeroes have to be inserted into the string $Z\upharpoonright_n$ in order to obtain $Y\upharpoonright_{n+g(n)}$. Note furthermore, that $K(g(n)) \leq g(n)/4$ for almost all n and thus $K(Y\upharpoonright_{n+g(n)}) \leq^+ n+3/4\cdot g(n)$ for infinitely many n, so Y cannot be Martin-Löf random. ◀

3.3 Failing a restricted type of test implies non-complex initial segments

We say that a Solovay test $\langle S_r\rangle_{r\in\mathbb{N}}$ is *strong* if each S_r is clopen and given by a strong index for a finite set of strings X_r such that $[X_r]^{\prec} = S_r$. For bit sequences, this means no restriction: any Solovay test can be replaced by a strong Solovay test listing the strings making up the Σ^0_1 sets one-by-one. (We conjecture that this equivalence of test notions no longer holds for measures.) The following is a weak version for measures of one implication of the Miller-Yu theorem [14, Thm. 7.1.]. We use elements of the proof in Bienvenu et al. [1]. We note that the result is a variation on (the contrapositive of) Theorem 10(a), proving a stronger conclusion from a stronger hypothesis.

▶ **Proposition 14.** *Suppose that μ fails a strong Solovay test $\langle S_r\rangle_{r\in\mathbb{N}}$ at level δ, namely $\exists^\infty r\,[\mu(S_r)\geq\delta]$. Then there is a computable function f such that $\sum_n 2^{-f(n)}<\infty$ and*

$$\exists^\infty n\,[C(\mu\upharpoonright_n|\,n) \leq^+ n-\delta f(n)].$$

Proof. Let $\langle X_r\rangle$ be as in the definition of a strong Solovay test. We may assume that $\sum \lambda S_r \leq 1/2$, and all strings in X_r have the same length n_r. Let f be computable such that

$$2^{-f(n_r)} \geq \lambda(S_r) > 2^{-f(n_r)-1}$$

and $f(m)=2^{-m}$ for m not of the form n_r. There is a constant d such that each bit string x in the set X_r satisfies (where $n=n_r$)

$$C(x\mid n) \leq n-f(n)+d =: g(r). \tag{1}$$

For, r can be computed from $n=n_r$, and each string $x\in X_r$ is determined by r and its position $i<2^n\lambda(X_r)$ in the lexicographical listing of X_r. We can determine i by $\log(2^n\lambda(X_r))\leq n-f(n)+d$ bits for some fixed d. In fact we may assume the description has exactly that many bits. Thus, there is a Turing machine L with two inputs such that for each $\sigma\in X_r$, we have $L(v_\sigma;n)=\sigma$ for some bit string v_σ of length $g(r)$.

Let c be a constant such that $C(x)\leq |x|+c$ for each string x. Now suppose that $\mu(S_r)\geq\delta$. Then for $n=n_r$,

$$\begin{aligned} C(\mu\upharpoonright_n|\,n) &\leq^+ \sum_{\sigma\in X_r}(n-f(n))\mu[\sigma]+\sum_{\sigma\notin X_r}(n+c)\mu[\sigma]\\ &\leq^+ n-f(n)\sum_{\sigma\in X_r}\mu[\sigma]\\ &\leq^+ n-\delta f(n).\end{aligned}$$

Since there are infinitely many such r by hypothesis, this completes the proof. ◀

4 K-triviality for measures

▶ **Definition 15.** *A measure μ is called K-trivial if $K(\mu\upharpoonright_n)\leq^+ K(n)$ for each n.*

For Dirac measures δ_A this is the same as saying that A is K-trivial in the usual sense. More generally, any finite convex combination of such Dirac measures is K-trivial.

▶ **Proposition 16.** *If μ is K-trivial, then μ is supported by its set of atoms.*
In fact the weaker hypothesis that $\exists p \exists^\infty n\, [K(\mu\restriction_n) \leq K(n) + p]$ suffices.

Proof. For a set $R \subseteq \{0,1\}^*$, by $[R]^\prec$ one denotes the open set $\{Z\colon \exists n\, Z\restriction_n \in R\}$.

Assume for a contradiction that μ gives a measure greater than $\epsilon > 0$ to the set of its non-atoms. Note that there is a constant b such that $K(x) \geq K(|x|) - b$ for each x. Fix c arbitrary with the goal of showing that $K(\mu\restriction_n) \geq K(n) - b + \epsilon c/2$ for large enough n.

There is d (in fact $d = O(2^c)$) such that for each n there are at most d strings x of length n with $K(x) \leq K(n) + c$ (see e.g. [17, 2.2.26]). Let $S_n = \{x\colon |x| = n \wedge \mu[x] \leq \epsilon/2d\}$. By hypothesis we have $\mu[S_n]^\prec \geq \epsilon$ for large enough n. Therefore by choice of d we have

$$\mu[S_n \cap \{x\colon K(x) > K(|x|) + c\}]^\prec \geq \epsilon/2.$$

Now we can give a lower bound for the μ-average of $K(x)$ over all strings x of length n:

$$\sum_{|x|=n} K(x)\mu[x] \geq (1 - \epsilon/2)(K(n) - b) + (\epsilon/2)(K(n) + c) \geq K(n) - b + \epsilon c/2,$$

as required. Notice that we have only used the weaker hypothesis. ◀

Thus, if μ is K-trivial for constant p, then μ has the form $\sum_{r<N} \alpha_r \delta_{A_r}$ where $N \leq \infty$ and each α_r is positive and $\sum_{r<N} \alpha_r = 1$.

▶ **Fact 17.** *Each A_r is K-trivial for constant $(p + c)/\alpha_r$, for some fixed c.*

Proof. Let c be a constant such that $K(x) \geq K(|x|) - c$ for each string x. We have

$$K(n) + p \geq K(\mu\restriction_n) = \sum_s \alpha_s K(A_s\restriction_n) \geq \alpha_r K(A_r\restriction_n) + \sum_{s\neq r} \alpha_s (K(n) - c).$$

Therefore $\alpha_r K(n) + p + c \geq \alpha_r K(A\restriction_n)$, as required. ◀

It would be interesting to characterise the countable convex combinations of K-trivials that yield K-trivial measures. The following is easily checked.

▶ **Fact 18.** *Suppose that A_r is K-trivial with constant b_r, and $\sum_r \alpha_r b_r \leq c < \infty$ where each α_r is positive and $\sum \alpha_r = 1$. Then $\mu = \sum_r \alpha_r \delta_{A_r}$ is K-trivial with constant c.*

For instance, we can build a computable K-trivial measure with infinitely many atoms as follows. Let $A_r = 0^{r+1}1^\infty$, so that $K(A_r\restriction_n) \leq^+ K(n) + 2\log r$. Let $\mu = \sum 2^{-r+1} A_r$. By the above fact μ is K-trivial. If we vary the construction by letting $A_r = 0^{r+1}1B$ where B is K-trivial but non-recursive, we obtain a K-trivial measure μ with infinitely many atoms, and none of them recursive.

On the other hand, the following example shows that not every infinite convex combination $\mu = \sum_k \alpha_k \delta_{A_k}$ of K-trivial Dirac measures for constant b_k yields a K-trivial measure, even if $\alpha_k b_k$ is bounded. Let $A_k = \{\ell : \ell \in \Omega \wedge \ell < k\}$, and $\alpha_k = (k+1)^{-1/2} - (k+2)^{-1/2}$. All sets A_k are finite and thus K-trivial for constant $2k + O(1)$. Furthermore, the sum of all α_k is 1 and $\alpha_k = O(1/k)$. We have

$$K(\mu\restriction_n) = \sum_{|x|=n} K(x)\mu(x) \geq (\sum_{m\geq n} \alpha_m) \cdot K(\Omega \upharpoonright n) \geq (n+2)^{-1/2} \cdot (n+2) = \sqrt{n+2}$$

for almost all n, and thus the average grows faster than $K(n) + c$. So the measure is not K-trivial.

In a sense, an atomless measure can come arbitrarily close to being K-trivial.

▶ **Proposition 19.** *For each nondecreasing unbounded function f which is computably approximable from above there is a non-atomic measure μ such that $K(\mu\restriction_n) \leq^+ K(n) + f(n)$.*

Proof. There is a recursively enumerable set A such that, for all n, $A \cap \{0, \dots, n\}$ has up to a constant $f(n)/2$ non-elements. One lets μ be the measure such that $\mu(x) = 2^{-m}$ in the case that all ones in x are not in A and $\mu(x) = 0$ otherwise, where m is the number of non-elements of A below $|x|$. One can see that when $\mu(x) = 2^{-m}$ then x can be computed from $|x|$ and the string $b_0 b_1 \dots b_{m-1}$ which describes the bits at the non-elements of A. Thus

$$K(x) \leq^+ K(|x|) + K(b_0 b_1 \dots b_{m-1}) \leq^+ K(|x|) + 2m.$$

It follows that $K(\mu\restriction_n) \leq^+ K(n) + f(n)$, as the μ-average of strings $x \in \{0,1\}^n$ with $K(x) \leq^+ K(n) + f(n)$ is at most $K(n) + f(n)$ plus a constant. ◀

5 Full Martin-Löf randomness of measures

Let $\mathcal{M}(\{0,1\}^{\mathbb{N}})$ be the space of probability measures on Cantor space (which is canonically a compact topological space). A probability measure $\mathbb{P}$ on this space has been introduced implicitly in Mauldin and Monticino [12]. Culver's thesis [4] shows that this measure is computable. So the framework of [8] yields a notion of Martin-Löf randomness for points in the space $\mathcal{M}(\{0,1\}^{\mathbb{N}})$.

To define $\mathbb{P}$, first let $\mathcal{R}$ be the closed set of representations of probability measures; namely, $\mathcal{R}$ consists of the functions $X\colon \{0,1\}^* \to [0,1]$ such that $X_\emptyset = 1$ and $X_\sigma = X_{\sigma 0} + X_{\sigma 1}$ for each string σ. $\mathbb{P}$ is the unique measure on $\mathcal{R}$ such that for each string σ and $r, s \in [0,1]$, we have $\mathbb{P}(X_{\sigma 0} \leq r \mid X_\sigma = s) = \min(1, r/s)$. Intuitively, we choose $X_{\sigma 0}$ at random w.r.t. the uniformly distribution on the interval $[0, X_\sigma]$, and the choices made at different strings are independent.

▶ **Proposition 20.** *Every probability measure μ that is Martin-Löf random wrt to $\mathbb{P}$ is Martin-Löf absolutely continuous.*

For the duration of this proof let μ range over $\mathcal{M}(\{0,1\}^{\mathbb{N}})$. For an open set $G \subseteq \{0,1\}^{\mathbb{N}}$, let

$$r_G = \int \mu(G) d\mathbb{P}(\mu).$$

Our proof of Prop. 20 is based on two facts.

▶ **Fact 21.** $r_G = \lambda(G)$.

Proof. Clearly, for each n we have

$$\sum_{|\sigma|=n} r_{[\sigma]} = \int \sum_{|\sigma|=n} \mu([\sigma]) d\mathbb{P}(\mu) = 1.$$

Furthermore, $r_\sigma = r_\eta$ whenever $|\sigma| = |\eta| = n$ because there is a $\mathbb{P}$-preserving transformation T of $\mathcal{M}(\{0,1\}^{\mathbb{N}})$ such that $\mu([\sigma]) = T(\mu)([\eta])$. Therefore $r_{[\sigma]} = 2^{-|\sigma|}$.

If σ, η are incompatible then $r_{[\sigma]\cup[\eta]} = r_{[\sigma]} + r_{[\eta]}$. Now it suffices to write $G = \bigcup_i [\sigma_i]$ where the strings σ_i are incompatible, so that $\lambda G = \sum_i 2^{-|\sigma_i|}$. ◀

▶ **Fact 22.** *Let $\mu \in \mathcal{M}(\{0,1\}^{\mathbb{N}})$ and let $\langle G_m \rangle_{m\in\mathbb{N}}$ be a ML-test such that there is $\delta \in \mathbb{Q}^+$ with $\forall m\, \mu(G_m) > \delta$. Then μ is not ML-random w.r.t. $\mathbb{P}$.*

Proof. Observe that by the foregoing fact

$$\delta \cdot \mathbb{P}(\{\mu\colon \mu(G_m) \geq \delta\}) \leq \int \mu(G_m) d\mathbb{P}(\mu) = \lambda(G_m) \leq 2^{-m}.$$

Let $\mathcal{G}_m = \{\mu\colon \mu(G_m) > \delta\}$ which is uniformly effectively open in the space of measures $\mathcal{M}(\{0,1\}^{\mathbb{N}})$. Fix k such that $2^{-k} \leq \delta$. By the inequality above, we have $\mathbb{P}(\mathcal{G}_m) \leq 2^{-m}/\delta \leq 2^{-m+k}$. Hence $\langle \mathcal{G}_{m+k} \rangle_{m\in\mathbb{N}}$ is a ML-test w.r.t. $\mathbb{P}$ that succeeds on μ. ◀

This argument also works for randomness notions stronger than Martin-Löf's. For instance, if there is a weak-2 test $\langle G_m \rangle_{m\in\mathbb{N}}$ such that $\mu G_m > \delta$ for each m, then μ is not weakly 2-random with respect to $\mathbb{P}$. The converse of Prop. 20 fails. Culver [4] shows that each measure μ that is Martin-Löf random w.r.t. $\mathbb{P}$ is non-atomic. So a measure δ_Z for a Martin-Löf random bit sequences Z is Martin-Löf a.c. but not Martin-Löf random with respect to $\mathbb{P}$.

6 Being ML-a.c. relative to computable ergodic measures

We review some notions from the field of symbolic dynamics, a mathematical area closely related to Shannon information theory. We will consider effective "almost-everywhere theorems" related to that area in the framework of randomness for measures.

It can be useful to admit alphabets other than the binary one. Let $\mathbb{A}^\infty$ denote the topological space of one-sided infinite sequences of symbols in an alphabet $\mathbb{A}$. Randomness notions etc. carry over from the case of $\mathbb{A} = \{0,1\}$. A dynamics on $\mathbb{A}^\infty$ is given by the shift operator T, which erases the first symbol of a sequence. A measure ρ on $\mathbb{A}^\infty$ is called *shift invariant* if $\rho(G) = \rho(T^{-1}(G))$ for each open (and hence each measurable) set G. The *empirical entropy* of a measure ρ along $Z \in \mathbb{A}^\infty$ is given by the sequence of random variables

$$h^\rho_n(Z) = -\frac{1}{n} \log_{|\mathbb{A}|} \rho[Z\restriction_n].$$

A shift invariant measure ρ on $\mathbb{A}^\infty$ is called *ergodic* if every ρ integrable function f with $f \circ T = f$ is constant ρ-almost surely. The following equivalent condition can be easier to check: for any strings $u, v \in \mathbb{A}^*$,

$$\lim_N \frac{1}{N} \sum_{k=0}^{n-1} \rho([u] \cap T^{-k}[v]) = \rho[u]\rho[v].$$

For ergodic ρ, the entropy $H(\rho)$ is defined as $\lim_n H_n(\rho)$, where

$$H_n(\rho) = -\frac{1}{n} \sum_{|w|=n} \rho[w] \log \rho[w].$$

Thus, $H_n(\rho) = \mathbb{E}_\rho h^\rho_n$ is the expected value with respect to ρ. One notes that $H_{n+1}(\rho) \leq H_n(\rho) \leq 1$ so that the limit $H(\rho)$ exists.

A well-known result from the 1950s due to Shannon, McMillan and Breiman states that for an ergodic measure ρ, for ρ-a.e. Z the empirical entropy along Z converges to the entropy of the measure. See e.g. [24], but note that the result is called the Entropy Theorem there.

▶ **Theorem 23** (SMB theorem, e.g. [24]). *Let ρ be an ergodic measure on the space $\mathbb{A}^\infty$. For ρ-almost every Z we have $\lim_n h^\rho_n(Z) = H(\rho)$.*

A measure ρ on $\mathbb{A}^\infty$ is called *computable* if each real $\rho[x]$ is computable, uniformly in $x \in \mathbb{A}^*$. For such a measure we can define Martin-Löf tests and Martin-Löf randomness with respect to ρ (called ρ-ML randomness for short) as above. Recall from Fact 4 that a measure μ is *Martin-Löf a.c. with respect to* ρ iff $\mu(\mathcal{C}) = 0$ where $\mathcal{C}$ is the class of sequences in $\mathbb{A}^\infty$ that are not ML-random with respect to ρ.

If a computable measure ρ is shift invariant, then $\lim_n h^\rho_n(Z)$ exists for each ρ-ML-random Z by a result of Hochman [6]. Hoyrup [7, Thm. 1.2] gave an alternative proof for ergodic ρ, and also showed that in that case we have $\lim_n h^\rho_n(Z) = H(\rho)$ for each ρ-ML random Z. We extend this result to measures μ that are Martin-Löf a.c. with respect to ρ, under the additional hypothesis that the h^ρ_n are uniformly bounded. This holds e.g. for Bernoulli measures and the measures given by a Markov process. On the other hand, using a renewal process it is not hard to construct an ergodic computable measure ρ over the binary alphabet where this hypothesis fails. For instance, take a computable sequence of rationals $\langle \alpha_k \rangle$ with sum 1 which decreases quickly enough such that $\lim_k -\frac{1}{k+2} \log_2 \alpha_k = \infty$, and let ρ be a shift invariant measure such that $\rho(10^k 1 \prec Z \mid Z_0 = 1) = \alpha_k$ for each $k \in \mathbb{N}$. This method yields an example showing that the boundedness hypothesis in the proposition below is necessary. See the Logic Blog 2020 posted from Nies' website.

▶ **Proposition 24.** *Let ρ be a computable ergodic measure on the space $\mathbb{A}^\infty$ such that for some constant D, each h^ρ_n is bounded above by D. Suppose the measure μ is Martin-Löf a.c. with respect to ρ. Then $\lim_n E_\mu h^\rho_n = H(\rho)$.*

Proof. By Hoyrup's result, $\lim_n h^\rho_n(Z) = H(\rho)$ for each ρ-ML random Z. Since the sequences that are not ML-random w.r.t. ρ form a null set w.r.t. μ, we infer that $\lim_n h^\rho_n(Z) = H(\rho)$ for μ-a.e. Z. The exception set V is measurable. Let $\widetilde{h}^\rho_n$ be the function obtained by changing the value of h^ρ_n to 0 on this set. Then $\widetilde{h}^\rho_n(Z) \to H(\rho) 1_{(\mathbb{A}^\infty \setminus V)(Z)}$ for each Z. The Dominated Convergence Theorem now shows that $\lim_n \mathbb{E}_\mu h^\rho_n = H(\rho)$, as required. ◀

The next observation shows that the asymptotic initial segment complexity of a ML-a.c. measure relative to ρ obeys some lower bound. Note that $H(\lambda) = 1$. So for $\rho = \lambda$, this shows that in Example 12 we cannot subtract, say, $n/4$ instead of n^θ.

▶ **Proposition 25.** *Let ρ be a computable ergodic measure, and suppose μ is a Martin-Löf a.c. measure with respect to ρ. Then*

$$\lim_n \frac{1}{n} K(\mu \upharpoonright_n) = \lim_n \frac{1}{n} C(\mu \upharpoonright_n) = H(\rho).$$

Proof. We can use K and C interchangeably because $C(x) \leq^+ K(x) \leq^+ C(x) + K(C(x))$ [17, 2.4.1]. We choose K. Let $k_n(Z) = K(Z \upharpoonright_n)/n$. The argument is very similar to the one in the theorem above, replacing the functions h_n by k_n. Note that k_n is bounded above by a constant because $K(x) \leq^+ |x| + 2 \log |x|$. Hoyrup's result [7, Thm. 1.2] states that $\lim_n k_n(Z) = H(\rho)$ for each ρ-ML random Z. Now we can apply the Dominated Convergence Theorem as in the proof of the foregoing proposition. ◀

A further interesting direction to tackle in the measure case would be the effective Birkhoff's ergodic theorem. This says that for ergodic computable ρ, if $f\{0,1\}^\mathbb{N} \to \mathbb{R}$ is ρ-integrable and lower semicomputable and Z is ρ-ML-random, then the limit of the usual ergodic averages $A_n f(Z) = \frac{1}{n} \sum_{k<n} (f \circ T^k)(Z)$ equals $\int f d\rho$. (For background see e.g. [16] which contains references to original work.) If the $A_n f$ are bounded then an argument similar to the one above shows that $\lim_n \int A_n f d\mu = \int f d\rho$ for any $\mu \ll_{ML} \rho$, but without this additional hypothesis the question remains open.

References

1 Laurent Bienvenu, Wolfgang Merkle, and Alexander Shen. A simple proof of the Miller-Yu Theorem. *Fundamenta Informaticae*, 83(1-2):21–24, 2008.

2 Cristian Calude. *Information and randomness*. Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 1994. With forewords by Gregory J. Chaitin and Arto Salomaa.

3 Gregory J. Chaitin. A theory of program size formally identical to information theory. *J. Assoc. Comput. Mach.*, 22:329–340, 1975.

4 Quinn Culver. *Topics in Algorithmic Randomness and Effective Probability*. PhD thesis, PhD thesis, University of Notre Dame, 2015.

5 Rodney G. Downey and Denis Hirschfeldt. *Algorithmic randomness and complexity*. Springer-Verlag, Berlin, 2010. 855 pages.

6 Michael Hochman. Upcrossing inequalities for stationary sequences and applications. *Annals of Probability*, 37(6):2135–2149, 2009.

7 Mathieu Hoyrup. The dimension of ergodic random sequences. In Christoph Dürr and Thomas Wilke, editors, *STACS*, pages 567–576, 2012.

8 Mathieu Hoyrup and Cristóbal Rojas. Computability of probability measures and Martin-Löf randomness over metric spaces. *Information and Computation*, 207(7):830–847, 2009.

9 Leonid A. Levin. The concept of a random sequence. *Dokl. Akad. Nauk SSSR*, 212:548–550, 1973.

10 Ming Li and Paul Vitányi. *An introduction to Kolmogorov complexity and its applications*. Graduate Texts in Computer Science. Springer-Verlag, New York, second edition, 1997.

11 Per Martin-Löf. The definition of random sequences. *Information and Control*, 9:602–619, 1966.

12 Daniel Mauldin and Michael Monticino. Randomly generated distributions. *Israel Journal of Mathematics*, 91(1-3):215–237, 1995.

13 Joseph S. Miller. The K-degrees, low for K-degrees, and weakly low for K sets. *Notre Dame J. Form. Log.*, 50(4):381–391, 2009. doi:10.1215/00294527-2009-017.

14 Joseph S. Miller and Liang Yu. On initial segment complexity and degrees of randomness. *Trans. Amer. Math. Soc.*, 360:3193–3210, 2008.

15 Joseph S. Miller and Liang Yu. Oscillation in the initial segment complexity of random reals. *Advances in Mathematics*, 226(6):4816–4840, 2011.

16 Kenshi Miyabe, André Nies, and Jing Zhang. Using almost-everywhere theorems from analysis to study randomness. *Bull. Symb. Logic*, 22:305–331, 2016. arXiv:1411.0732.

17 André Nies. *Computability and Randomness*, volume 51 of *Oxford Logic Guides*. Oxford University Press, Oxford, 2009. 444 pages. Paperback version 2011. doi:10.1093/acprof:oso/9780199230761.001.0001.

18 André Nies and Volkher Scholz. Martin-Löf random quantum states. *Journal of Mathematical Physics*, 60(9):092201, 2019. Available at doi.org/10.1063/1.5094660.

19 André Nies, Frank Stephan, and Sebastiaan Terwijn. Randomness, relativization and Turing degrees. *J. Symbolic Logic*, 70(2):515–535, 2005.

20 Editor André Nies. Logic Blog 2017. Available at arXiv:1804.05331, 2017.

21 Piergiorgio Odifreddi. *Classical Recursion Theory (Volume I)*. North–Holland Publishing Co., Amsterdam, 1989.

22 Piergiorgio Odifreddi. *Classical recursion theory: The theory of functions and sets of natural numbers*, volume 125. Elsevier, 1992.

23 Claus-Peter Schnorr. Process complexity and effective random tests. *J. Comput. System Sci.*, 7:376–388, 1973. Fourth Annual ACM Symposium on the Theory of Computing (Denver, Colo., 1972).

24 Paul C. Shields. *The Ergodic Theory of Discrete Sample Paths*. Graduate Studies in Mathematics 13. American Mathematical Society, 1996.

25 Robert I. Soare. *Recursively Enumerable Sets and Degrees*. Perspectives in Mathematical Logic, Omega Series. Springer–Verlag, Heidelberg, 1987.

26 Frank Stephan. Recursion theory. Lecture notes, School of Computing, 2012.

Computing Shrub-Depth Decompositions

Jakub Gajarský
Technical University Berlin, Germany
jakub.gajarsky@tu-berlin.de

Stephan Kreutzer
Technical University Berlin, Germany
stephan.kreutzer@tu-berlin.de

Abstract

Shrub-depth is a width measure of graphs which, roughly speaking, corresponds to the smallest depth of a tree into which a graph can be encoded. It can be thought of as a low-depth variant of clique-width (or rank-width), similarly as treedepth is a low-depth variant of treewidth. We present an fpt algorithm for computing decompositions of graphs of bounded shrub-depth. To the best of our knowledge, this is the first algorithm which computes the decomposition directly, without use of rank-width decompositions and FO or MSO logic.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Mathematics of computing → Combinatorial algorithms

Keywords and phrases shrub-depth, tree-model, decomposition, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.56

Funding The research is supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC consolidator grant DISTRUCT, agreement No. 648527).

Acknowledgements We want to thank Sang-il Oum for suggesting that our techniques may be used to obtain the results of Section 6.

1 Introduction

Among the numerous width parameters used in graph theory and algorithmics, treewidth and its dense counterparts clique-width and rank width are arguably the most prominent and extensively studied. In recent years, more restrictive parameters, which could be collectively called depth parameters, are attracting increasing attention. The best-known of these is treedepth [10], which can be seen as a low-depth variant of treewidth. Inspired by the usefulness of treedepth, the authors of [8] defined the notion of shrub-depth, which can be seen as a low-depth variant of clique-width, analogously to the relation between treedepth and treewidth.

Since shrub-depth is a more restrictive notion than clique-width, it is natural to ask what algorithmic advantages it offers over clique-width, if any. The question whether some problems which are parameterized intractable on graphs of bounded clique-width are fixed-parameter tractable on graphs of bounded shrub-depth was addressed in [7], where it was shown that the Hamiltonian path and the chromatic number problem remain hard on graph classes of bounded shrub-depth. On the other hand, bounded shrub-depth offers certain quantitative advantages over clique-width. For instance, a well-known result of Courcelle, Makowski and Rotics [2] states that every MSO definable property φ of graphs can be solved in time $f(\varphi) \cdot |V(G)|$ on any class of graphs of bounded clique-width. The price for the generality of this result is that the function f is non-elementary, i.e. it grows like a tower of exponentials whose height depends on the formula. But as shown in [5], on any graph class shrub-depth d the function f is only d-fold exponential, i.e. its height does not depend on the formula.

Besides being interesting in their own right, shrub-depth has important consequences for much more general graph classes. To explain this, we will first describe the analogous situation in the case of sparse graphs and treedepth. Nešetřil and Ossona de Mendez introduced [11] the notions of bounded expansion and nowhere denseness as a general approach to studying sparse graphs in a unified setting. Graphs from classes of bounded expansion and nowhere dense graph classes can be decomposed[1] into several overlapping graphs each of small treedepth. Such a decomposition has successfully been used for solving many problems on graph classes of bounded expansion and nowhere dense graph classes efficiently. In some cases, the problems can be reduced to solving them on the graphs of small treedepth obtained by this decomposition, and in more complicated cases one can repeatedly perform computations on graphs of small treedepth from the decomposition and then combine the results.

Recently a dense counterpart of the notion of bounded expansion has been studied under the name *structurally bounded expansion*, and the results of [6] indicate that shrub-depth could play a role analogous to treedepth in the sparse case: it was shown that every graph from a class of graphs of structurally bounded expansion can be decomposed into a bounded number of subgraphs of small shrub-depth and this decomposition can then be used for algorithmic purposes. Given this promising application of shrub-depth in the theory of dense but structurally simple graphs, it is very likely that a better structural understanding of shrub-depth will be of increasing relevance in the near future.

All width and depth measures mentioned so far are defined by an associated concept of decomposition. For shrub-depth this decomposition is called a *tree-model*. Unlike other width measures, tree-models are defined in terms of two parameters, commonly denoted by d and m. To use any of the mentioned depth or width measures algorithmically, one usually needs to be able to compute the corresponding types of decompositions for an input graph G. In most cases, the problem of finding an optimal decomposition of an input graph is NP-hard, but often for fixed values of the relevant width parameters one can, in polynomial time, either find a decomposition with the prescribed value or correctly decide that no such decomposition exists. This is the case for treewidth [1], rank-width [9] and treedepth [12]. Algorithms like this are called *parameterized algorithms* and are studied in the framework of parameterized complexity theory. We refer to [3] for an indepth introduction to parameterized complexity and only briefly recall the concepts from parameterized complexity needed below. A *parameterized problem* P is essentially a classical problem but in addition to the normal input instance w we are given an integer k, the so-called *parameter*. The problem P is called *fixed-parameter tractable*, or in the complexity class FPT, if there is a computable function f and a constant c such that the problem can be solved by an algorithm whose running time on input (w, k) is bounded by $f(k) \cdot |w|^c$. The class FPT can be seen as the parameterized equivalent to the classical complexity class P as abstraction of efficiently solvable problems.

A much weaker requirement on the running time is imposed by the parameterized complexity class XP. The problem P is in XP if there is a computable function f such that the problem can be solved by an algorithm whose running time is bounded by $|G|^{f(k)}$. Thus, a problem is in XP if it can be solved in polynomial time for every fixed value of the parameter k.

Our contribution. We provide combinatorial and conceptually simple algorithms for computing tree-models with given parameters d and m of input graphs G. To obtain our algorithms, in Section 3 and 4, we introduce a new concept of k-modules and prove several

[1] The precise meaning of this is rather technical and is not important for our purpose, so we omit the precise definition.

properties relating k-modules and tree-models of graphs. The structural results we obtain provide a new and very different insight into the structure of graphs of low shrub-depth which we believe will be of further interest. These results provide the basis for our algorithms for computing tree-models, which we present in Section 5. Our main algorithmic result is an fpt-algorithm for computing tree-models with prescribed parameters d and m in a graph G, provided G has such a model. Finally, in Section 6, we present another application of our results to forbidden induced subgraphs of graph classes of bounded shrub-depth.

Previous work. To the best of our knowledge, no papers explicitly address the problem of computing tree-models with given parameters d and m of a given input graph. However, there exists a folklore fpt algorithm for computing an optimal SC-decomposition of a given graph, where SC-decomposition is a notion closely related to tree-model. This algorithm requires computing a rank decomposition of the input graph G first and then uses a powerful algorithmic metatheorem for MSO logic to obtain the SC-decomposition. Similarly, the results and techniques of [5] can likely be adjusted to compute tree-models, but also in this case one would have to rely on computing a rank decomposition first and using a logical metatheorem.

2 Preliminaries

For $n \in \mathbb{N}$ we denote the set $\{1, \dots, n\}$ by $[n]$. For sets A, B we denote by $A \Delta B$ their symmetric difference $(A \setminus B) \cup (B \setminus A)$.

Graphs. All graphs in this paper are finite, undirected and simple. We use standard graph theoretic notation, see e.g. [4]. Let G be a graph. If $X \subseteq V(G)$ we denote by $G[X]$ the subgraph of G induced by X and by $G - X$ the subgraph induced by $V \setminus X$. If $X = \{v\}$ is a singleton set, we simply write $G - v$ for $G - \{v\}$. We denote by $N_G(v)$ the *neighbourhood* of $v \in V(G)$ in G. If G is understood we omit the index and just write $N(v)$. Let G, H be graphs and let $\lambda : V(G) \to [m]$ and $\lambda' : V(H) \to [m]$ be labelling functions. A *label preserving isomorphism* between (G, λ) and (H, λ') is a bijective function $f : V(G) \to V(H)$ such that $\{u, v\} \in E(G)$ if, and only if, $\{f(u), f(v)\} \in E(H)$ and $\lambda'(f(v)) = \lambda(v)$ for all $u, v \in V(G)$.

Trees. By a tree in this paper we mean a rooted connected acyclic graph. Let T be a tree with root r. The *ancestors* of t in T are the vertices on the unique path from r to t in T other than t itself. The *parent* of t is the ancestor of t adjacent to t. The root r itself does not have a parent. For $t \in V(T) \setminus \{r\}$ we define the *subtree* T_t *of* T *rooted at* t as the component of $T - e$ containing t, where e is the edge incident to t and its parent. For $t = r$ we set $T_r = T$. The *children* of t are the neighbours of t other than the parent. The *descendants* of t are the vertices in $V(T_t) \setminus \{t\}$.

A *leaf* of T is a node of degree 1 which is not the root. We denote the set of leaves of T by *leaves*(T). Nodes $s, t \in V(T)$ are *comparable* (in T) if $t \in V(T_s)$ or $s \in V(T_t)$. Otherwise they are *incomparable*. The *height* of t is the maximal length of a path from t to a leaf of T_t. Given a set $X \subseteq V(T)$ we define the *least common ancestor* of X, denoted by *lca*(X), as the node $t \in V(T)$ of minimal height such that $X \subseteq V(T_t)$. We also write *lca*$(u_1, \dots, u_t)$ for *lca*$(\{u_1, \dots, u_t\})$. The *distance* between two nodes $s, t \in V(T)$, denoted by $dist_T(s, t)$, is the length of the unique path between s and t in T.

Shrub-depth. Shrub-depth was defined by Ganian et al. in [8]. It is defined using the following notion of *tree-model*.

▶ **Definition 2.1.** *Let d and m be non-negative integers and let G be a graph. A* tree-model *of G is a triple (T, S, λ), where T is a tree, $S \subseteq [m]^2 \times [d]$ is a relation, and $\lambda : leaves(T) \to [m]$ is a function, such that*

- **i.** *the length of each root-to-leaf path is exactly d,*
- **ii.** *the set $leaves(T)$ of leaves is exactly $V(G)$,*
- **iii.** *$(i, j, d) \in S$ if, and only, if $(j, i, d) \in S$ (symmetry in the colours), and*
- **iv.** *for any two vertices $u, v \in V(G)$, if $\lambda(v) = i$, $\lambda(v) = j$ and $dist_T(u, v) = 2l$, then $\{u, v\} \in E(G)$ if, and only if, $(i, j, l) \in S$.*

Note that the leaves $leaves(T)$ of T are the vertices of G. Thus, if $v \in V(G)$, then $v \in leaves(T)$ and therefore $\lambda(t)$ is defined. The number d in the above definition is referred to as the depth of the tree-model. We will often speak of a (d, m)-tree-model instead of a "tree-model of depth d with m colours".

Note that every graph G has a tree-model of depth 1 with $|V(G)|$ colours (each vertex gets its own colour and the relation S is essentially $E(G)$). Thus it does not make sense to ask about the smallest depth of a tree-model of a graph G. This is the reason why the notion of *shrub-depth* is defined only for classes of graphs.

▶ **Definition 2.2.** *The* shrub-depth *of a graph class $\mathcal{C}$ is the smallest $d > 0$ for which there exists an $m > 0$ such that every graph $G \in \mathcal{C}$ has a (d, m)-tree-model.*

We remark that even though the shrub-depth of a graph class $\mathcal{C}$ is defined as one number (d in the above definition), to each class $\mathcal{C}$ of graphs of bounded shrub-depth there actually correspond two numbers – d and m from the above definition. Thus, in what follows, we usually work directly with (classes of) graphs which have a (d, m)-tree-model for some fixed d and m.

▶ **Definition 2.3.** *Let G be a graph and $G' \subseteq G$ be an induced subgraph of G. Let $d, m > 0$. A (d, m)-tree-model (T, S, λ) of G* extends *a (d, m)-tree-model (T', S', λ') of G' if $T' \subseteq T$, $S = S'$ and $\lambda(v) = \lambda'(v)$ for all $v \in V(G')$.*

We frequently use the following result from [8] which follows immediately from the definition of shrub-depth: if (T, S, λ) is a (d, m)-tree-model of G, then we can obtain a (d, m)-tree-model (T', S', λ') of G' as follows: the tree T' is the minimal subtree of T containing the root of T and all leaves form $V(G')$. Similarly, λ' is the restriction of λ to $V(G')$ and $S' = S$. In particular, the tree-structure of T is preserved in the reduced tree-model T'.

▶ **Proposition 2.4** ([8]). *Let $d, m > 0$. If G has a (d, m)-tree-model and G' is an induced subgraph of G, then G' also has a (d, m)-tree-model.*

3 Twin tuples, k-modules and outline of our approach

In this section we introduce the concepts of *twin tuples* and *(strict) k-modules* which will be pivotal in the rest of the paper and briefly outline the key idea behind our algorithm for finding a (d, m)-tree-model of an input graph G.

3.1 Twin tuples and (strict) k-modules

▶ **Definition 3.1.** *Let $k \in \mathbb{N}$ and let G be a graph. Two disjoint tuples $(a_1, \ldots, a_k)$, $(b_1, \ldots, b_k) \in V(G)^k$ are* twin tuples *if*

1. *the function $f(a_i) = b_i$, $1 \leq i \leq k$, is an isomorphism between $G[\{a_1, \ldots, a_k\}]$ and $G[\{b_1, \ldots, b_k\}]$,*
2. *$\{a_i, b_j\} \in E(G)$ if, and only if, $\{a_j, b_i\} \in E(G)$, for all $1 \leq i < j \leq k$, and*
3. *$N(a_i) \setminus \{a_1, \ldots, a_k, b_1, \ldots, b_k\} = N(b_i) \setminus \{a_1, \ldots, a_k, b_1, \ldots, b_k\}$ for all $1 \leq i \leq k$.*

For $k = 1$ we simply call a_1 and b_1 twins*. A set M of pairwise disjoint k-tuples of vertices of G is a* structured k-module *if all tuples in M are pairwise twin tuples.*

The next definition introduces a different characterisation of k-modules which we will use frequently in the sequel. The equivalence between the two definitions is easily seen (and stated formally in the lemma thereafter).

▶ **Definition 3.2.** *Let $k \in \mathbb{N}$ and let α, β be symmetric relations on $[k]^2$. Let G be a graph. A set $M \subseteq V(G)^k$ of pairwise disjoint k-tuples is an* (α, β)–module *if*

1. *for every $(a_1, \ldots, a_k) \in M$, $\{a_i, a_j\} \in E(G)$ if, and only, if $(i, j) \in \alpha$,*
2. *for all distinct tuples $a_1, \ldots, a_k, b_1, \ldots, b_k \in M$, $\{a_i, b_j\} \in E(G)$ if, and only, if $(i, j) \in \beta$, and*
3. *$N(a_i) \setminus S = N(b_i) \setminus S$ for all $1 \leq i \leq k$ and all $(a_1, \ldots, a_k), (b_1, \ldots, b_k) \in M$, where $S := \bigcup\{a_i : 1 \leq i \leq k, (a_1, \ldots, a_k) \in M\}$.*

*We call k-tuples $\bar{a}, \bar{b} \in V(G)^k$ (α, β)-*twins*, if $\{\bar{a}, \bar{b}\}$ is an (α, β)-module in G.*

▶ **Lemma 3.3.** *A set M of pairwise distinct k-tuples is a structured k-module if, and only if, there are symmetric relations α, β on $[k]^2$ such that M is an (α, β)-module.*

Thus, in a structured k-module M, the relation α determines the adjacency within each tuple $\bar{a} \in M$, or the isomorphism type of the subgraphs of G induced by the tuples in the module and the relation β fixes the adjacency between different tuples from M. Furthermore, any two vertices at the same position within their respective tuples have the same adjacency to the vertices outside the module. The notion of k-module is illustrated on Figure 1.

Given α, β as above, we say that the structured k-module M is *determined* or *induced* by α, β. Also, if $\bar{a}$ and $\bar{b}$ are twin tuples such that the adjacency within $\bar{a}$ and $\bar{b}$ and between $\bar{a}$ and $\bar{b}$ is determined by relations α and β, we say that $\bar{a}$ and $\bar{b}$ are (α, β)-twin tuples. The following simple fact will be used often in the sequel.

▶ **Lemma 3.4.** *Let $\bar{a}, \bar{b}, \bar{c} \in V(G)^k$ be tuples such that $\bar{a}$ is an (α, β)-twin tuple of $\bar{b}$ and $\bar{b}$ is an (α, β)-twin tuple of $\bar{c}$. Then $\bar{a}$ is an (α, β)-twin tuple of $\bar{c}$. In other words, for any fixed α and β the relation of being (α, β)-twin tuples is transitive.*

The next lemma captures the intuition behind k-modules and their connection to tree-models.

▶ **Lemma 3.5.** *Let (T, S, λ) be a tree-model of a graph G and let $u \in V(T)$ be a node with $L \geq 2$ children $\{1, \ldots, L\}$ such that for any pair i, j of its children there exists a label preserving isomorphism ι_{ij} between T_i and T_j. Then G contains a k-module with L tuples, where $k = |\mathit{leaves}(T_1)|$.*

Proof. Fix an ordering $\leq_1$ on $\mathit{leaves}(T_1)$. Then, for every $j > 1$, we define an order $\leq_j$ on $\mathit{leaves}(T_j)$ as follows: $u \leq_j v$, for $u, v \in \mathit{leaves}(T_1)$, if, and only if, $\iota_{j1}(u) \leq_1 \iota_{j1}(v)$. Each pair $(\mathit{leaves}(T_i), \leq_i)$ can be thought of as a k-tuple, and it is easy to see that $M := \{(\mathit{leaves}(T_1), \leq_1), \ldots, (\mathit{leaves}(T_L), \leq_L)\}$ is a k-module with L tuples as claimed in the statement of the lemma. ◀

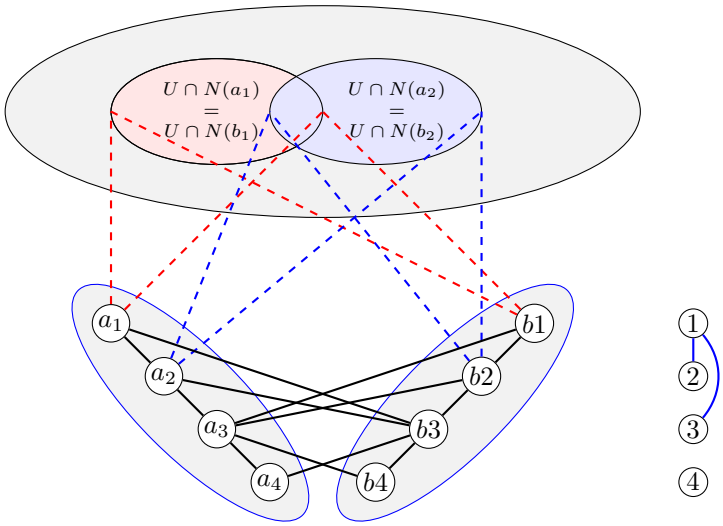


Figure 1 An example of a 4-module M with two tuples (a_1, a_2, a_3, a_4) and (b_1, b_2, b_3, b_4). The set U is $V(G) \setminus \{a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4\}$. Relations α and β are defined as follows: $\alpha = \{(1,2),(2,1),(2,3),(3,2),(3,4),(4,3)\}$, $\beta = \{(1,3),(3,1),(2,3),(3,2),(3,4),(4,3)\}$. The conflict graph of M (Definition 3.7) is on the right. Note that $M' = \{(a_1, a_2, a_3)(b_1, b_2, b_3)\}$ is also a module, and since its conflict graph (only on vertices $1, 2, 3$) is connected, it is an example of a strict module (Definition 3.8).

3.2 Outline of our approach

We now present an outline of our approach for computing tree-models. By Lemma 3.5, if a tree-model of a graph G contains several isomorphic subtrees with common parent, then these subtrees induce a k-module in G. If the converse statement

> $(*)$ if $\bar{a}$, $\bar{b}$, $\bar{c}, \ldots$, form a module M of G, then in every tree-model T of G the vertices of tuples $\bar{a}, \bar{b}, \bar{c}, \ldots$ are the leaves of distinct but isomorphic subtrees $T^a, T^b, T^c, \ldots$ with a common parent u

was also true, then this could be used to compute tree-models as follows: compute a module $M = \{\bar{a}, \bar{b}, \bar{c}\}$ in the input graph G and remove $\bar{c}$ from G to obtain G' with module $M' = \{\bar{a}, \bar{b}\}$. Then it would be enough to find *any* (d, m)-tree-model T' of G' which then can easily be extended to a tree-model T of G – by $(*)$ the tree-model T' contains different isomorphic subtrees T^a and T^b (representing $\bar{a}$ and $\bar{b}$) with a common parent u, and so we can create a copy T_c of T_a and add it as a child of u to create a tree-model T of G. This essentially means that by finding a module M in G we can, by deleting a tuple from M, reduce the problem of computing a tree-model with parameters d and m of G to a problem of finding a tree-model T' with the same parameters but for a smaller graph G'.

We will essentially follow this idea but use a weaker statement than $(*)$ instead. It turns out that even less than having isomorphic subtrees is enough to make the above idea work – it is enough to have a tree-model T' of G' in which two tuples $\bar{a}$ and $\bar{b}$ are in a "good position" with respect to each other, as shown in the following lemma, proved below, which is the basis of our approach.

▶ **Lemma 3.6.** *Let G be a graph and let $\{(a_1, \ldots, a_k), (b_1, \ldots, b_k), (c_1, \ldots, c_k)\}$ be a k-module in G. Let $G' = G - \{c_1, \ldots, c_k\}$ and let (T', S', λ') be a (d, m)-tree-model of G', for some $d, m > 0$, st.*

1. *$lca(\{a_1, \dots, a_k\})$ and $lca(\{b_1, \dots, b_k\})$ are incomparable in T' and*
2. *for all $i \in [k]$ the labels $\lambda'(a_i)$ and $\lambda'(b_i)$ are the same in T'.*

Then (T', S', λ') can be extended to a (d, m)-tree-model (T, S, λ) of G.

Unfortunately, the statement $(*)$ does not hold and neither does the following weaker statement $(**)$ which would still be strong enough for our purpose: if M is a k-module in a graph G, then in any tree model T of G there are at least two tuples $(a_1, \dots, a_k), (b_1, \dots, b_k) \in M$ which satisfy the requirements of Lemma 3.6 with respect to T. In general, if $\bar{a}$ and $\bar{b}$ are twin tuples of graph G, the vertices $(a_1, \dots, a_k)$ and $(b_1, \dots, b_k)$ can be placed almost arbitrarily "badly" in a tree-model T of G. In order to be able prove a variant of $(**)$, we will have to restrict ourselves to a more structured notion of module, which we call a *strict* module. Strict modules are modules in which the vertices in each tuple are forced to "stick together" – we want to avoid the situation when it is possible to exchange a_i for b_i between two tuples $(a_1, \dots, a_k)$ and $(b_1, \dots, b_k)$ without violating α or β. More generally, we want it to be impossible for any non-empty subset I of $[k]$ to exchange $\{a_i\}_{i \in I}$ for $\{b_i\}_{i \in I}$ between two tuples $(a_1, \dots, a_k)$ and $(b_1, \dots, b_k)$ without violating α and β. This will be accomplished using the notions of *conflict* and *conflict graph*.

▶ **Definition 3.7.** *Let M be a k-module of a graph G induced by relations α, β. We say that positions $i, j \in [k]$ with $i \neq j$ are in conflict if $(i, j) \in \alpha$ but $(i, j) \notin \beta$ or vice versa. The* conflict graph *$\mathcal{C}(M)$ of M is the graph with vertex set $[k]$ and an edge between i and j if, and only if, the positions i and j are in conflict.*

▶ **Definition 3.8.** *An (α, β)-module M is a* strict k-module *if its conflict graph is connected.*

The definition of strict k-modules will allow us to prove Lemma 4.1, which can be seen as a variant of $(**)$. Informally it says that if G is a graph with a large strict k-module M, then in any (d, m)-tree-model T of G there are two tuples in a "good" mutual position in T (i.e. in the position required in Lemma 3.6). With Lemma 4.1 at hand, it remains to bound the value k in terms of d and m and to show that sufficiently large strict k-modules always exist in graphs which have (d, m)-tree-models (Corollary 4.6). Finally, we need to design algorithms for computing large strict k-modules in an input graph G (Section 5).

We close this section by proving Lemma 3.6 and a corollary, which will be used in Section 5.

Proof of Lemma 3.6. Let v_a, v_b be the least common ancestors of $\{a_1, \dots, a_k\}$ and $\{b_1, \dots, b_k\}$, resp., and let $v = lca(v_a, v_b)$. Let T_a be the minimal subtree of T_v containing v_a and $\{a_1, \dots, a_k\}$. Thus, T_a has exactly k leaves $a_1, \dots, a_k$. Let T_d be an isomorphic copy of T_a. Let $d_1, \dots, d_k$ be the leaves of T_d such that d_i is the copy of a_i, for all $1 \leq i \leq k$.

Let T be the tree obtained from $T' \cup T_d$ by identifying the root of T_d with v, i.e. T is the tree $T' \cup S_1 \cup \ldots \cup S_l$ plus the edges $\{v, s_i\}$, for $1 \leq i \leq l$, where $S_1, \dots, S_l$ are the subtrees of T_d rooted at the children $s_1, \dots, s_l$ of the root of T_d. We set $S = S'$ and define a labelling function λ on the leaves of T by setting $\lambda(t) = \lambda'(t)$, if $t \notin \{d_1, \dots, d_k\}$ and $\lambda(d_i) = \lambda'(a_i)$.

Then (T, S, λ) is a tree-model of the same height as (T', S', λ') using the same set of labels.

It remains to verify that the graph G^T defined by T is isomorphic to G. By construction, $V(G^T) = V(G) \setminus \{c_1, \dots, c_k\} \cup \{d_1, \dots, d_k\}$. Let $\pi : V(G^T) \to V(G)$ be the function with $\pi(u) = u$ for all $u \in V(G) \setminus \{d_1, \dots, d_k\}$ and $\pi(d_i) = c_i$, for $1 \leq i \leq k$. We claim that π is an isomorphism between G^T and G.

As π is bijective by construction, it suffices to show that $\{u, w\} \in E(G^T)$ if, and only if, $\{\pi(u), \pi(w)\} \in E(G)$. If $u, w \in V(G) \setminus \{c_1, \dots, c_k\}$ there is nothing to show as the adjacency of all vertices within $G' = G - \{c_1, \dots, c_k\}$ remains unchanged by attaching $S_1, \dots, S_l$ to T'.

Now suppose $u = d_i$ and $w = d_j$, for some $1 \leq i \neq j \leq k$. Then $\lambda(d_i) = \lambda(a_i)$ and $\lambda(d_j) = \lambda(a_j)$ and the distance $2l$ between d_i and d_j in T is the same as the distance between a_i and a_j. Thus

$$\{d_i, d_j\} \in E(G^T) \Leftrightarrow (\lambda(d_i), \lambda(d_j), l) \in S \Leftrightarrow \{a_i, a_j\} \in E(G^T) \Leftrightarrow \{a_i, a_j\} \in E(G),$$

as argued above. As $\{\{a_1, \dots, a_k\}, \{c_1, \dots, c_k\}\}$ is a k-module in G, $\{c_i, c_j\} \in E(G)$ if, and only if, $\{a_i, a_j\} \in E(G)$. Thus, the restriction of π to $\{d_1, \dots, d_k\}$ is an isomorphism between the subgraphs $G[\{c_1, \dots, c_k\}]$ and $G^T[\{d_1, \dots, d_k\}]$.

The last case to consider is when $u = d_i$, for some $1 \leq i \leq k$, and $w \notin \{d_1, \dots, d_k\}$. Suppose first that $w \notin V(T_v)$, where $v = lca(v_a, v_b)$. Then the distance between w and a_i in T is the same as between w and d_i and therefore

$$\{d_i, w\} \in E(G^T) \Leftrightarrow \{a_i, w\} \in E(G^T) \Leftrightarrow \{a_i, w\} \in E(G) \Leftrightarrow \{c_i, w\} \in E(G).$$

Finally, suppose $w \in V(T_v)$. In this case either the path between w and u_i contains v and therefore the distance between w and u_i is the same as the distance between w and a_i, or the path between w and u_i does not contain v and the distance between w and u_i is the same as the distance between w and b_i. As the adjacency between w, a_i, w, b_i and w, c_i is the same in G, this implies that $\{w, d_i\} \in E(G)$ if, and only if, $\{w, c_i\} \in E(G)$. ◀

The next result follows easily by induction on $|M \setminus \{\bar{a}, \bar{b}\}|$ using Lemma 3.6.

▶ **Corollary 3.9.** *Let $d \geq 0$ and $k, m \geq 1$. Let G be a graph and let M be a k-module in G. Let $\bar{a}, \bar{b} \in M$ and let $G' = G - \bigcup\{\bar{c} : \bar{c} \in M \setminus \{\bar{a}, \bar{b}\}\}$. If there is a (d, m)-tree-model (T', S', λ') of G' such that $lca(\{a_1, \dots, a_k\})$ and $lca(\{b_1, \dots, b_k\})$ are incomparable in T' and $\lambda'(a_i) = \lambda'(b_i)$ for all $i \in [k]$, then G has a (d, m)-tree-model (T, S, λ) which extends (T', S', λ').*

4 Strict modules in graphs of low shrub-depth

In this section we prove several results about strict k-modules and their relation to tree-models. We start with Lemma 4.1 which states that if G contains a sufficiently large strict k-module M, then in every tree-model T of G there are two tuples of M in a mutual position which allows us to apply Lemma 3.6 and Corollary 3.9. We then establish Lemma 4.4, which is an analogue of Lemma 3.5 and which establishes the connection between tuples in strict k-modules and groups of isomorphic subtrees of *unsplittable* tree-models. Finally, we prove the technical Lemma 4.5 and Corollary 4.6 which establish that for each d and m there exist bounded values of k such that sufficiently large strict modules exist in large enough graphs which have (d, m)-tree-models.

▶ **Lemma 4.1.** *There exists a function $L : \mathbb{N}^3 \to \mathbb{N}$ such that for all $d, m, k > 0$ the following holds: if G is a graph and M is a strict k-module in G of size $|M| \geq L(m, d, k)$, then in any (d, m)-tree-model (T, S, λ) of G there are at least two tuples $(a_1, \dots, a_k)$ and $(b_1, \dots, b_k)$ in M such that*

1. *$lca_T(\{a_1, \dots, a_k\})$ and $lca_T(\{b_1, \dots, b_k\})$ are incomparable and*
2. *$\lambda(a_i) = \lambda(b_i)$ for all $i \in [k]$.*

Proof. Let T be a (d,m)-tree-model of G. Fix a linear order $<_\rho$ of the leaves of T such that for any three leaves u,v,w of T the following holds: if $u <_\rho v <_\rho w$ then v is a descendant of $lca_T(u,w)$ or $v = lca_T(u,v)$. It is easy to see that such order exists – for example if we run the DFS algorithm from the root T, then the order in which the leaves of T are visited by the algorithm has this property. For any tuple $\bar{a} = (a_1,\dots,a_k)$ let $T^{\bar{a}}$ denote the smallest subtree of T which contains $a_1,\dots,a_k$ and $lca_T(a_1,\dots,a_k)$. We say that two tuples $\bar{a},\bar{b}$ are (T,ρ)-similar if $T^{\bar{a}}$ and $T^{\bar{b}}$ are isomorphic, where we require that the isomorphism maps a_i to b_i and respects the colors of leaves and also the order $<_\rho$. It is easy to see that the (T,ρ)-similarity relation is an equivalence with finitely many classes; let $\gamma(m,d,k)$ denote the number of equivalence classes. We set $L(m,d,k) := (d+1)\gamma(m,d,k)$. Assume now that a strict k-module M of G has size at least $L(m,d,k)$. Then there are at least $d+1$ tuples in M which are (T,ρ)-similar; let us denote this set of tuples by S. We claim that there exists a pair of tuples $\bar{a} = (a_1,\dots,a_k)$ and $\bar{b} = (b_1,\dots,b_k)$ in S such that $lca_T(\{a_1,\dots,a_k\})$ and $lca_T(\{b_1,\dots,b_k\})$ are incomparable, in which case we are done. Assume for contradiction that there is no such pair. In this case there have to be two tuples $\bar{a} = (a_1,\dots,a_k)$ and $\bar{b} = (b_1,\dots,b_k)$ in S such that $lca_T(a_1,\dots,a_k) = lca_T(b_1,\dots,b_k)$, because the least common ancestors of all tuples in S are comparable and $|S| \geq d+1$. In the remainder of the proof we show that there exist i,j with $1 \leq i,j \leq k$ such that i,j is a conflict pair in M but the adjacency between a_i,a_j,b_i,b_j in G does not lead to a conflict, which is a contradiction. Set $v := lca_T(a_1,\dots,a_k) = lca_T(b_1,\dots,b_k)$. We take as the pair i,j any conflicting pair of M such that $lca_T(a_i,a_k) = v$. To see that such pair exists, partition $\{a_1,\dots,a_k\}$ into sets $A^1,A^2,\dots$ according to the following rule: two vertices of $\{a_1,\dots,a_k\}$ are in the same set if their least common ancestor is not v (note that in this case the least common ancestor is a descendant of v as $v = lca_T(a_1,\dots,a_k)$). It is easily seen that this is an equivalence. Since the conflict graph of M is connected, there have to be vertices $a_i \in A^1$ and $a_j \in A^2$ such that i,j is a conflict pair. Since they are in different sets, it has to hold that $lca_T(a_i,a_k) = v$. Without loss of generality we may assume that $i = 1$ and $j = 2$.

We now examine the adjacency of a_1,a_2,b_1,b_2 in G. To disprove that $(1,2)$ is a conflict pair in M, it is enough to show that (i) the adjacency between a_1 and a_2 is the same as the adjacency between a_1 and b_2 or (ii) the adjacency between b_1 and b_2 is the same as the adjacency between b_1 and a_2 Without loss of generality assume that $a_1 <_\rho a_2$ (which also means that $b_1 <_\rho b_2$ because $\bar{a}$ and $\bar{b}$ are isomorphic) and that $a_1 < b_1$ (otherwise we just swap $\bar{a}$ and $\bar{b}$). There are two cases to consider. First, if $a_2 <_\rho b_1$, then we have $a_1 <_\rho a_2 <_\rho b_1 <_\rho b_2$. In this case, since $lca_T(a_1,a_2) = v$ we also have to have $lca_T(a_1,b_2) = v$ (this follows from the definition of $<_\rho$), which means that the distance between a_1 and a_2 is the same as the distance between a_1 and b_2. Since $\lambda(a_2) = \lambda(b_2)$ (again, because $\bar{a}$ and $\bar{b}$ are isomorphic), we have $\{a_1,a_2\} \in E(G) \Leftrightarrow \{a_1,b_2\} \in E(G)$ and we have shown (i) above. The second case to consider is when $b_1 <_\rho a_2$. Then we have either $a_1 <_\rho b_1 <_\rho a_2 <_\rho b_2$ or $a_1 <_\rho b_1 <_\rho b_2 <_\rho a_2$. In the first situation we argue as in the previous case, and in the second situation we know that since $lca_T(b_1,b_2) = v$ it also has to hold $lca_T(b_1,a_2) = v$, and we get that $\{b_1,b_2\} \in E(G) \Leftrightarrow \{b_1,a_2\} \in E(G)$, which is the situation (ii) above. ◀

Let T be a tree and A a subset of leaves of T. We define T^A to be the smallest subtree of T which contains the root of T and all vertices from A.

▶ **Definition 4.2.** *A tree-model T is* splittable *if there exists a node u such that the leaves of the tree T_u can be partitioned into two sets A and B such that if we remove T_u from T and replace it by attaching T_u^A and T_u^B to the parent of u, then the resulting tree-model defines the same graph as T. If T is not splittable, it is* unsplittable.

▶ **Lemma 4.3.** *Every graph which has a (d, m)-tree-model has an unsplittable (d, m)-tree-model.*

Proof. Let T be a (d, m)-tree-model of G. If T is splittable, we keep splitting it (as in the Definition 4.2) as long as possible. Each splitting increases the number of internal nodes in T, and since there are at most $|V(G)| \cdot (d-1) + 1$ internal nodes in any tree-model of depth d of G, the process has to stop. ◀

The next lemma captures the connection between unsplittable tree-models and strict modules.

▶ **Lemma 4.4.** *Let (T, S, λ) be an unsplittable tree-model of a graph G and let u be a node of T with $L \geq 2$ children $\{1, \ldots, L\}$ such that for any two of its children i and j there exists a label preserving isomorphism ι_{ij} between T_i and T_j. Then G contains a strict k-module with L tuples, where k is the number of leaves in T_1.*

Proof. Let W_i denote the set of leaves of T_i. Fix any ordering on W_1 and for every $j > 1$ use ι_{ij} to define the corresponding ordering on W_j, i.e. define $\leq_j$ on W_j by setting $u \leq_j v$ if, and only if, $\iota_{j1}(u) \leq_1 \iota_{j1}(v)$. Each pair $(W_i, \leq_i)$ can be thought of as an ordered k-tuple, and we claim that $M := \{(W_1, \leq_1), \ldots, (W_L, \leq_L)\}$ is a strict k-module with L tuples claimed in the statement of the lemma.

The fact that M is a k-module in G with L tuples is clear from its definition, and so it remains to argue that M is in fact strict. For the sake of contradiction assume that M is not strict, which means that its conflict graph $\mathcal{C}(M)$ on the vertex set $[k]$ is not connected. Let C be a connected component of $\mathcal{C}(M)$ on $p < k$ vertices, and without loss of generality assume that $V(C) = [p]$. We will prove that T is splittable, which is the contradiction with our assumption on T. To simplify the notation in the rest of the proof, we will from now on denote the tuples of M by the usual $\bar{a}, \bar{b}, \ldots$ instead of $(W_1, \leq_1), (W_2, \leq_2), \ldots$. Let $\bar{a}$ be a tuple of M. Let $A := \{a_1, \ldots, a_p\}$ be the set of the first p vertices in $\bar{a}$ and let $A' = \{a_1, \ldots, a_k\} \setminus A$. We remove T_1 from T and replace it by attaching to u the trees T_u^A and $T_u^{A'}$ to obtain a new tree-model (T', S, λ) (the relation S and labeling function λ are the same as in T). We claim that (T', S, λ) is a tree-model of G. Since the transformation of T into T' does not change any labels, the only change in the adjacency defined by T' compared to T can come from a change of distance between two leaves. The only situation when $dist_T(v, w) \neq dist_{T'}(v, w)$ is when $v \in A$ and $w \in A'$ or vice versa – in this case it holds that $dist_{T'}(v, w) = 2l$, where l is the height of u in T (and also in T'). We now argue that in this case the adjacency between u and u remains unchanged, i.e. $\{v, w\} \in E(G^T) \Leftrightarrow \{v, w\} \in E(G^{T'})$. Since $v \in A$ and $w \in A'$ we have that $v = a_i$ for some $i \leq p$ and $w = a_j$ for some $j > p$. Assume that $\{a_i, a_j\} \in E(G^T)$. We need to show that $\{a_i, a_j\} \in E(G^{T'})$, which is the case exactly when $(\lambda(a_i), \lambda(a_j), 2l) \in S$. Let α and β be the relations of M. Since a_i and a_j are in the same tuple of M, it holds that $\{i, j\} \in \alpha$. Since i and j are in different connected components of the conflict graph $\mathcal{C}(M)$, it also holds that $\{i, j\} \in \beta$. Let $\bar{b} := (b_1, \ldots, b_k)$ be a tuple of M different from $\bar{a}$. Since $\{i, j\} \in \beta$, there is an edge between a_i and b_j in G. Because the distance between a_i and b_j in T is $2l$, this means that $(\lambda(a_i), \lambda(b_j), 2l) \in S$. Because M is a module, $\lambda(a_j) = \lambda(b_j)$, and so $(\lambda(a_i), \lambda(a_j), 2l) \in S$, which means that $\{a_i, a_j\} \in E(G^{T'})$ as desired. The other direction is proved analogously. ◀

▶ **Lemma 4.5.** *For every $d, m \geq 1$ and every sequence $0 < L_1 \leq L_2 \leq \ldots$ there exist K and N such that every graph which has a (d, m)-tree-model and has more than N vertices contains, for some $k \leq K$, a strict k-module with more than L_k tuples.*

Proof. For a tree-model T we say that two nodes u, v of T are T-isomorphic if they have the same parent and there is a label preserving isomorphism between T_u and T_v.

We will prove by induction on d the following statement, which implies the lemma by means of Lemma 4.4. For every d and m there exist numbers $K(d,m)$ and $N(d,m)$ such that the following holds: In every (d,m)-tree-model with at least $N(d,m)$ leaves there is a node which, for some $k \leq K(d,m)$, has more than L_k pairwise T-isomorphic children each of which has k leaves.

For $d = 1$ we set $k(1,m) := 1$ and $N(1,m) := mL_1$. Let G be a graph on more than $N(1,m)$ vertices and let T be its tree-model of height 1. Then T has more than $N(1,m)$ leaves and for at least one label there are more than L_1 leaves having this label, which means that they form a set of more than L_1 pairwise T-isomorphic children of the root.

Assume now that $d > 1$ and the statement holds for $d-1$. Set $K(d,m) := N(d-1,m)$ and $N(d,m) := N(d-1,m) \cdot L_{K(d,m)} \cdot \gamma(d-1,m)$, where $\gamma(d-1,m)$ is the number of non-isomorphic $(d-1,m)$-tree-models with at most $N(d-1,m)$ leaves, and where it is understood that the isomorphisms are label preserving. Let T be a (d,m)-tree-model with more than $N(d,m)$ leaves. We distinguish two cases:

1. There is a child u of the root of T such that T_u has more than $N(d-1,m)$ leaves. Then by the inductive assumption there is a node in T_u which, for some $k \leq K(d-1,m)$, has more than L_k pairwise T-isomorphic children each of which has k leaves. Since $K(d-1,m) < K(d,m)$ we are done.
2. For every child u of the root r of T it holds that T_u has at most $N(d-1,m)$ leaves. In this case r has more than $\frac{N(d,m)}{N(d-1,m)} = L_{K(d,m)} \cdot \gamma(d-1,m)$ children, each of which corresponds to a subtree T_u of T with at most $N(d-1,m)$ leaves. We group the subtrees of T determined by the children of r into groups $C_1, \ldots, C_{\gamma(d-1,m)}$ according to their labeled isomorphism type. Because there are more than $L_{K(d,m)} \cdot \gamma(d-1,m)$ of these trees, at least one group C_i has more than $L_{K(d,m)}$ trees in it. All these trees are pairwise isomorphic and have at most $N(d-1,m) = K(d,m)$ leaves; let us denote this number of leaves by k. Since $k \leq K(d,m)$, we have $L_k \leq L_{K(d,m)}$ and therefore C_i has more than L_k trees, as desired. ◀

▶ **Corollary 4.6.** *For every $d, m \geq 1$ there exist K and N such that every graph which has a (d,m)-tree-model and has more than N vertices contains, for some $k \leq K$, a strict k-module with more than $L(m,d,k)$ tuples, where $L(m,d,k)$ is the function from Lemma 4.1.*

Proof. For every k set L_k to be the $L(m,d,k)$ from Lemma 4.1 (where it is easily seen that $L(m,d,k-1) \leq L(m,d,k)$ for each k) and apply Lemma 4.5. ◀

5 Algorithms

In this section we use the results from the previous section to obtain two algorithms for computing tree-models of graphs.

The results obtained in the previous section suggest the following strategy to compute, given a graph G and $d, m > 0$ as input, a (d,m)-tree-model of G, provided such a tree-model of G exists.

The main algorithmic strategy. Let G be a graph and $d, m > 0$ be integers.

Step 1. Given d, m, let K and N be the numbers stated in Corollary 4.6.

Step 2. As long as $|G| > N$, repeat the following steps.

- **a.** find, for some $k < K$, a strict k-module M in G of size $|M| > L(m,d,k)$
- **b.** choose a set $M^* \subseteq M$ of order exactly $L(m,d,k)$ and delete all elements of all tuples in $M \setminus M^*$. Remember the sets M and M^* for each such step.

Step 3. Let G' be the remaining graph of order $|G'| \leq N$. Compute a (d, m)-tree-model (T', S', λ') of G' by brute force.

Step 4. In reverse order of their creation, for each module M and set $M^* \subseteq M$ constructed in the iterations of Step 2

a. find tuples $\bar{a}, \bar{b}$ in T' satisfying the requirements of Corollary 3.9.

b. Extend (T', S', λ') by adding the vertices in $M \setminus M^*$ as described in Corollary 3.9.

The correctness of this approach follows from the results in Section 4. By Corollary 4.6, given d and m, the numbers K and N used in Step 1 depending only on d and m but not on G exist such that if $|G| > N$, then G contains a strict k-module M of size $> L = L(m, d, k)$, for some $k < K$. Here and below $L(m, d, k)$ is the function defined in Lemma 4.1.

In Step 2 we iteratively reduce the size of G until its size is bounded by a function of the parameters d and m. This creates a sequence $G = G_0 \supset_i G_1 \supset_i \ldots \supset_r = G'$ of graphs, where r is the number of iterations in Step 2. Notice that in each iteration, when we remove $M \setminus M^*$ from G_i to obtain G_{i+1}, we keep in G_{i+1} enough tuples of M to be able to construct a (d, m)-tree-model T^i of G_i from a (d, m)-tree-model T^{i+1} of G_{i+1}. To see this, notice that M^* (which was not removed and is a strict k-module of G_{i+1}) has $L(m, d, k)$ tuples, and so by of Lemma 4.1 there are tuples $(a_1, \ldots, a_k)$ and $(b_1, \ldots, b_k)$ in M^* such that the vertices $\{a_1, \ldots, a_k, b_1, \ldots, b_k\}$ are placed in T^{i+1} in accordance with the assumptions of Corollary 3.9, the application of which allows us to put all tuples from $M \setminus M^*$ into T^{i+1} to obtain T^i.

After completing Step 2 we are left with an induced subgraph G' of G of size $|G'| \leq N$. In Step 3 we compute a (d, m)-tree-model (T', S', λ') of G'. As N only depends on the parameters d and m we can compute T' by brute-force. If no such (d, m)-tree-model of G' exists, then G does not have a (d, m)-tree-model as G' is an induced subgraph of G and the existance of tree-models is preserved by taking induced subgraphs.

Otherwise, if we find a (d, m)-tree-model (T', S', λ') of G' we extend it to a (d, m)-tree-model of the input graph G in Step 4. For this, we iterate again over all modules M and subsets M^* constructed in the iterations of Step 2 and apply Corollary 3.9 to extend T' so that it contains the vertices in $M \setminus M^*$.

This proves the general correctness of the algorithmic approach described above. In the remainder of this section we show how the various steps in the algorithm above can be implemented to eventually yield an fpt-algorithm for comptuting tree-models.

As a first step towards this goal we present a simple XP-algorithm implementing the approach described above. The methods we use for this algorithm for the Steps 3 and 4 but not for Step 2 are already good enough for an fpt-algorithm. What remains to be done is to improve Step 2.

As a second step, we present an improved XP-algorithm with a better algorithm for Step 2. Finally we show how this new strategy for Step 2 can be implemented in a way to yield an fpt-algorithm as required.

The XP algorithms. The first algorithmic step we prove is the following lemma, which is an easy consequence of Lemma 3.6 and 4.1. The lemma essentially states that once we have found a tree-model T' for the reduced graph G' obtained from G by removing some tuples of a k-module, the tree-model of G can be computed efficiently from T.

▶ **Lemma 5.1.** *Let G be a graph which has a (d, m)-tree-model and let M be a strict k-module containing more than $L = L(m, d, k)$ tuples. Let $Q \subseteq M$ be such that $|M \setminus Q| = L$ and let G' be the graph obtained from G by deleting all vertices contained in tuples in Q. Then any (d, m)-tree-model T' of G' can be extended in linear time to a (d, m)-tree-model T of G.*

Proof. Let (T', S', λ') be a (d, m)-tree-model of G'. Since $M \setminus Q$ is a strict k-module in G' containing $L = L(m, d, k)$ tuples, Lemma 4.1 guarantees that there are tuples $\bar{a} := (a_1, \ldots, a_k)$, $\bar{b} := (b_1, \ldots, b_k) \in M \setminus Q$ in G such that $lca_{T'}(\{a_1, \ldots, a_k\})$ and $lca_{T'}(\{b_1, \ldots, b_k\})$ are incomparable in T' and $\lambda'(a_i) = \lambda'(b_i)$ for all $i \in [k]$. By Corollary 3.9, (T', S', λ') can be extended to a (d, m)-tree-model (T, S, λ) of G. It is straight forward to verify that the proof of 3.9 can be made algorithmic and can be implemented in linear time. Note that when we delete the tuples in Q, we will store with Q also the tuples $\bar{a}, \bar{b}$, as we need them to add the elements of Q to the tree-model T'. ◀

The previous lemma shows how Step 4 above can be implemented. Step 3 can be done by brute-force, so all that remains is to provide an algorithm for Step 2.

Towards this aim, note that since k and $L_k = L(m, d, k)$ depend only on d and m and not on $|V(G)|$, we can simply go over all subsets $X \subseteq V(G)$ of size $|X| = k(L_k + 1)$ in time $|V(G)|^{k(L_k+1)}$ and for each such X check whether it can be partitioned into a strict k-module. Using this as a sub-routine for Step 2 above to obtain our first XP-algorithm for computing tree-models.

However, this way of finding strict k-modules is highly inefficient, and in the remainder of this section we argue that the runtime can be improved to $|V(G)|^{3k+1}$ by using the following simple greedy procedure. For every set $X \subseteq V(G)$ of $2k$ vertices of G we 1) generate all partitions of X into two disjoint sets X_a, X_b of k vertices each and 2) for each of these we consider all possible ways to order the vertices in X_a and X_b so that we obtain ordered k-tuples $\bar{a}$ and $\bar{b}$ and then we 3) check whether $\bar{a}$ and $\bar{b}$ are k-twin tuples. For any pair $\bar{a}, \bar{b}$ of twin tuples obtained in this way we let $M := \{\bar{a}, \bar{b}\}$ and iterate over all k-tuples $\bar{c}$ of vertices from $V(G) \setminus V(M)$. For any such $\bar{c}$ we check whether $\bar{c}$ is a twin tuple of every tuple already contained in M. If so, we add $\bar{c}$ to M and repeat, extending M as long as possible.

Observe that this procedure is approximate in the following sense: it finds a strict k-module with more than L_k tuples provided that a strict k-module with more than kL_k tuples exists in G. As this procedure requires G to contain a strict k-module of size kL_k instead of L_k, whenever we apply Lemma 4.5 in the general algorithm above, we use m, d but with a different sequence $L_1 \leq L_2 \leq \ldots$ defined as $L_k = kL(m, d, k)$. This guarantees that Lemma 4.5 applied to m, d and this new sequence $L_1 \leq L_2 \leq \ldots$ yields suitable K and N that make the algorithm above work.

We show next that this revised procedure for Step 2a is correct in the sense that it indeed produces a strict k-module of size $> L(m, d, k)$ as required. Towards this aim, let Z be a strict k-module of G with the maximum number of tuples (in particular note that Z has more than $kL(m, d, k)$ tuples).

1. At least one initial guess of $\bar{a}$ and $\bar{b}$ yields a pair of twin tuples from Z, because we go over all sets of size $2k$, all possible ways to split them into k-tuples $\bar{a}$ and $\bar{b}$. The pair $\bar{a}, \bar{b}$ also determines relations α and β, which guarantees that these are the same for M and Z.
2. Even if every tuple $\bar{c}$ we find is suboptimal (i.e. $\bar{c}$ is not one of the tuples in Z but intersects several of these), it can intersect at most k tuples in Z. The remaining tuples in Z which do not contain any vertex contained in a tuple in M are twin tuples of every tuple in M and therefore also of $\bar{c}$, as the relation of being an (α, β)-twin tuple with respect to fixed α and β is transitive.

Item 2 implies that after the i-th iteration of adding a tuple $\bar{c}$ to M there are more than $kL_k - 2 - ki$ tuples in Z left which can still be added to M. Thus, at least $L_k - 1$ iterations will be performed and therefore M will have at least $L_k + 1$ tuples.

The FPT-algorithm. We are now ready to present the fpt-algorithm for computing (d, m)-tree-models. The reason why the previous algorithm is only an XP-algorithm is that it iterates over all possible sets of vertices of size $2k$ to find a pair of twin tuples $\bar{a}, \bar{b}$ and then later on again iterates over all sets of size k to find a suitable tuple $\bar{c}$.

In this section we prove that in order to find a pair of twin tuples $\bar{a}, \bar{b}$ in G it is enough to guess a pair u, v of vertices and then check in linear time whether they can be extened to an (α, β)-twin tuple. Similarly, given a strict k-module M of G, to find a twin k-tuple $\bar{c}$ of all tuples in M it is enough to guess one vertex u of $\bar{c}$ and the check in linear time whether it can be extended appropriately.

▶ **Lemma 5.2.** *Let $k \in \mathbb{N}$ and let α and β be relations on $[k]$ such that they determine a connected conflict graph. Let G be a graph and u, v be vertices of G. Then the number of twin tuples $(a_1, \ldots, a_k)$, $(b_1, \ldots, b_k)$ such that $u = a_1$ and $v = b_1$ is bounded by a function in k and there is an algorithm running in time $f(k) \cdot |V(G)|$ which, given G, u and v as input, computes all such pairs of twin tuples.*

Proof. We will prove that for every pair $\big((A, f_A), (B, f_B)\big)$ where A and B are disjoint subsets of $V(G)$ with $|A| = |B| = p \leq k$ and $f_A : A \to [k]$ and $f_B : B \to [k]$ are injective functions with the same image in $[k]$ we can generate in time $f(k) \cdot |V(G)|$ all pairs $\big((\bar{A}, f_{\bar{A}}), (\bar{B}, f_{\bar{B}})\big)$ such that:

- $\bar{A}$ and $\bar{B}$ are disjoint, $|\bar{A}| = |\bar{B}| = k$ and $A \subseteq \bar{A}$, $B \subseteq \bar{B}$
- $f_{\bar{A}} : \bar{A} \to [k]$ and $f_{\bar{B}} : \bar{B} \to [k]$ are injective
- $\bar{A}$ and $\bar{B}$ with the orderings induced by $f_{\bar{A}}$ and $f_{\bar{B}}$ in the obvious way are (α, β)-twin-tuples.

Moreover, the number of such pairs $(\bar{A}, f_{\bar{A}})$, $(\bar{B}, f_{\bar{B}})$ will be bounded in terms of k.

We prove this by induction on $j = k - p$. If $j = 0$, then there is nothing to generate and we only need to check whether (A, f_A) and (B, f_B) have the adjacency prescribed by α and β. Now assume that the statement holds for all integers less than j and let (A, f_A), (B, f_B) be an instance with $|A| = |B| = p$ where $j = k - p$. First, we check whether $G[A]$ and $G[B]$ and all edges between A and B in $G[A \cup B]$ are consistent with α and β (with respect to the ordering induced by f_A and f_B). If this is not the case, we can immediately say that (A, f_A), (B, f_B) cannot be extended.

So we may assume that $G[A], G[B]$, and $G[A \cup B]$ are consistent with α and β. To simplify the notation, in the rest of this proof we will denote by a_i the element of A such that $f_A(a_i) = i$ and by b_i the element of B with $f_B(b_i) = i$. For all i in the image $im(f_A)$ of f_A (and thus also in the image $im(f_B)$ of f_B), let $S_i := (N(a_i) \Delta N(b_i)) \setminus (A \cup B)$. That is, S_i is the set of all vertices outside of $A \cup B$ on which a_i and b_i differ. Let $S := \bigcup_{i \in im(f_A)} S_i$. Since the conflict graph determined by α and β is connected, for any $j > 0$ and any pair (A, f_A), (B, f_B) which can be extended to an (α, β)-twin pair $(\bar{A}, f_{\bar{A}})$, $(\bar{B}, f_{\bar{B}})$ the set S will be non-empty. Moreover, all vertices in S have to be included in all extensions $(\bar{A}, f_{\bar{A}})$, $(\bar{B}, f_{\bar{B}})$ satisfying the properties above. For, otherwise there would be an i such that a_i and b_i are not twins in $V(G) \setminus \bar{A} \cup \bar{B}$, which contradicts the definition of twin-tuples. If S is larger than $2k - 2j$ we know that A and B cannot be extended as desired, because then we would have $|\bar{A}| + |\bar{B}| > 2k$, again a contradiction. If S has size at most $2k - 2j$, then we consider all partitions of S into sets S_A and S_B of equal size, set $A' := A \cup S_A$ and $B' := B \cup S_B$ and consider all injective functions $f_{A'} : A' \to [k]$, $f_{B'} : B' \to [k]$ which have the same image and which agree with f_A and f_B on A and B, respectively. Since $|A'| > |A|$ and $|B'| > |B|$, we have that $k - |A| < j$ and we can apply the induction hypothesis to $(A', f_{A'})$, $(B', f_{B'})$.

Clearly in the case when we use the induction hypothesis the size of the set S is bounded by k, and so is the number of its bipartitions into S_A and S_B and also the number of different functions $f_{A'}$ and $f_{B'}$. This completes the proof. ◀

▶ **Lemma 5.3.** *Let $k \in \mathbb{N}$ and let α and β be relations on $[k]$ such that they determine a connected conflict graph. Let G be a graph, M be a strict (α, β)-module in G and let $v \in V(G) \setminus V(M)$. Then in time $g(k) \cdot |V(G)|$ one can find a tuple $\bar{c} := (c_1, \ldots, c_k)$ in $V(G) \setminus V(M)$ with $c_1 = v$ such that $\bar{c}$ is an (α, β)-twin of all tuples in M, or determine that no such tuple exists.*

Proof. Let $\bar{a}$ be a tuple of M and let G' be obtained from G by deleting all tuples of M with the exception of $\bar{a}$. Note that if $\bar{c}$ exists in G, then it is an (α, β)-twin tuple of $\bar{a}$ in G'. We apply Lemma 5.2 to a_1 and v to obtain the set of all (α, β)-twin tuples which have a_1 and v on their first positions, resp. If any of these pairs of tuples contains $\bar{a}$, then the the second tuple of this pair can be taken as $\bar{c}$. The fact that $\bar{c}$ is also an (α, β)-twin tuple of all tuples in M in G follows from trasitivity of being (α, β)-twin tuple. ◀

We are now ready to prove the main algorithmic result of this section.

▶ **Theorem 5.4.** *Let G be a graph which contains a strict k-module Z with more than kL tuples. Then it is possible to find a strict k-module M with more than L tuples in time $h(k) \cdot |V(G)|^4$.*

Proof. The algorithm iterates over all pairs of symmetric relations α and β on $[k]^2$ which determine a connected conflict graph and for each such pair proceeds as follows. For every pair u, v of vertices in G it uses the algorithm from Lemma 5.2 to generate the set S of all pairs of (α, β)-twin tuples which have u and v on their first positions. Then, for each pair $\bar{a}, \bar{b} \in S$ we set $M := \{\bar{a}, \bar{b}\}$ and by repeated application of Lemma 5.3 we extend M by finding a tuple $\bar{c}$ which is an (α, β)-twin of every tuple in M and adding $\bar{c}$ to M for as long as possible.

We now argue that at least for one choice of α, β, u and v the algorithm produces a strict k-module with more than L elements. Set α and β to be the relations of Z and let u, v be vertices which are on the first position of tuples $\bar{a}, \bar{b}$ of Z. By applying the algorithm from Lemma 5.2 to α, β, u, v we find all pairs of (α, β)-twin tuples which have u and v on their first position. In particular we will find $\bar{a}$ and $\bar{b}$. We then set $M := \{\bar{a}, \bar{b}\}$ and try to extend M as much as possible using Lemma 5.3. We can argue in exactly the same way as in the previous section that the number of successful iterations of extending M by a tuple $\bar{c}$ will be at least $L - 1$. Thus, together with $\bar{a}$ and $\bar{b}$, the k-module M we find will have at least $L + 1$ tuples. ◀

Using the algorithm of Theorem 5.4 in Step 2 of the general algorithmic strategy outlined at the beginning of this section, we obtain our main algorithmic result.

▶ **Theorem 5.5.** *There is an algorithm which, given a graph G and numbers $m, d > 0$ as input, in time $f(d, m) \cdot |G|^c$, for a computable function f and a constant c both independent of G, d, and m, either computes a (d, m)-tree-model of G or correctly determines that no such module exists.*

6 Application for forbidden induced subgraphs

As an easy consequence of our results from Section 4 we obtain a simple proof of the following theorem, which was originally proven in [8].

▶ **Theorem 6.1.** *For every d, m there exists a finite set of graphs $\mathcal{F}_{d,m}$ such that a graph G has a (d, m)-tree-model if, and only if, G does not have an induced subgraph isomorphic to a member of $\mathcal{F}_{d,m}$.*

Compared to the proof of Theorem 6.1 given in [8], our proof given below has the advantage of providing explicit bounds on the size of graphs in $\mathcal{F}_{d,m}$ and therefore being constructive.

Proof. Fix d and m. Let $\mathcal{F}_{d,m}$ be the set of all graphs H such that H does not have a (d, m)-tree-model and every proper induced subgraph of H has a (d, m)-tree-model. It is easy to see that for every graph G it holds that G has a (d, m)-tree-model if, and only if, G does not have an induced subgraph isomorphic to a member of $\mathcal{F}_{d,m}$. We will show that there is a bound on the size of graphs in $\mathcal{F}_{d,m}$.

Let K and N be the numbers obtained from Corollary 4.6 applied to d and m. We claim that no graph in $\mathcal{F}_{d,m}$ has more than N vertices. Assume towards a contradiction that there is a graph H from $\mathcal{F}_{d,m}$ which has more than N vertices. By Corollary 4.6 there is a strict k-module M in H with more than $L_(d, m, k)$ tuples for some $k \leq K$. Let H' be the graph obtained from H by removing one tuple $\bar{c}$ from M. Then H' is a proper induced subgraph of H and has strict k-module M' with at least $L_(d, m, k)$ tuples. Since H' is a proper induced subgraph of H, it has (d, m)-tree-model T'. By Lemma 4.1 there are two tuples $\bar{a}$ and $\bar{b}$ in M' such that they are in different subtrees of T'. By Lemma 3.6 we can extend T' (by adding $\bar{c}$ into it) to a (d, m)-tree-model T of H, which is a contradiction with the assumption that H has no (d, m)-tree-model. Thus, no member of $\mathcal{F}_{d,m}$ has more than N vertices. ◀

References

1 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

2 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

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

4 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

5 Jakub Gajarský and Petr Hlinený. Kernelizing MSO properties of trees of fixed height, and some consequences. *Logical Methods in Computer Science*, 11(1), 2015. doi:10.2168/LMCS-11(1:19)2015.

6 Jakub Gajarský, Stephan Kreutzer, Jaroslav Nesetril, Patrice Ossona de Mendez, Michal Pilipczuk, Sebastian Siebertz, and Szymon Toru'nczyk. First-order interpretations of bounded expansion classes. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 126:1–126:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.126.

7 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In Gregory Z. Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation - 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*, volume 8246 of *Lecture Notes in Computer Science*, pages 163–176. Springer, 2013. doi:10.1007/978-3-319-03898-8_15.

8 Robert Ganian, Petr Hlinený, Jaroslav Nešetřil, Jan Obdržálek, and Patrice Ossona de Mendez. Shrub-depth: Capturing height of dense graphs. *Logical Methods in Computer Science*, 15(1), 2019. URL: https://lmcs.episciences.org/5149, doi:10.23638/LMCS-15(1:7)2019.

9 Petr Hlinený and Sang-il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008. doi:10.1137/070685920.

10 Jaroslav Nesetril and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006. doi:10.1016/j.ejc.2005.01.010.

11 Jaroslav Nesetril 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.

12 Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A faster parameterized algorithm for treedepth. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 931–942. Springer, 2014. doi:10.1007/978-3-662-43948-7_77.

Typical Sequences Revisited – Computing Width Parameters of Graphs

Hans L. Bodlaender
Department of Computer Science, Utrecht University, The Netherlands
h.l.bodlaender@uu.nl

Lars Jaffke
Department of Informatics, University of Bergen, Norway
lars.jaffke@uib.no

Jan Arne Telle
Department of Informatics, University of Bergen, Norway
jan.arne.telle@uib.no

Abstract

In this work, we give a structural lemma on merges of typical sequences, a notion that was introduced in 1991 [Lagergren and Arnborg, Bodlaender and Kloks, both ICALP 1991] to obtain constructive linear time parameterized algorithms for treewidth and pathwidth. The lemma addresses a runtime bottleneck in those algorithms but so far it does not lead to asymptotically faster algorithms. However, we apply the lemma to show that the cutwidth and the modified cutwidth of series parallel digraphs can be computed in $\mathcal{O}(n^2)$ time.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Graph algorithms

Keywords and phrases typical sequences, treewidth, series parallel digraphs, cutwidth, modified cutwidth

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.57

Related Version A full version of the paper is available at `https://arxiv.org/abs/1905.03643`.

Funding *Hans L. Bodlaender*: Partially supported by the Networks project, funded by the Netherlands Organization for Scientific Research (NWO).
Lars Jaffke: Supported by the Bergen Research Foundation (BFS).

Acknowledgements This work was started when the third author was visiting Universitat Politecnica de Valencia, and part of it was done while the second author was visiting Utrecht University.

1 Introduction

In this paper we revisit an old key technique from what currently are the theoretically fastest parameterized algorithms for treewidth and pathwidth, namely the use of *typical sequences*, and give additional structural insights for this technique. In particular, we show a structural lemma, which we call the *Merge Dominator Lemma*. The technique of typical sequences brings with it a partial ordering on sequences of integers, and a notion of possible merges of two integer sequences; surprisingly, the Merge Dominator Lemma states that for any pair of integer sequences there exists a *single* merge that dominates all merges of these integer sequences, and this dominating merge can be found in linear time. On its own, this lemma does not lead to asymptotically faster parameterized algorithms for treewidth and pathwidth, but, as we discuss below, it is a concrete step towards such algorithms.

The notion of typical sequences was introduced independently in 1991 by Lagergren and Arnborg [15] and Bodlaender and Kloks [8]. In both papers, it is a key element in an explicit dynamic programming algorithm that given a tree decomposition of bounded width ℓ, decides

if the pathwidth or treewidth of the input graph G is at most a constant k. Lagergren and Arnborg build upon this result and show that the set of forbidden minors of graphs of treewidth (or pathwidth) at most k is computable; Bodlaender and Kloks show that the algorithm can also construct a tree or path decomposition of width at most k, if existing, in the same asymptotic time bounds. The latter result is a main subroutine in Bodlaender's linear time algorithm [3] for treewidth-k. If one analyses the running time of Bodlaender's algorithm for treewidth or pathwidth $\leq k$, then one can observe that the bottleneck is in the subroutine that calls the Bodlaender-Kloks dynamic programming subroutine, with both the subroutine and the main algorithm having time $\mathcal{O}(2^{\mathcal{O}(k^3)}n)$ for treewidth, and $\mathcal{O}(2^{\mathcal{O}(k^2)}n)$ for pathwidth. See also the recent work by Fürer for pathwidth [13], and the simplified versions of the algorithms of [3, 8] by Althaus and Ziegler [1]. Now, over a quarter of a century after the discovery of these results, even though much work has been done on treewidth recognition algorithms (see e.g. [2, 5, 11, 12, 13, 14, 16, 17]), these bounds on the function of k are still the best known, i.e. no $\mathcal{O}(2^{o(k^3)}n^{O(1)})$ algorithm for treewidth, and no $\mathcal{O}(2^{o(k^2)}n^{O(1)})$ algorithm for pathwidth is known. An interesting question, and a long-standing open problem in the field [4, Problem 2.7.1], is whether such algorithms can be obtained. Possible approaches to answer such a question is to design (e.g. ETH or SETH based) lower bounds, find an entirely new approach to compute treewidth or pathwidth in a parameterized setting, or improve upon the dynamic programming algorithms of [15] and [8]. Using our Merge Dominator Lemma we can go one step towards the latter, as follows.

The algorithms of Lagergren and Arnborg [15] and Bodlaender and Kloks [8] are based upon tabulating characteristics of tree or path decompositions of subgraphs of the input graph; a characteristic consists of an *intersection model*, that tells how the vertices in the current top bag interact, and for each *part* of the intersection model, a typical sequence of bag sizes.[1] The set of characteristics for a join node is computed from the sets of characteristics of its (two) children. In particular, each pair of characteristics with one from each child can give rise to exponentially (in k) many characteristics for the join node. This is because exponentially many typical sequences may arise as the merges of the typical sequences that are part of the characteristics. In the light of our Merge Dominator Lemma, only *one* of these merges has to be stored, reducing the number of characteristics arising from each pair of characteristics of the children from $2^{\mathcal{O}(k)}$ to just 1. Moreover, this dominating merge can be found in $\mathcal{O}(k)$ time, with no large constants hidden in the "$\mathcal{O}$".

Merging typical sequences at a join node is however not the only way the number of characteristics can increase throughout the algorithm, e.g. at introduce nodes, the number of characteristics increases in a different way. Nevertheless, the number of intersection models is $\mathcal{O}(k^{\mathcal{O}(k)})$ for pathwidth and $\mathcal{O}(k^{\mathcal{O}(k^2)})$ for treewidth; perhaps, with additional techniques, the number of typical sequences per part can be better bounded – in the case that a single dominating typical sequence per part suffices, this would reduce the number of table entries per node to $\mathcal{O}(k^{\mathcal{O}(k)})$ for pathwidth-k, and to $\mathcal{O}(k^{\mathcal{O}(k^2)})$ for treewidth-k, and yield $\mathcal{O}(k^{\mathcal{O}(k)}n)$ and $\mathcal{O}(k^{\mathcal{O}(k^2)}n)$ time algorithms for the respective problems.

We give direct algorithmic consequences of the Merge Dominator Lemma in the realm of computing width parameters of directed acyclic graphs (DAGs). Specifically, we show that the (Weighted) Cutwidth and Modified Cutwidth problems on DAGs, which given a directed acyclic graph on n vertices, ask for the topological order that minimizes the

[1] This approach was later used in several follow up results to obtain explicit constructive parameterized algorithms for other graph width measures, like cutwidth [18, 19], branchwidth [9], different types of search numbers like linear width [10], and directed vertex separation number [7].

cutwidth and *modified cutwidth*, respectively, can be solved in $\mathcal{O}(n^2)$ time on *series parallel digraphs*. Note that the restriction of the solution to be a *topological* order has been made as well in other works, e.g. [6].

Our algorithm for Cutwidth of series parallel digraphs has the same structure as the dynamic programming algorithm for undirected Cutwidth [6], but, in addition to obeying directions of edges, we have a step that only keeps characteristics that are not dominated by another characteristic in a table of characteristics. Now, with help of our Merge Dominator Lemma, we can show that in the case of series parallel digraphs, there is a unique dominating characteristic; the dynamic programming algorithm reverts to computing for each intermediate graph a single "optimal partial solution". This strategy also works in the presence of edge weights, which gives the algorithm for the corresponding Weighted Cutwidth problem on series parallel digraphs. Note that the cutwidth of a directed acyclic graph is at least the maximum indegree or outdegree of a vertex; e.g., a series parallel digraph formed by the parallel composition of $n-2$ paths with three vertices has n vertices and cutwidth $n-2$. To compute the *modified* cutwidth of a series parallel digraph, we give a linear-time reduction to the Weighted Cutwidth problem on series parallel digraphs.

This paper is organized as follows. In Section 2, we give a number of preliminary definitions, and review existing results, including several results on typical sequences from [8]. In Section 3, we state and prove the main technical result of this work, the Merge Dominator Lemma. Section 4 gives our algorithmic applications of this lemma, and shows that the cutwidth and modified cutwidth of a series parallel digraph can be computed in polynomial time. Some final remarks are made in Section 5.

Statements marked with "♣" are proved in the full version of the paper.

2 Preliminaries

We use the following notation. For two integers $a, b \in \mathbb{N}$ with $a \leq b$, we let $[a..b] := \{a, a+1, \ldots, b\}$ and for $a > 0$, we let $[a] := [1..a]$. If X is a set of size n, then a *linear order* is a bijection $\pi \colon X \to [n]$. Given a subset $X' \subseteq X$ of size $n' \leq n$, we define the *restriction of* π *to* X' as the bijection $\pi|_{X'} \colon X' \to [n']$ which is such that for all $x', y' \in X'$, $\pi|_{X'}(x') < \pi|_{X'}(y')$ if and only if $\pi(x') < \pi(y')$.

Sequences and Matrices. We denote the elements of a sequence s by $s(1), \ldots, s(n)$. We denote the *length* of s by $l(s)$, i.e. $l(s) := n$. For two sequences $r = r(1), \ldots, r(m)$ and $s = s(1), \ldots, s(n)$, we denote their *concatenation* by $r \circ s = r(1), \ldots, r(m), s(1), \ldots, s(n)$. For two sets of sequences R and S, we let $R \odot S := \{r \circ s \mid r \in R \wedge s \in S\}$. For a sequence s of length n and a set $X \subseteq [n]$, we denote by $s[X]$ the *subsequence of* s *induced by* X, i.e. let $X = \{x_1, \ldots, x_m\}$ be such that for all $i \in [m-1]$, $x_i < x_{i+1}$; then, $s[X] := s(x_1), \ldots, s(x_m)$. For $x_1, x_2 \in [n]$ with $x_1 \leq x_2$, we use the shorthand "$s[x_1..x_2]$" for "$s[[x_1..x_2]]$".

Let Ω be a set. A *matrix* $M \in \Omega^{m \times n}$ over Ω is said to have m rows and n columns. For sets $X \subseteq [m]$ and $Y \subseteq [n]$, we denote by $M[X, Y]$ the *submatrix* of M *induced* by X and Y, which consists of all the entries from M whose indices are in $X \times Y$. For $x_1, x_2 \in [m]$ with $x_1 \leq x_2$ and $y_1, y_2 \in [n]$ with $y_1 \leq y_2$, we use the shorthand "$M[x_1..x_2, y_1..y_2]$" for "$'M[[x_1..x_2], [y_1..y_2]]$". For a sequence $s(1), s(2), \ldots, s(\ell)$ of indices of a matrix M, we let

$$M[s] := M[s(1)], M[s(2)], \ldots, M[s(\ell)] \tag{1}$$

be the corresponding sequence of entries from M.

For illustrative purposes we enumerate the columns of a matrix in a bottom-up fashion throughout this paper, i.e. we consider the index $(1,1)$ as the "bottom left corner" and the index (m,n) as the "top right corner".

Integer Sequences. Let s be an integer sequence of length n. We use the shorthand "$\min(s)$" for "$\min_{i\in[n]} s(i)$" and "$\max(s)$" for "$\max_{i\in[n]} s(i)$"; we use the following definitions. We let

$$\operatorname{argmin}(s) \coloneqq \{i \in [n] \mid s(i) = \min(s)\} \text{ and } \operatorname{argmax}(s) \coloneqq \{i \in [n] \mid s(i) = \max(s)\}$$

be the set of indices at whose positions there are the minimum and maximum element of s, respectively. Whenever we write $i \in \operatorname{argmin}(s)$ ($j \in \operatorname{argmax}(s)$), then the choice of i (j) can be arbitrary. In some places we require a canonical choice of the position of a minimum or maximum element, in which case we will always choose the smallest index. Formally, we let

$$\operatorname{argmin}^\star(s) \coloneqq \min \operatorname{argmin}(s), \text{ and } \operatorname{argmax}^\star(s) \coloneqq \min \operatorname{argmax}(s).$$

The following definition contains two notions on pairs of integer sequences that are necessary for the definitions of domination and merges.

▶ **Definition 1.** *Let r and s be two integer sequences of the same length n.*
(i) *If for all $i \in [n]$, $r(i) \leq s(i)$, then we write "$r \leq s$".*
(ii) *We write $q = r + s$ for the integer sequence $q(1), \ldots, q(n)$ with $q(i) = r(i) + s(i)$ for all $i \in [n]$.*

▶ **Definition 2** (Extensions)**.** *Let s be a sequence of length n. We define the set $E(s)$ of* extensions *of s as the set of sequences that are obtained from s by repeating each of its elements an arbitrary number of times, and at least once. Formally, we let*

$$E(s) \coloneqq \{s_1 \circ s_2 \circ \cdots \circ s_n \mid \forall i \in [n] \colon l(s_i) \geq 1 \wedge \forall j \in [l(s_i)] \colon s_i(j) = s(i)\}.$$

▶ **Definition 3** (Domination)**.** *Let r and s be integer sequences. We say that r* dominates *s, in symbols "$r \prec s$", if there are extensions $r^* \in E(r)$ and $s^* \in E(s)$ of the same length such that $r^* \leq s^*$. If $r \prec s$ and $s \prec r$, then we say that r and s are* equivalent*, and we write $r \equiv s$.*

If r is an integer sequence and S is a set of integer sequences, then we say that r dominates *S, in symbols "$r \prec S$", if for all $s \in S$, $r \prec s$.*

▶ Remark 4 (Transitivity of "$\prec$")**.** In [8, Lemma 3.7], it is shown that the relation "$\prec$" is transitive. As this is fairly intuitive, we may use this fact without stating it explicitly throughout this text.

▶ **Definition 5** (Merges)**.** *Let r and s be two integer sequences. We define the set of all* merges *of r and s, denoted by $r \oplus s$, as $r \oplus s \coloneqq \{r^* + s^* \mid r^* \in E(r), s^* \in E(s), l(r^*) = l(s^*)\}$.*

2.1 Typical Sequences

We now define typical sequences, show how to construct them in linear time, and restate several lemmas due to Bodlaender and Kloks [8] that will be used throughout this text.

▶ **Definition 6.** *Let $s = s(1), \ldots, s(n)$ be an integer sequence of length n. The* typical sequence of *s, denoted by $\tau(s)$, is obtained from s by an exhaustive application of the following two operations:*

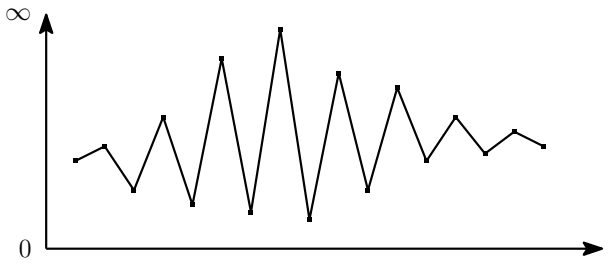


Figure 1 Illustration of the shape of a typical sequence.

Removal of Consecutive Repetitions. *If there is an index $i \in [n-1]$ such that $s(i) = s(i+1)$, then we change the sequence s from $s(1), \ldots, s(i), s(i+1), \ldots, s(n)$ to $s(1), \ldots, s(i), s(i+2), \ldots, s(n)$.*

Typical Operation. *If there exist $i, j \in [n]$ such that $j - i \geq 2$ and for all $i \leq k \leq j$, $s(i) \leq s(k) \leq s(j)$, or for all $i \leq k \leq j$, $s(i) \geq s(k) \geq s(j)$, then we change the sequence s from $s(1), \ldots, s(i), s(i+1), \ldots, s(j), \ldots, s(n)$ to $s(1), \ldots, s(i), s(j), \ldots, s(n)$, i.e. we remove all elements (strictly) between index i and j.*

To support intuition, we illustrate the rough shape of a typical sequence in Figure 1. It is not difficult to see that the typical sequence can be computed in quadratic time, by an exhaustive application of the definition. Here we discuss how to do it in linear time. We may view a typical sequence $\tau(s)$ of an integer sequence s as a subsequence of s. While $\tau(s)$ is unique, the choice of indices that induce $\tau(s)$ may not be unique. We show that we can find a set of indices that induce the typical sequence in linear time, with help of the following structural proposition.

▶ **Proposition 7 (♣).** *Let s be an integer sequence and let $i^\star \in \{\text{argmin}^\star(s), \text{argmax}^\star(s)\}$. Let $1 =: j_0 < j_1 < j_2 < \ldots < j_t < j_{t+1} := i^\star$ be pairwise distinct integers, such that $s(j_0), \ldots, s(j_{t+1})$ are pairwise distinct. If for all $h \in [0..t]$,*

- *if $s(j_h) > s(j_{h+1})$ then $j_h = \text{argmax}^\star(s[1..j_{h+1}])$ and $j_{h+1} = \text{argmin}^\star(s[1..j_{h+1}])$, and*
- *if $s(j_h) < s(j_{h+1})$ then $j_h = \text{argmin}^\star(s[1..j_{h+1}])$ and $j_{h+1} = \text{argmax}^\star(s[1..j_{h+1}])$,*

then the typical sequence of s restricted to $[i^\star]$ is equal to $s(j_0), s(j_1), \ldots, s(j_t), s(j_{t+1})$.

The idea of the algorithm is as follows. First, it is immediate that the typical sequence of s must contain its minimum and its maximum. We then observe the structure of $\tau(s)$ between $i^\star := \min \text{argmin}(s) \cup \text{argmax}(s)$ and $k^\star := \max \text{argmin}(s) \cup \text{argmax}(s)$. Next, we find a set of indices from $[i^\star]$ that satisfy the preconditions of Proposition 7 which gives indices inducing the typical sequence on $s[1..i^\star]$. By symmetry, we can again use Proposition 7 to find the indices inducing $\tau(s)$ on $s[k^\star..n]$.

▶ **Lemma 8 (♣).** *Let s be an integer sequence of length n. Then, one can compute $\tau(s)$, the typical sequence of s, in time $\mathcal{O}(n)$.*

We summarize several lemmas from [8] regarding integer sequences and typical sequences that we will use in this work.

▶ **Lemma 9** (Bodlaender and Kloks [8]). *Let r and s be two integer sequences.*

(i) *(Cor. 3.11 in [8]). We have that $r \prec s$ if and only if $\tau(r) \prec \tau(s)$.*

(ii) *(Lem. 3.13 in [8]). Suppose r and s are of the same length and let $y = r + s$. Let $r_0 \prec r$ and $s_0 \prec s$. Then there is an integer sequence $y_0 \in r_0 \oplus s_0$ such that $y_0 \prec y$.*

(iii) *(Lem. 3.14 in [8]). Let $q \in r \oplus s$. Then, there is an integer sequence $q' \in \tau(r) \oplus \tau(s)$ such that $q' \prec q$.*

(iv) *(Lem. 3.15 in [8]). Let $q \in r \oplus s$. Then, there is an integer sequence $q' \in r \oplus s$ with $\tau(q') = \tau(q)$ and $l(q') \leq l(r) + l(s) - 1$.*

(v) *(Lem. 3.19 in [8]). Let r' and s' be two more integer sequences. If $r' \prec r$ and $s' \prec s$, then $r' \circ s' \prec r \circ s$.*

2.2 Directed Acyclic Graphs

A *directed graph* (or *digraph*) G is a pair of a set of *vertices* $V(G)$ and a set of ordered pairs of vertices, called *arcs*, $A(G) \subseteq V(G) \times V(G)$. (If $A(G)$ is a multiset, we call G *multidigraph.*) We say that an arc $a = (u, v) \in A(G)$ is directed from u to v, and we call u the *tail* of a and v the *head* of a. We use the shorthand "uv" for "(u, v)". A sequence of vertices $v_1, \ldots, v_r$ is called a *walk* in G if for all $i \in [r-1]$, $v_i v_{i+1} \in A(G)$. A *cycle* is a walk $v_1, \ldots, v_r$ with $v_1 = v_r$ and all vertices $v_1, \ldots, v_{r-1}$ pairwise distinct. If G does not contain any cycles, then we call G *acyclic* or a *directed acyclic graph*, DAG for short.

Let G be a DAG on n vertices. A *topological order* of G is a linear order $\pi \colon V(G) \to [n]$ such that for all arcs $uv \in A(G)$, we have that $\pi(u) < \pi(v)$. We denote the set of all topological orders of G by $\Pi(G)$. We now define the width measures studied in this work. Note that we restrict the orderings of the vertices that we consider to *topological* orderings.

▶ **Definition 10.** *Let G be a directed acyclic graph and let $\pi \in \Pi(G)$.*

(i) *The* cutwidth *of π is* $\mathsf{cutw}(\pi) := \max_{i \in [n-1]} |\{uv \in A(G) \mid \pi(u) \leq i \wedge \pi(v) > i\}|$.

(ii) *The* modified cutwidth *of π is* $\mathsf{mcutw}(\pi) := \max_{i \in [n]} |\{uv \in A(G) \mid \pi(u) < i \wedge \pi(v) > i\}|$.

We define the cutwidth and modified cutwidth of a directed acyclic graph G as the minimum of the respective measure over all topological orders of G.

We now introduce series parallel digraphs. Note that the following definition coincides with the notion of "edge series-parallel multidigraphs" in [20].

▶ **Definition 11** (Series Parallel Digraph (SPD))**.** *A (multi-)digraph G with an ordered pair of* terminals $(s, t) \in V(G) \times V(G)$ *is called* series parallel digraph (SPD)*, often denoted by $(G, (s, t))$, if one of the following hold.*

(i) *$(G, (s, t))$ is a single arc directed from s to t, i.e. $V(G) = \{s, t\}$, $A(G) = \{(s, t)\}$.*

(ii) *$(G, (s, t))$ can be obtained from two series parallel digraphs $(G_1, (s_1, t_1))$ and $(G_2, (s_2, t_2))$ by one of the following operations.*

- **(a)** Series Composition. *$(G, (s, t))$ is obtained by taking the disjoint union of G_1 and G_2, identifying t_1 and s_2, and letting $s = s_1$ and $t = t_2$. In this case we write $(G, (s, t)) = (G_1, (s_1, t_1)) \vdash (G_2, (s_2, t_2))$ or simply $G = G_1 \vdash G_2$.*
- **(b)** Parallel Composition. *$(G, (s, t))$ is obtained by taking the disjoint union of G_1 and G_2, identifying s_1 and s_2, and identifying t_1 and t_2, and letting $s = s_1 = s_2$ and $t = t_1 = t_2$. In this case we write $(G, (s, t)) = (G_1, (s_1, t_1)) \perp (G_2, (s_2, t_2))$, or simply $G = G_1 \perp G_2$.*

It is not difficult to see that each series parallel digraph is acyclic. One can naturally associate a notion of *decomposition trees* with series parallel digraphs as follows. A decomposition tree T is a rooted and ordered binary tree whose leaves are labeled with a single arc, and each internal node $t \in V(T)$ with left child ℓ and right child r is either a *series node* or a *parallel node*. We then associate an SPD G_t with t that is $G_\ell \vdash G_r$ if t is a series node and $G_\ell \perp G_r$ if t is a parallel node. It is clear that for each SPD G, there is a decomposition tree T with root $\mathfrak{r}$ such that $G = G_\mathfrak{r}$. In that case we say that T *yields* G. Valdes et al. [20] have shown that one can decide in linear time whether a directed graph G is an SPD and if so, find a decomposition tree that yields G.

▶ **Theorem 12** (Valdes et al. [20]). *Let G be a directed graph on n vertices and m arcs. There is an algorithm that decides in time $\mathcal{O}(n+m)$ whether G is a series parallel digraph and if so, it outputs a decomposition tree that yields G.*

3 The Merge Dominator Lemma

In this section we prove the main technical result of this work. It states that given two integer sequences, one can find in linear time a merge that dominates all merges of those two sequences.

▶ **Lemma 13** (Merge Dominator Lemma). *Let r and c be integer sequences of length m and n, respectively. There exists a dominating merge of r and c, i.e. an integer sequence $t \in r \oplus c$ such that $t \prec r \oplus c$, and this dominating merge can be computed in time $\mathcal{O}(m+n)$.*

Outline of the proof. First, we show that we can restrict our search to finding a dominating path in a matrix that, roughly speaking, contains all merges of r and c of length at most $l(r)+l(c)-1$. The goal of this step is mainly to increase the intuitive insight to the proofs in this section. Next, we prove the "Split Lemma" (Lemma 19 in Section 3.2) which asserts that we can obtain a dominating path in our matrix M by splitting M into a submatrix M_1 that lies in the "bottom left" of M and another submatrix M_2 in the "top right" of M along a minimum row and a minimum column, and appending a dominating path in M_2 to a dominating path in M_1. In M_1, the last row and column are a minimum row and column, respectively, and in M_2, the first row and column are a minimum row and column, respectively. This additional structure will be exploited in Section 3.3 where we prove the "Chop Lemmas" that come in two versions. The "bottom version" (Lemma 20) shows that in M_1, we can find a dominating path by repeatedly chopping away the *last* two rows or columns and remembering a vertical or horizontal length-2 path. The "top version" (Corollary 21) is the symmetric counterpart for M_2. The proofs of the Chop Lemmas only hold when r and c are *typical sequences*, and in Section 3.4 we present the "Split-and-Chop Algorithm" that computes a dominating path in a merge matrix of two typical sequences. Finally, in Section 3.5, we generalize this result to arbitrary integer sequences, using the Split-and-Chop Algorithm and one additional construction. ◀

3.1 The Merge Matrix, Paths, and Non-Diagonality

Let us begin by defining the basic notions of a merge matrix and paths in matrices.

▶ **Definition 14** (Merge Matrix). *Let r and c be two integer sequences of length m and n, respectively. Then, the* merge matrix *of r and c is an $m \times n$ integer matrix M such that for $(i,j) \in [m] \times [n]$, $M[i,j] = r(i) + c(j)$.*

▶ **Definition 15** (Path in a Matrix). *Let M be an $m \times n$ matrix. A* path *in M is a sequence $p(1), \ldots, p(\ell)$ of indices from M such that*

(i) *$p(1) = (1,1)$ and $p(\ell) = (m,n)$, and*

(ii) *for $t \in [\ell - 1]$, let $p(t) = (i,j)$; then, $p(t+1) \in \{(i+1,j), (i,j+1), (i+1,j+1)\}$.*

We denote by $\mathcal{P}(M)$ the set of all paths in M. For two paths $p, q \in \mathcal{P}(M)$, we may simply say that p dominates *q, if $M[p]$ dominates $M[q]$.*

A path $p(1), \ldots, p(\ell)$ is called non-diagonal *if the second condition is replaced by the following.*

(ii)' *For $t \in [\ell - 1]$, let $p(t) = (i,j)$; then, $p(t+1) \in \{(i+1,j), (i,j+1)\}$.*

In analogy with extensions of integer sequences, an extension e of a path p in a matrix M is as well a sequence of indices of M, and we again denote the corresponding integer sequence by $M[e]$. A consequence of Lemma 9(i) and (iv) is that we can restrict ourselves to all paths in a merge matrix when trying to find a dominating merge of two integer sequences: it is clear from the definitions that in a merge matrix M of integer sequences r and c, $\mathcal{P}(M)$ contains all merges of r and c of length at most $l(r) + l(c) - 1$.

▶ **Corollary 16.** *Let r and c be integer sequences and M be the merge matrix of r and c. There is a dominating merge in $r \oplus c$, i.e. an integer sequence $t \in r \oplus c$ such that $t \prec r \oplus c$, if and only if there is a dominating path in M, i.e. a path $p \in \mathcal{P}(M)$ such that $p \prec \mathcal{P}(M)$.*

We now consider a type of merge that corresponds to non-diagonal paths in the merge matrix. These merges will be used in a construction presented in Section 3.5, and in the algorithmic applications of the Merge Dominator Lemma given in Section 4. For two integer sequences r and s, we denote by $r \boxplus s$ the set of all *non-diagonal merges* of r and s, which are not allowed to have "diagonal" steps: we have that for all $t \in r \boxplus s$ and all $i \in [l(t) - 1]$, if $t(i) = r(i_r) + s(i_s)$, then $t(i+1) \in \{r(i_r + 1) + s(i_s), r(i_r) + s(i_s + 1)\}$. We now show that for each merge that uses diagonal steps, there is always a non-diagonal merge that dominates it.

▶ **Lemma 17.** *Let r and s be two integer sequences of length m and n, respectively. For any merge $q \in r \oplus s$, there is a non-diagonal merge $q' \in r \boxplus s$ such that $q' \prec q$. Furthermore, given q, q' can be found in time $\mathcal{O}(m + n)$.*

We define two special paths in a matrix M that will reappear in several places throughout this section. These paths can be viewed as the "corner paths", where the first one follows the first row until it hits the last column and then follows the last column ($p_{\lrcorner}(M)$), and the second one follows the first column until it hits the last row and then follows the last row ($p_{\ulcorner}(M)$). Formally, we define them as follows:

$$p_{\lrcorner}(M) \coloneqq (1,1), (1,2), \ldots, (1,n), (2,n), \ldots, (m,n)$$
$$p_{\ulcorner}(M) \coloneqq (1,1), (2,1) \ldots, (m,1), (m,2), \ldots, (m,n)$$

We use the shorthands "$p_{\lrcorner}$" for "$p_{\lrcorner}(M)$" and "$p_{\ulcorner}$" for "$p_{\ulcorner}(M)$" whenever M is clear from the context. These paths appear in the following special cases of the Merge Dominator Lemma, which will be crucial for the proof of the Split Lemma.

▶ **Lemma 18.** *Let r and c be integer sequences of length m and n, respectively, and let M be the merge matrix of r and c. Let $i \in \operatorname{argmin}(r)$ and $j \in \operatorname{argmin}(c)$.*

(i) *If $i = 1$ and $j = n$, then $p_{\lrcorner}$ dominates all paths in M, i.e. $p_{\lrcorner} \prec \mathcal{P}(M)$.*

(ii) *If $i = m$ and $j = 1$, then $p_{\ulcorner}$ dominates all paths in M, i.e. $p_{\ulcorner} \prec \mathcal{P}(M)$.*

Proof. (i) For an illustration of this proof see the left side of Figure 2. Let q be any path in M and let $t^* \coloneqq \operatorname{argmax}^\star(q)$. Let furthermore $q(t^*) = (t^*_r, t^*_c)$. We divide $p_{\lrcorner}$ and q in three consecutive parts each to show that $p_{\lrcorner}$ dominates q.

- We let $p^1_{\lrcorner} \coloneqq p_{\lrcorner}(1), \ldots, p_{\lrcorner}(t^*_c - 1)$ and $q_1 \coloneqq q(1), \ldots, q(t^* - 1)$.
- We let $p^2_{\lrcorner} \coloneqq p_{\lrcorner}(t^*_c), \ldots, p_{\lrcorner}(n + t^*_r - 1)$ and $q_2 \coloneqq q(t^*)$.
- We let $p^3_{\lrcorner} \coloneqq p_{\lrcorner}(n + t^*_r), \ldots, p_{\lrcorner}(m + n - 1)$ and $q_3 \coloneqq q(t^* + 1), \ldots, q(l(q))$.

Since $r(1)$ is a minimum row in M, we have that for all $(k, \ell) \in [m] \times [n]$, $M[1, \ell] \leq M[k, \ell]$. This implies that there is an extension e_1 of $p^1_{\lrcorner}$ of length $t^* - 1$ such that $M[e_1] \leq M[q_1]$. Similarly, there is an extension e_3 of $p^3_{\lrcorner}$ of length $l(q) - t^*$ such that $M[e_3] \leq M[q_3]$. Finally, let f_2 be an extension of q_2 that repeats its only element, $q(t^*)$, $n - t^*_c + t^*_r$ times. Since $M[q(t^*)]$ is the maximum element on the sequence $M[q]$ and $r(1)$ is a minimum row and $c(n)$ a minimum column in M, we have that $M[p^2_{\lrcorner}] \leq M[f_2]$.

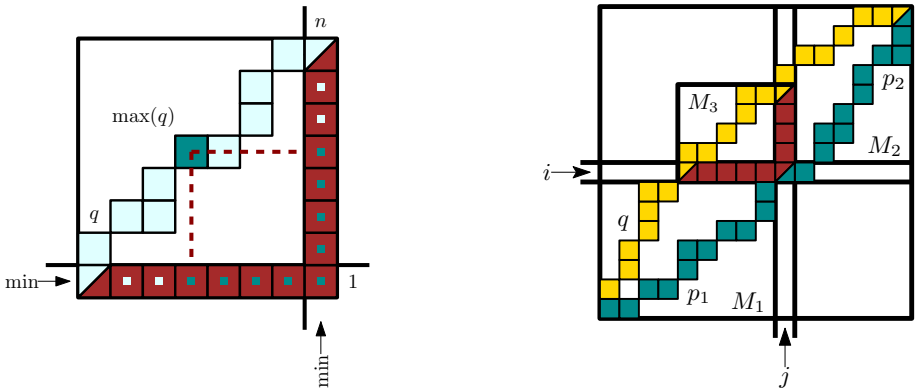


Figure 2 Illustration of the proof strategy of the Split Lemma (Lem. 19).

We define an extension e of $p_\lrcorner$ as $e := e_1 \circ p_\lrcorner^2 \circ e_3$ and an extension f of q as $f := q_1 \circ f_2 \circ q_3$. Note that $l(e) = l(f) = l(q) + n + t_r^* - (t_c^* + 1)$, and by the above discussion, we have that $M[e] \leq M[f]$. (ii) follows from a symmetric argument. ◀

3.2 The Split Lemma

In this section we prove the first main step towards the Merge Dominator Lemma. It is fairly intuitive that a dominating merge has to contain the minimum element of a merge matrix. (Otherwise, there is a path that cannot be dominated by that merge.) The Split Lemma states that in fact, we can split the matrix M into two smaller submatrices, one that has the minimum element in the top right corner, and one the has the minimum element in the bottom left corner, compute a dominating path for each of them, and paste them together to obtain a dominating path for M.

▶ **Lemma 19** (Split Lemma). *Let r and c be integer sequences of length m and n, respectively, and let M be the merge matrix of r and c. Let $i \in \operatorname{argmin}(r)$ and $j \in \operatorname{argmin}(c)$. Let $M_1 := M[1..i, 1..j]$ and $M_2 := M[i..m, j..n]$ and for all $h \in [2]$, let $p_h \in \mathcal{P}(M_h)$ be a dominating path in M_h, i.e. $p_h \prec \mathcal{P}(M_h)$. Then, $p_1 \circ p_2$ is a dominating path in M, i.e. $p_1 \circ p_2 \prec \mathcal{P}(M)$.*

Proof. Let q be any path in M. If q contains (i, j), then q has two consecutive parts, say q_1 and q_2, such that $q_1 \in \mathcal{P}(M_1)$ and $q_2 \in \mathcal{P}(M_2)$. Hence, $p_1 \prec q_1$ and $p_2 \prec q_2$, so by Lemma 9(v), $p_1 \circ p_2 \prec q_1 \circ q_2$.

Now let $p := p_1 \circ p_2$ and suppose q does not contain (i, j). Then, q either contains some (i, j') with $j' < j$, or some (i', j), for some $i' < i$. We show how to construct extensions of p and q that witness that p dominates q in the first case, and remark that the second case can be shown symmetrically. We illustrate this situation in the right side of Figure 2.

Suppose that q contains (i, j') with $j' < j$. We show that $p \prec q$. First, q also contains some (i', j), where $i' > i$. Let h_1 be the index of (i, j') in q, i.e. $q(h_1) = (i, j')$, and h_2 denote the index of (i', j) in q, i.e. $q(h_2) = (i', j)$. We derive the following sequences from q.

- We let $q_1 := q(1), \ldots, q(h_1)$ and $q_1^+ := q_1 \circ (i, j' + 1), \ldots, (i, j)$.
- We let $q_{12} := q(h_1), \ldots, q(h_2)$.
- We let $q_2 := q(h_2), \ldots, q(l(q))$ and $q_2^+ := (i, j), (i + 1, j), \ldots, (i', j) \circ q_2$.

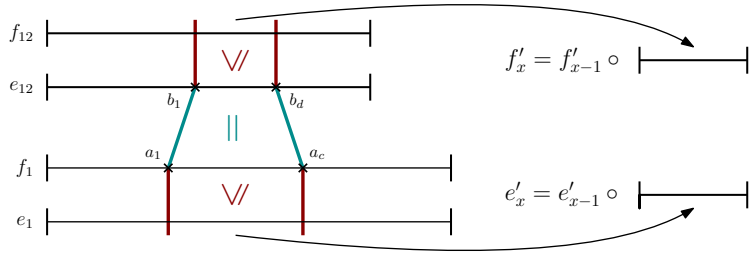


Figure 3 Constructing extensions in the proof of Lemma 19.

Since $q_1^+ \in \mathcal{P}(M_1)$ and $p_1 \prec \mathcal{P}(M_1)$, we have that $p_1 \prec q_1^+$, similarly that $p_2 \prec q_2^+$ and considering $M_3 := M[i'..i, j..j']$, we have by Lemma 18(i) that $p_{12} := p_\lrcorner(M_3) = (i, j'), (i, j'+1), \ldots, (i, j), (i+1, j), \ldots, (i', j)$ dominates q_{12}. Consequently, we consider the following extensions of these sequences.

- **(I)** We let $e_1 \in E(p_1)$ and $f_1 \in E(q_1^+)$ such that $l(e_1) = l(f_1)$ and $M[e_1] \leq M[f_1]$.
- **(II)** We let $e_{12} \in E(p_{12})$, and $f_{12} \in E(q_{12})$ such that $l(e_{12}) = l(f_{12})$ and $M[e_{12}] \leq M[f_{12}]$.
- **(III)** We let $e_2 \in E(p_2)$, and $f_2 \in E(q_2^+)$ such that $l(e_2) = l(f_2)$ and $M[e_2] \leq M[f_2]$.

We construct extensions $e' \in E(p)$ and $f' \in E(q)$ as follows. Let z be the last index in q of any element that is matched up with (i, j) in the extensions of (II). (Following the proof of Lemma 18, this would mean z is the index of $\max(q_{12})$ in q.) We first construct a pair of extensions $e'_j \in E(p_1)$, and $f'_j \in E(q[1..z])$ with $l(e'_j) = l(f'_j)$ and $M[e'_j] \leq M[f'_j]$. With a symmetric procedure, we can obtain extensions of p_2 and of $q[(z+1)..l(q)]$, and use them to obtain extensions of $p = p_1 \circ p_2$ and $q = q[1..z] \circ q[(z+1)..l(q)]$ witnessing that $p \prec q$.

We give the details of the first part of the construction. Let a be the index of the last repetition in f_1 of $q(h_1 - 1)$, i.e. the index that appears just before $q(h_1) = (i, j')$ in f_1. We let $e'_{j'-1}[1..a] := e_1[1..a]$ and $f'_{j'-1}[1..a] := f_1[1..a]$. By (I), $M[e'_{j'-1}] \leq M[f'_{j'-1}]$.

For $x = j', j'+1, \ldots, j$, we inductively construct e'_x and f'_x using e'_{x-1} and f'_{x-1}, for an illustration see Figure 3. We maintain as an invariant that $l(e'_{x-1}) = l(f'_{x-1})$ and that $M[e'_{x-1}] \leq M[f'_{x-1}]$. Let $a_1, \ldots, a_c$ denote the indices of the occurrences of (i, x) in f_1, and $b_1, \ldots, b_d$ denote the indices of the occurrences of (i, x) in e_{12}. We let:

$$e'_x := e'_{x-1} \circ e_1[a_1, \ldots, a_c]; f'_x := f'_{x-1} \circ f_{12}[b_1, \ldots, b_d], \qquad \text{if } c = d$$

$$e'_x := e'_{x-1} \circ e_1[a_1, \ldots, a_c] \circ \overbrace{e_1(a_c), \ldots, e_1(a_c)}^{d-c \text{ times}}; f'_x := f'_{x-1} \circ f_{12}[b_1, \ldots, b_d], \qquad \text{if } c < d$$

$$e'_x := e'_{x-1} \circ e_1[a_1, \ldots, a_c]; f'_x := f'_{x-1} \circ f_{12}[b_1, \ldots, b_d] \circ \overbrace{f_{12}(b_d), \ldots, f_{12}(b_d)}^{c-d \text{ times}}, \qquad \text{if } c > d$$

In each case, we extended e'_{x-1} and f'_{x-1} by the same number of elements; furthermore we know by (I) that for $y \in \{a_1, \ldots, a_c\}$, $M[e_1(y)] \leq M[f_1(y)]$, by choice we have that for all $y' \in \{b_1, \ldots, b_d\}$, $f_1(y) = e_{12}(y')$ and we know that $M[e_{12}(y')] \leq M[f_{12}(y')]$ by (II). Hence, $M[e'_x] \leq M[f'_x]$ in either of the above cases. In the end of this process, we have $e'_j \in E(p_1)$ and $f'_j \in E(q[1..z])$, and by construction, $l(e'_j) = l(f'_j)$ and $M[e'_j] \leq M[f'_j]$. ◀

3.3 The Chop Lemmas

Assume the notation of the Split Lemma. If we were to apply it recursively, it only yields a size-reduction whenever $(i, j) \notin \{(1, 1), (m, n)\}$. Motivated by this issue, we prove two more lemmas to deal with the cases when $(i, j) \in \{(1, 1), (m, n)\}$, and we coin them the "Chop

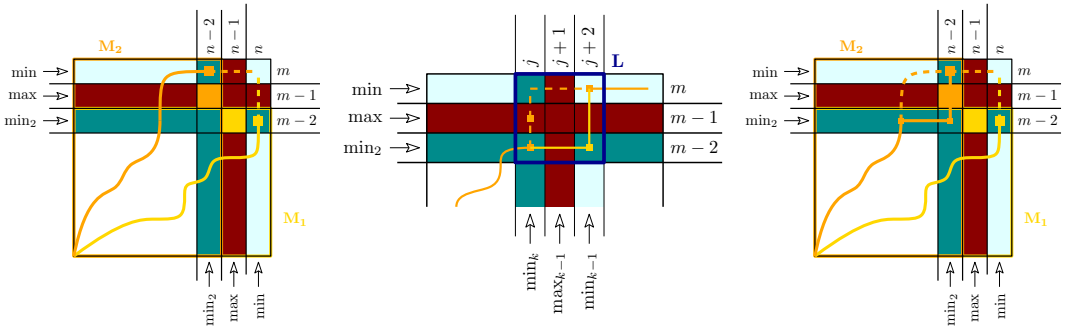


Figure 4 Illustration of the ideas in the main step of the proof of the bottom version of the Chop Lemmas (Lem. 20).

Lemmas". It will turn out that when applied to typical sequences, a repeated application of these lemmas yields a dominating path in M. This insight crucially helps in arguing that the dominating path in a merge matrix can be found in *linear* time. We would like to stress that up to this point, all results in this section were shown in terms of arbitrary integer sequences. For the next lemma, we require the sequences considered to be *typical sequences*. In Section 3.5 we will generalize the results that rely on the following lemmas to arbitrary integer sequences.

▶ **Lemma 20** (Chop Lemma – Bottom, ♣). *Let r and c be typical sequences of length $m \geq 3$ and $n \geq 3$, respectively, and let M be the merge matrix of r and c. Suppose that $m \in \operatorname{argmin}(r)$ and $n \in \operatorname{argmin}(c)$ and let $M_1 := M[1..(m-2), 1..n]$ and $M_2 := M[1..m, 1..(n-2)]$ and for all $h \in [2]$, let $p_h \prec \mathcal{P}(M_h)$. Let $p_1^+ := p_1 \circ (m-1, n), (m, n)$ and $p_2^+ := p_2 \circ (m, n-1), (m, n)$.*

(i) *If $M[m-2, n-1] \leq M[m-1, n-2]$, then $p_1^+ \prec \mathcal{P}(M)$.*

(ii) *If $M[m-1, n-2] \leq M[m-2, n-1]$, then $p_2^+ \prec \mathcal{P}(M)$.*

Outline of the proof. We first argue that each path in M is dominated by at least one of p_1^+ and p_2^+. If a path contains either $(m-2, n)$, or $(m, n-2)$, then it is easily seen that it is dominated by p_1^+ or p_2^+, respectively. If a path does not contain either of them, then there always is a path that dominates it, and contains either $(m-2, n)$ or $(m, n-2)$. This is due to the fact that $m-1$ is a maximum row and $n-1$ a maximum column.

Next, consider the setting of (i), and for an illustration see Figure 4, beginning with the left hand side. Most effort is spent in proving that p_1^+ dominates p_2^+ under the stated assumption. Towards this claim, we first show that we can find a path p_2' in M_2 that uses $(m-2, n-2)$, and dominates p_2. This is done by considering situations such as in the middle of Figure 4. Assume the notation used there, and note that p_2 uses the element (m, j). Since $M[m-2, n-1] \leq M[m-1, n-2]$, and using the structure of typical sequences ending in the minimum element, we can conclude that $M[m-2, j+1] \leq M[m-1, j]$. We then show that in the 3×3-submatrix L, the corner path $p_{\lrcorner}(L)$ is a dominating path. Using the corresponding extensions, we show that the path obtained from p_2 by replacing the $p_{\ulcorner}(L)$ path with the $p_{\lrcorner}(L)$ path, dominates p_2. Iteratively applying such arguments and transitivity of "$\prec$" lets us conclude that there is a path p_2' in M_2 that uses $(m-2, n-2)$, and dominates p_2, see the right hand side of Figure 4. Now, let p_2'' be the subpath of p_2' ending in $(m-2, n-2)$, and note that $p_2'' \circ (m-2, n-1), (m-2, n) \in \mathcal{P}(M_1)$. Then,

$$p_1^+ \prec p_2'' \circ (m-2, n-1), (m-2, n), (m-1, n), (m, n) \tag{2}$$
$$\prec p_2' \circ (m, n-1), (m, n) \tag{3}$$
$$\prec p_2^+, \tag{4}$$

Algorithm 1 The Split-and-Chop Algorithm.

Input : Typical sequences $r(1), \ldots, r(m)$ and $c(1), \ldots, c(n)$
Output : A dominating merge of r and c

1 Let $i \in \operatorname{argmin}(r)$ and $j \in \operatorname{argmin}(c)$;
2 **return** `Chop-bottom` $(r[1..i], c[1..j]) \circ$ `Chop-top` $(r[i..m], c[j..n])$;
3 **Procedure** `Chop-bottom`(r *and* c *as above*)
4 **if** $m \leq 2$ **then return** $r(1)+c(1), r(m)+c(1), r(m)+c(2), \ldots, r(m)+c(n)$;
5 **if** $n \leq 2$ **then return** $r(1)+c(1), r(1)+c(n), r(2)+c(n), \ldots, r(m)+c(n)$;
6 **if** $r(m-2)+c(n-1) \leq r(m-1)+c(n-2)$ **then return**
`Chop-bottom`$(r[1..(m-2)], c) \circ (r(m-1)+c(n)), r(m)+c(n)$;
7 **if** $r(m-1)+c(n-2) \leq r(m-2)+c(n-1)$ **then return**
`Chop-bottom`$(r, c[1..(n-2)]) \circ (r(m)+c(n-1)), r(m)+c(n)$;
8 **Procedure** `Chop-top`(r *and* c *as above*)
9 **if** $m \leq 2$ **then return** $r(1)+c(1), r(1)+c(2), \ldots, r(1)+c(n), r(m)+c(n)$;
10 **if** $n \leq 2$ **then return** $r(1)+c(1), r(2)+c(1), \ldots, r(m)+c(1), r(m)+c(n)$;
11 **if** $r(3)+c(2) \leq r(2)+c(3)$ **then return**
$r(1)+c(1), (r(2)+c(1)) \circ$ `Chop-top`$(r[3..m], c)$;
12 **if** $r(2)+c(3) \leq r(3)+c(2)$ **then return**
$r(1)+c(1), (r(1)+c(2)) \circ$ `Chop-top`$(r, c[3..n])$;

where (2) is due to $p_1 \prec \mathcal{P}(M_1)$ and therefore $p_1 \prec p_2'' \circ (m-2, n-1), (m-2, n)$, next (3) follows from another application of the 3×3-subcase, and (4) is guaranteed since $p_2' \prec p_2$ by the iterative construction sketched above. ◀

By symmetry, these arguments also prove the "top" case of the Chop Lemmas.

▶ **Corollary 21** (Chop Lemma - Top). *Let r and c be typical sequences of length $m \geq 3$ and $n \geq 3$, respectively, and let M be the merge matrix of r and c. Suppose that $1 \in \operatorname{argmin}(r)$ and $1 \in \operatorname{argmin}(c)$ and let $M_1 \coloneqq M[3..m, 1..n]$ and $M_2 \coloneqq M[1..m, 3..n]$ and for all $h \in [2]$, let $p_h \prec \mathcal{P}(M_h)$. Let $p_1^+ \coloneqq (1,1), (2,1) \circ p_1$ and $p_2^+ \coloneqq (1,1), (1,2) \circ p_2$.*

(i) *If $M[3,2] \leq M[2,3]$, then $p_1^+ \prec \mathcal{P}(M)$.*

(ii) *If $M[2,3] \leq M[3,2]$, then $p_2^+ \prec \mathcal{P}(M)$.*

3.4 The Split-and-Chop Algorithm

Equipped with the Split Lemma and the Chop Lemmas, we are now ready to give the algorithm that computes a dominating merge of two typical sequences. Consequently, we call this algorithm the "Split-and-Chop Algorithm", the details are given in Algorithm 1. Note that the base cases (lines 4, 5, 9, and 10) are easily justified: in the bottom case, the last row and column are minimum, and in the top case, the first row and column are minimum.

▶ **Lemma 22** (♣). *Let r and c be typical sequences of length m and n, respectively. Then, there is an algorithm that finds in $\mathcal{O}(m+n)$ time a dominating path in the merge matrix of r and c.*

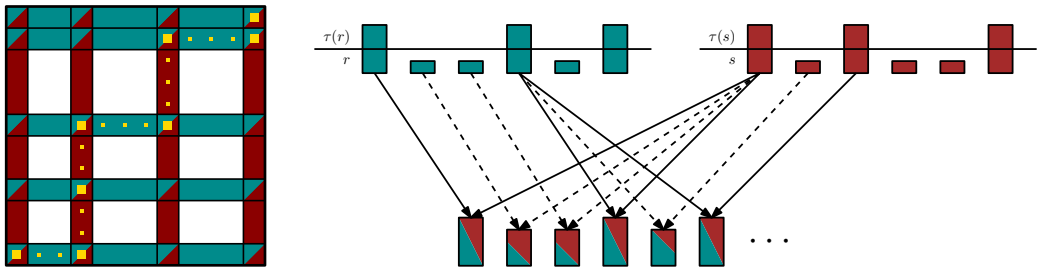


Figure 5 Illustration of the typical lift. On the left side, the view of the merge matrix M, with the rows and columns corresponding to elements of the typical sequences highlighted. Inside there, M_τ can be seen as a highlighted submatrix. The merge t' is depicted as the large yellow squares within M_τ and the small yellow squares outside of M_τ show its completion to the typical lift of t. On the right side, an illustration that does not rely on the "matrix view".

3.5 Generalization to Arbitrary Integer Sequences

In this section we show how to generalize Lemma 22 to arbitrary integer sequences. In particular, we will show how to construct from a merge of two typical sequences $\tau(r)$ and $\tau(s)$ that dominates all of their merges, a merge of r and s that dominates all merges of r and s. The claimed result then follows from an application of Lemma 22. For an illustration of the following construction, see Figure 5.

The Typical Lift. Let r and s be integer sequences and let $t \in \tau(r) \oplus \tau(s)$. Then, the *typical lift of* t, denoted by $\rho(t)$, is an integer sequence $\rho(t) \in r \oplus s$, obtained from t as follows. For convenience, we will consider $\rho(t)$ as a path in the merge matrix M of r and s.

Step 1. We construct $t' \in \tau(r) \boxplus \tau(s)$ such that $t' \prec t$ using Lemma 17. Throughout the following, consider t' to be a path in the merge matrix M_τ of $\tau(r)$ and $\tau(s)$.

Step 2. First, we initialize $\rho_t^1 := t'(1) = (1,1)$. For $i = \{2, \ldots, l(t')\}$, we proceed inductively as follows. Let $(i_r, i_s) = t(i)$ and let $(i'_r, i'_s) = t(i-1)$. (Note that $t(i-1)$ and $t(i)$ are indices in M_τ.) Let furthermore (j_r, j_s) be the index in M corresponding to (i_r, i_s), and let (j'_r, j'_s) be the index in M corresponding to (i'_r, i'_s). Assume by induction that $\rho_t^{i-1} \in \mathcal{P}(M[1..j'_r, 1..j'_s])$. We show how to extend ρ_t^{i-1} to a path in ρ_t^i in $M[1..j_r, 1..j_s]$. Since t' is non-diagonal, we have that $(i'_r, i'_s) \in \{(i_r - 1, i_s), (i_r, i_s - 1)\}$, so one of the two following cases applies.

Case S2.1 ($i'_r = i_r - 1$ and $i'_s = i_s$). In this case, we let $\rho_t^i := \rho_t^{i-1} \circ (j'_r + 1, j_s), \ldots, (j_r, j_s)$.

Case S2.2 ($i'_r = i_r$ and $i'_s = i_s - 1$). In this case, we let $\rho_t^i := \rho_t^{i-1} \circ (j_r, j'_s + 1), \ldots, (j_r, j_s)$.

Step 3. We return $\rho(t) := \rho_t^{l(t')}$.

The following lemma captures the desired property of the typical lift, and its proof essentially follows from Lemmas 9(iii) and 17.

▶ **Lemma 23 (♣).** *Let r and s be integer sequences and let $q \in r \oplus s$. Let $t \in \tau(r) \oplus \tau(s)$ such that $t \prec \tau(r) \oplus \tau(s)$. Then, $\rho(t) \prec q$ (and $\rho(t) \in r \boxplus s$).*

We can now prove the main result of this work. Note that the following lemma is a strengthening of Lemma 13 in that it shows that there is always a non-diagonal dominating merge. This will be important for the algorithmic applications given in Section 4.

▶ **Lemma 24.** *Let r and c be integer sequences of length m and n, respectively. There exists a dominating non-diagonal merge of r and c, i.e. an integer sequence $t \in r \boxplus c$ such that $t \prec r \oplus c$, and this dominating merge can be computed in time $\mathcal{O}(m+n)$.*

Proof. The algorithm proceeds in the following steps.

Step 1. Compute $\tau(r)$ and $\tau(c)$.
Step 2. Apply the Split-and-Chop Algorithm on input $(\tau(r), \tau(c))$ to obtain $t \prec \tau(r) \oplus \tau(c)$.
Step 3. Return the typical lift $\rho(t)$ of t.

Correctness of the above algorithm follows from Corollary 16 and Lemmas 22 and 23 which together guarantee that $\rho(t) \prec r \oplus c$, and that $\rho(t)$ is a non-diagonal merge, i.e. $\rho(t) \in a \boxplus b$. By Lemma 8, Step 1 can be done in time $\mathcal{O}(m+n)$, by Lemma 22, Step 2 takes time $\mathcal{O}(m+n)$ as well, and the typical lift of t can also be computed in time $\mathcal{O}(m+n)$. The overall runtime of the algorithm is $\mathcal{O}(m+n)$. ◀

4 Directed Width Measures of Series Parallel Digraphs

In this section, we give algorithmic consequences of the Merge Dominator Lemma. We discuss the (weighted) cutwidth on series parallel digraphs problem in Section 4.1 and (briefly) the modified cutwidth on SPD's problem in Section 4.2.

4.1 Cutwidth

In this section we provide an $\mathcal{O}(n^2)$ time algorithm for the problem of computing the cutwidth of a series parallel digraph on n vertices.

Cutwidth of Series Parallel Digraphs

Input:	A series parallel digraph G.
Question:	What is the cutwidth of G?

Given a series parallel digraph G, we follow a bottom-up dynamic programming scheme along the decomposition tree T that yields G. Each node $t \in V(T)$ has a subgraph G_t of G associated with it, that is also series parallel. Naturally, we use the property that G_t is obtained either via series or parallel composition of the SPD's associated with its two children.

To make this problem amenable to be solved using merges of integer sequences, we define the following notion of a cut-size sequence of a topological order of a directed acyclic graph which records for each position in the order, how many arcs cross it.

▶ **Definition 25** (Cut-Size Sequence). *Let G be a directed acyclic graph on n vertices and let π be a topological order of G. The sequence $x_1, \ldots, x_{n-1}$, where for $i \in [n-1]$,*

$$x_i = |\{uv \in A(G) \mid \pi(u) \leq i \wedge \pi(v) > i\}|,$$

is the cut-size sequence *of π, and denoted by $\sigma(\pi)$. For a set of topological orders $\Pi' \subseteq \Pi(G)$, we let $\sigma(\Pi') \coloneqq \{\sigma(\pi) \mid \pi \in \Pi'\}$.*

It is clear that when a cut-size sequence $\sigma(\pi_1)$ dominates another cut-size sequence $\sigma(\pi_2)$, then $\mathsf{cutw}(\pi_1) \leq \mathsf{cutw}(\pi_2)$. The proof of the next theorem goes via the following two steps. First, we show that in the dynamic programming algorithm, it is sufficient to store a dominating cut-size sequence. Second, suppose that an SPD is obtained from two smaller SPD's G_1 and G_2, and let π_1 and π_2 be topological orders of G_1 and G_2, respectively, such that $\sigma(\pi_1)$ dominates all cut-size sequences of G_1, and $\sigma(\pi_2)$ dominates all cut-size sequences of G_2. For the case $G = G_1 \vdash G_2$, we show that $\pi_1 \circ \pi_2$ yields a topological order that

dominates all cut-size sequences of G, and for the case $G = G_1 \perp G_2$, we show that the topological order π of G such that $\sigma(\pi)$ dominates $\sigma(\pi_1) \boxplus \sigma(\pi_2)$ also dominates all cut-size sequences of G.

The algorithm of the following theorem in fact works for the more general problem of computing the *weighted* cutwidth of a series parallel digraph, where we are also given an arc-weight function and the value of a cut is computed as the sum of the weights of the arcs crossing the cut.

▶ **Theorem 26 (♣).** *Let G be an SPD on n vertices (together with an arc-weight function). There is an algorithm that computes in time $\mathcal{O}(n^2)$ the (weighted) cutwidth of G, and outputs a topological ordering that achieves the upper bound.*

4.2 Modified Cutwidth

We now consider the following computational problem.

Modified Cutwidth of Series Parallel Digraphs

Input:	A series parallel digraph G.
Question:	What is the modified cutwidth of G?

In the full version, we provide a transformation that given a series parallel digraph G on n vertices, outputs an instance of Weighted Cutwidth on series parallel digraphs whose digraph has $\mathcal{O}(n)$ vertices that we can use to determine the modified cutwidth of G. Using the algorithm for Weighted Cutwidth on series parallel digraphs due to Theorem 26, we have the following result.

▶ **Theorem 27 (♣).** *Let G be an SPD on n vertices. There is an algorithm that computes in time $\mathcal{O}(n^2)$ the modified cutwidth of G, and outputs a topological ordering of G that achieves the upper bound.*

5 Conclusions

In this paper, we obtained a new technical insight in a now over a quarter century old technique, namely the use of typical sequences. The insight led to new polynomial time algorithms. Since its inception, algorithms based on typical sequences give the best asymptotic bounds for linear time FPT algorithms for treewidth and pathwidth, as functions of the target parameter. It still remains a challenge to improve upon these bounds ($2^{O(pw^2)}$, respectively $2^{O(tw^3)}$), or give non-trivial lower bounds for parameterized pathwidth or treewidth. Possibly, the Merge Dominator Lemma can be helpful to get some progress here.

As other open problems, we ask whether there are other width parameters for which the Merge Dominator Lemma implies polynomial time or XP algorithms, or whether such algorithms exist for other classes of graphs. For instance, for which width measures can we give XP algorithms when parameterized by the treewidth of the input graph?

References

1 Ernst Althaus and Sarah Ziegler. Optimal tree decompositions revisited: A simpler linear-time FPT algorithm, 2019. arXiv:1912.09144.

2 Eyal Amir. Approximation algorithms for treewidth. *Algorithmica*, 56(4):448–479, 2010.

3 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

4 Hans L. Bodlaender, Leizhen Cai, Jianer Chen, Michael R. Fellows, Jan Arne Telle, and Dániel Marx. Open problems in parameterized and exact computation – IWPEC 2006. Technical Report UU-CS-2006-052, Department of Information and Computing Sciences, Utrecht University, 2006.

5 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michał Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016.

6 Hans L. Bodlaender, Michael R. Fellows, and Dimitrios M. Thilikos. Derivation of algorithms for cutwidth and related graph layout parameters. *J. Comput. Syst. Sci.*, 75(4):231–244, 2009.

7 Hans L. Bodlaender, Jens Gustedt, and Jan Arne Telle. Linear-time register allocation for a fixed number of registers. In *Proc. 9th SODA*, pages 574–583. ACM/SIAM, 1998.

8 Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996.

9 Hans L. Bodlaender and Dimitrios M. Thilikos. Constructive linear time algorithms for branchwidth. In *Proc. 24th ICALP*, volume 1256 of *LNCS*, pages 627–637. Springer, 1997.

10 Hans L. Bodlaender and Dimitrios M. Thilikos. Computing small search numbers in linear time. In *Proc. 1st IWPEC*, volume 3162 of *LNCS*, pages 37–48. Springer, 2004.

11 Mikolaj Bojanczyk and Michal Pilipczuk. Optimizing tree decompositions in MSO. In *Proc. 34th STACS*, volume 66 of *LIPIcs*, pages 15:1–15:13, 2017.

12 Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM J. Comput.*, 38(2):629–657, 2008.

13 Martin Fürer. Faster computation of path-width. In *Proc. 27th IWOCA*, volume 9843 of *LNCS*, pages 385–396. Springer, 2016.

14 Jens Lagergren. Efficient parallel algorithms for graphs of bounded tree-width. *J. Algorithms*, 20(1):20–44, 1996.

15 Jens Lagergren and Stefan Arnborg. Finding minimal forbidden minors using a finite congruence. In *Proc. 18th ICALP*, volume 510 of *LNCS*, pages 532–543. Springer, 1991.

16 Bruce A. Reed. Finding approximate separators and computing tree width quickly. In *Proc. 24th STOC*, pages 221–228. ACM, 1992.

17 Neil Robertson and Paul D. Seymour. Graph minors. XIII. The disjoint paths problem. *J. Combin. Theory, Ser. B*, 63(1):65–110, 1995.

18 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.

19 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.

20 Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. The recognition of series-parallel digraphs. *SIAM J. Comput.*, 11(2):298–313, 1982.

Grundy Coloring & Friends, Half-Graphs, Bicliques

Pierre Aboulker
DI/ENS, PSL University, Paris, France
pierreaboulker@gmail.com

Édouard Bonnet
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France
edouard.bonnet@ens-lyon.fr

Eun Jung Kim
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France
eun-jung.kim@dauphine.fr

Florian Sikora
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France
florian.sikora@dauphine.fr

Abstract

The *first-fit* coloring is a heuristic that assigns to each vertex, arriving in a specified order σ, the smallest available color. The problem Grundy Coloring asks how many colors are needed for the most adversarial vertex ordering σ, i.e., the maximum number of colors that the first-fit coloring requires over all possible vertex orderings. Since its inception by Grundy in 1939, Grundy Coloring has been examined for its structural and algorithmic aspects. A brute-force $f(k)n^{2^{k-1}}$-time algorithm for Grundy Coloring on general graphs is not difficult to obtain, where k is the number of colors required by the most adversarial vertex ordering. It was asked several times whether the dependency on k in the exponent of n can be avoided or reduced, and its answer seemed elusive until now. We prove that Grundy Coloring is W[1]-hard and the brute-force algorithm is essentially optimal under the Exponential Time Hypothesis, thus settling this question by the negative.

The key ingredient in our W[1]-hardness proof is to use so-called *half-graphs* as a building block to transmit a color from one vertex to another. Leveraging the half-graphs, we also prove that b-Chromatic Core is W[1]-hard, whose parameterized complexity was posed as an open question by Panolan et al. [JCSS '17]. A natural follow-up question is, how the parameterized complexity changes in the absence of (large) half-graphs. We establish fixed-parameter tractability on $K_{t,t}$-free graphs for b-Chromatic Core and Partial Grundy Coloring, making a step toward answering this question. The key combinatorial lemma underlying the tractability result might be of independent interest.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Fixed parameter tractability

Keywords and phrases Grundy coloring, parameterized complexity, ETH lower bounds, $K_{t,t}$-free graphs, half-graphs

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.58

Related Version A full version of the paper is available at `http://arxiv.org/abs/2001.03794`.

Acknowledgements A part of this work was done while the authors attended the "2019 IBS Summer research program on Algorithms and Complexity in Discrete Structures", hosted by the IBS discrete mathematics group. The third and the fourth authors partially supported by the ANR project "ESIGMA" (ANR-17-CE23-0010).

1 Introduction

A coloring is said *proper* if no two adjacent vertices receive the same color. The *chromatic number* of a graph G denoted $\chi(G)$ is the minimum number of colors required to properly color G. Let us now consider a natural heuristic to build a proper coloring of a graph G. Given an ordering σ of the vertices of G, consider each vertex of G in the order σ and assign to the current vertex the smallest possible color (without creating any conflict), i.e., the smallest color *not* already given to one of its already colored neighbors. The obtained coloring is obviously proper and it is called a *first-fit* or *greedy* coloring. The Grundy number, denoted by $\Gamma(G)$, is the *largest* number of colors used by the first-fit coloring on some ordering of the vertices of G. Thus $\Gamma(G)$ is an upper-bound to the output of a first-fit heuristic.

The Grundy number has been introduced in 1939 [18], but was formally defined only forty years ago, independently by Christen and Selkow [9] and by Simmons [32]. Grundy Coloring in directed graphs already appears as a NP-complete problem in the monograph of Garey and Johnson [15]. The undirected version remains NP-hard on bipartite graphs [21] and their complements [35], chordal graphs [31] and line graphs [20]. When the input is a tree, Grundy Coloring can be solved in linear time [23]. This result is generalized to bounded-treewidth graphs with an algorithm running in time $k^{O(w)}2^{O(wk)}n = O(n^{3w^2})$ for graphs of treewidth w and Grundy number k [33], but this cannot be improved to $O^*(2^{o(w \log w)})$ under the ETH [4]. It is also possible to solve Grundy Coloring in time $O^*(2.443^n)$ [4].

In 2006, Zaker [36] observed that since a minimal witness (we will formally define a witness later) for Grundy number k has size at most 2^{k-1}, the brute-force approach gives an algorithm running in time $f(k)n^{2^{k-1}}$, that is, an XP algorithm in the words of parameterized complexity. Since then it has been open whether Grundy Coloring can be solved in FPT time, i.e., $f(k)n^{O(1)}$ (where the exponent does not depend on k). FPT algorithms were obtained in chordal graphs, claw-free graphs, and graphs excluding a fixed minor [4], or with respect to the dual parameter $n - k$ [21]. The parameterized complexity of Grundy Coloring in general graphs was raised as an open question in several papers in the past decade [31, 22, 16, 4].

Closely related to Grundy coloring is the notion of *partial Grundy coloring* and *b-coloring*. Let $G = (V, E)$ be a graph. We say that a proper coloring $V_1 \uplus \cdots \uplus V_k$ is a *partial Grundy coloring* of order k if there exists $v_i \in V_i$ for each $i \in [k]$ such that v_i has a neighbor in every V_j with $j < i$. The problem Partial Grundy Coloring takes a graph G and a positive integer k, and asks if there is a partial Grundy coloring of order k. Erdős et al. [13] showed that the partial Grundy number coincides with the so-called *upper ochromatic* number. This echoes another result of Erdős et al. [12] that Grundy number and ochromatic number (introduced by Simmons [32]) are the same.

The *b-chromatic core of order k* of a graph G is a vertex-subset C of G with the following property: C admits a partition into $V_1 \uplus \cdots \uplus V_k$ such that there is $v_i \in V_i$ for each $i \in [k]$ which contains a neighbor in every V_j with $j \neq i$. The goal of the problem b-Chromatic Core is to determine whether an input graph G contains a b-chromatic core of order k. This notion was studied in [11, 30] in relation to *b-coloring*, which is a proper coloring such that for every color i, there is a vertex of color i which neighbors a vertex of every other color. The maximum number k such that G admits a b-coloring with k colors is called the *b-chromatic number* of G. In [30], it was proven that deciding whether a graph G has b-chromatic number at least k is $W[1]$-hard parameterized by k. The problem might be even harder since no polytime algorithm is known when k is constant. The authors left it as an open question whether b-Chromatic Core is W[1]-hard or FPT.

Our contribution: the half-graph is key. We prove that Grundy Coloring is W[1]-complete, thus settling the open question posed in [31, 22, 16, 4]. More quantitatively we show that the double-exponential XP algorithm is essentially optimal. Indeed we prove that there is no computable function f such that Grundy Coloring is solvable in time $f(k)n^{o(2^{k-\log k})}$, unless the ETH fails. This further answers by the negative an alternative question posted in [31, 22], whether there is an algorithm in time $n^{k^{O(1)}}$.

A key element in the hardness proof of Grundy Coloring is what we call a *half-graph* (definition in Section 2.1). The main obstacle encountered when one sets out to prove W[1]-hardness of Grundy Coloring is the difficulty of propagating a chosen color from a vertex to another while keeping the Grundy number low (i.e., bounded by a function of k). Employing half-graphs turns out to be crucial to circumvent this obstacle, which we further examine in Section 4. Leveraging half-graphs as color propagation apparatus, we also prove that b-Chromatic Core is W[1]-complete (albeit with a very different construction). This settles the question posed by [30].

Our contribution: delineating the boundary of tractability. All three problems, Grundy Coloring, Partial Grundy Coloring, and b-Chromatic Core are FPT for $k = \Gamma(G)$ on nowhere dense graphs. The existence of each induced witness can be expressed as a first-order formula on at most 2^{k-1} variables in the case of Grundy Coloring, and on at most k^2 variables in the case of Partial Grundy Coloring and b-Chromatic Core. The problem is therefore expressible in first-order logic as a disjunction of the existence of every induced witness while the number of induced witnesses is bounded by $2^{2^{2(k-1)}}$. And first-order formulas can be decided in FPT time on nowhere dense graphs [17]. The next step is $K_{t,t}$-free graphs, i.e., those graphs without a biclique $K_{t,t}$ as a (non necessarily induced) subgraph, which is a dense graph class that contains nowhere dense graphs and graphs of bounded degeneracy. In the realm of parameterized complexity, $K_{t,t}$-free graphs have been observed to admit FPT algorithms for otherwise W[1]-hard problems [34].

We prove that Partial Grundy Coloring and b-Chromatic Core are fixed-parameter tractable on $K_{t,t}$-free graphs, even in the parameter $k + t$, now assuming that t is not a fixed constant. To this end, a combinatorial lemma plays a crucial role by letting us rule out the case when many vertices have large degree: if there are many vertices of large degree in a $K_{t,t}$-free graph, one can find a collection of k vertex-disjoint and pairwise non-adjacent stars on k-vertices, which is a witness for b-Chromatic Core and Partial Grundy Coloring. Now, we can safely confine the input instances to have bounded degrees, save a few vertices. We present an FPT algorithm that works under this setting.

Statements marked with a ♠ symbol have their proof entirely deferred to the long version, while statements marked with a ♣ come with a proof sketch or a partial proof. All the missing proofs can be found in the full version [2] (or in one case in [4]).

2 Preliminaries

For any integer i, j, we denote by $[i, j]$ the set of integers that are at least i and at most j, and $[i]$ is a short-hand for $[1, i]$. We use the standard graph notations [10]: for a graph G, $V(G)$ denotes the set of vertices of G, $E(G)$ denotes the set of edges. A vertex u is a neighbor of v if uv is an edge of G. The *open neighborhood* of a vertex v is the set of all neighbors of v and $N[v]$ denotes the *closed neighborhood* of v defined as $N(v) \cup \{v\}$. The open (closed, respectively) neighborhood of a vertex-set S is $\bigcup_{v \in S} N(v) \setminus S$ ($\bigcup_{v \in S} N(v) \cup S$, respectively). For a vertex-set $Y \subseteq V(G)$, we denote $N(v) \cap Y$ ($N[v] \cap Y$, respectively) simply as $N_Y(v)$

($N_Y[v]$, respectively) and the same applies the open and closed neighborhood of a vertex-set S. For two disjoint vertex-sets X and Y, we say that X is (anti-)complete with Y if every vertex of X is (non-)adjacent with every vertex of Y.

2.1 Half-graphs

We call *anti-matching* the complement of an induced matching. The *anti-matching of height t* is the complement of t edges. It will soon be apparent that all the coloring numbers considered in this paper are lowerbounded by t, in presence of an anti-matching of height t. Therefore in the subsequent FPT reductions, we will not have the luxury to have anti-matchings of unbounded size. This will constitute an issue since they are useful to propagate choices. Imagine we have two sets A and B of size unbounded by the parameter, and we want to relate a choice in A to the same choice in B. Let us put an antimatching between A and B. Trivially independent sets of size 2 will correspond to consistent choices. So it all boils down to expressing our problem in terms of finding large enough independent sets. Now this option is not available, another way to propagate choices is to use *half-graphs*.

We call *half-graph* a graph whose vertices can be partitioned into (A, B) such that there is no induced $2K_2$ in the graph induced by the edges with one endpoint in A and the other endpoint in B, and $G[A]$ and $G[B]$ are both edgeless. These graphs are sometimes called *bipartite chain graphs*. Equivalently we say that (A, B) induces, or by a slight abuse of notation, *is* a half-graph if A and B can be totally ordered, say $a_1, \ldots, a_{|A|}$ and $b_1, \ldots, b_{|B|}$ such that $N_B(a_1) \supseteq N_B(a_2) \supseteq \ldots \supseteq N_B(a_t)$ and if $a_i b_j$ is an edge then for every $j' \in [j+1, t]$, $a_i b_{j'}$ is also an edge. The orderings $a_1, \ldots, a_{|A|}$ and $b_1, \ldots, b_{|B|}$ are called *orders* of the half-graph.

The half-graph of height t is a bipartite graph with partition $(A = \{a_1, \ldots, a_t\}, B = \{b_1, \ldots, b_t\})$ such that there is an edge between a_i and b_j if and only if $i < j$. We denote this graph by $H_{t,t}$. The *level* of a vertex $v \in A$ ($v \in B$) in the half-graph of height t is its index in the ordering of A (or B). Note that this is not uniquely defined for a half-graph in general, but it is for *the* (canonical) half-graph of height t. Any half-graph can be obtained from the half-graph of height t (for some t) by duplicating some vertices. The name *half-graph* actually comes from Erdős and Hajnal (see for instance [14]). More precisely what Erdős defines as a half-graph corresponds in this paper to the (canonical) half-graph of height t.

A *length-ℓ path of half-graphs* is a graph H whose vertex-set can be partitioned into $(H_1, H_2, \ldots, H_{\ell+1})$ such that the three following conditions hold:

- **(i)** there is no edge between H_i and H_j when $|i - j| \geqslant 2$,
- **(ii)** for every $i \in [\ell]$, $H[H_i \cup H_{i+1}]$ is a half-graph with bipartition (H_i, H_{i+1}), and
- **(iii)** for every $i \in [2, \ell]$, the ordering of H_i in the half-graph induced by (H_{i-1}, H_i) is the same as in the half-graph (H_i, H_{i+1}).

2.2 Grundy coloring

We say that an induced subgraph H of G is a *witness achieving* (color) k if H has a Grundy coloring of order at least k; in this case, we simply say that H is a *k-witness* (also called *atom* by Zaker [36] or *critical* [19]). We say that a k-witness is *minimal* if there is no proper induced subgraph of it whose Grundy number is at least k. A graph G has Grundy number at least k if and only if it contains a minimal k-witness as an induced subgraph [36].

Let $V_1 \uplus \cdots \uplus V_k$ be a Grundy coloring of order k. We say that a vertex u colored c' *supports* v colored c if u and v are adjacent and $c' < c$. A vertex v colored in c is said to be *supported* if the colors of the vertices supporting v span all colors from 1 to $c - 1$.

It was observed that the largest minimal k-witness uses 2^{k-1} vertices [36]. These witnesses are implemented by a family of rooted trees called *binomial trees* (see for instance [4]). The set of binomial trees $(T_k)_{k\geqslant 1}$ is defined recursively as follows:

- T_1 consists of a single vertex, declared as the root of T_1.
- T_k consists of two binomial trees T_{k-1} such that the root of the first one is a child of the root of the other. The root of the latter is declared as the root of T_k.

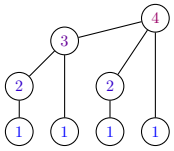


Figure 1 The binomial tree T_4, where the labels denote the color of each vertex in a first-fit coloring achieving the highest possible color.

We outline some basic properties of k-witnesses and binomial trees T_k.

▶ **Observation 1.** *Any subset of k' color classes of a k-witness, with $k' < k$, induces a k'-witness.*

The following is shown in a more general form in Lemma 7 of [4].

▶ **Lemma 2 (♣).** *Let $i \in [2, k-2]$, $X \subseteq V(T_k)$ be a subset of roots of T_i whose parent is a root of T_{i+1}, and T'_k be a tree obtained from T_k by removing the subtree T_{i-1} of every vertex in X. We assume that T'_k is an induced subgraph of a graph G and that $N(V(G) \setminus V(T'_k)) = X$. Then the three following conditions are equivalent in G:*

- **(i)** *There is a Grundy coloring that colors k the root of T'_k.*
- **(ii)** *There is a Grundy coloring that colors i every vertex of X without coloring their parent in T'_k first.*
- **(iii)** *There is a Grundy coloring that colors $i-1$ at least one neighbor of each vertex of X without coloring any vertex of T'_k first.*

Proof. (iii) implies (ii), and (ii) implies (i) are a direct consequence of the optimum Grundy coloring of a binomial tree, as depicted in Figure 1. We show that (i) implies (ii). This is equivalent to showing that the only way for a Grundy coloring of T_k to color its root k, even when there is joker that enables us to give any color to a vertex of X, is to respect the coloring of Figure 1. This holds since coloring a vertex of X with a color greater than i prevents from coloring its parent w with color $i+1$. Indeed in that case w cannot find a neighbor colored i (which is not its own parent). Coloring a vertex of X with a color smaller than i, simply will not work, since the Grundy coloring of T_k that gives color k to its root is unique. Finally (ii) implies (iii), since for every vertex of X, its only neighbor that can obtain color $i-1$ and is not its parent is outside T'_k. For a complete proof, see Lemma 7 of [4]. ◀

▶ **Lemma 3 (♠).** *If u and v are false twins in G, i.e., $N_G(u) = N_G(v)$, then $\Gamma(G) = \Gamma(G - \{v\})$.*

▶ **Lemma 4 (♠).** *Let H be an induced subgraph of G such that all the vertices of $N(V(H))$ have degree at most s. Then no vertex of $V(H)$ can get a color higher than $\Gamma(H) + s$ in a Grundy coloring of G.*

▶ **Corollary 5.** *In any greedy coloring, a vertex with at most t neighbors that have degree at most s cannot receive a color higher than $s + t + 1$.*

2.3 Partial Grundy and b-Chromatic Core

It is easy to see that admitting a partial Grundy coloring of order k is monotone under taking an induced subgraph.

▶ **Observation 6.** *A graph G admits a partial Grundy coloring of order at least k if and only if there exists a vertex-set $S \subseteq V(G)$ such that $G[S]$ admits a partial Grundy coloring of order k.*

Following from the observation, we can formally define Partial Grundy Coloring as:

Partial Grundy Coloring	**Parameter:** k
Input: An integer $k > 0$, a graph G.	
Question: Is there a vertex-subset $S \subseteq V(G)$ such that $G[S]$ admits a partial Grundy coloring of order k?	

On the other hand, b-coloring is not monotone under taking induced subgraphs. This leads us to the following monotone problem, which is distinct from deciding whether the b-chromatic number of G is at least k.

b-Chromatic Core	**Parameter:** k
Input: An integer $k > 0$, a graph G.	
Question: Is there a vertex-subset $S \subseteq V(G)$ such that $G[S]$ admits a b-coloring of order k?	

For both Partial Grundy Coloring and b-Chromatic Core, the subgraph of G induced by S is referred to as a *k-witness* if $S \subseteq V(G)$ is a solution to the instance (G, k). A k-witness H is called a *minimal k-witness* if $H - v$ is not a k-witness for every $v \in V(H)$.

Let $V_1 \uplus \cdots \uplus V_k$ be a proper coloring of G. In the context of partial Grundy coloring (b-coloring, respectively), we say that a vertex v colored c is *supported by* u if $uv \in E(G)$ and u is colored $c' < c$ ($c' \neq c$, respectively). In the partial Grundy coloring (b-coloring, respectively), a vertex v colored c is *supported* if the colors of the supporting vertices of v span all colors from 1 to $c-1$ (all colors of $[k] \setminus c$, respectively). Such a vertex v is also called a *center*. A color c is said *realized* if a vertex v colored c is supported. That vertex v is then *realizing* color c. Notice the crucial difference with Grundy colorings that these $c-1$ vertices do not need to be supported themselves.

As each center requests at most $k-1$ supporting vertices, a minimal k-witnesses of Partial Grundy Coloring or b-Chromatic Core has size bounded by k^2 [11]. We denote by $\Gamma'(G)$, respectively $\Gamma_b(G)$, the maximum integer k such that G admits a k-witness for Partial Grundy Coloring, respectively b-Chromatic Core.

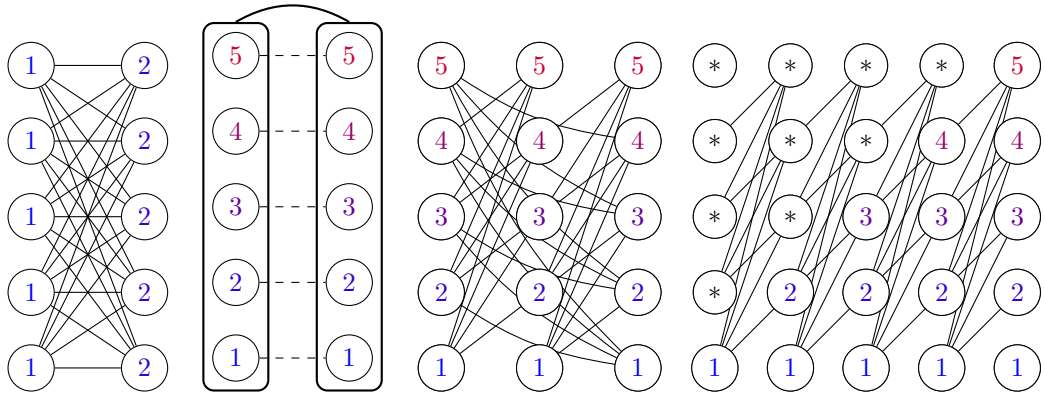
3 Barriers to the Parameterized Hardness of Grundy Coloring

It is not difficult to see that deciding if a fixed vertex can get color k in a greedy coloring is W[1]-hard. Let us call this problem Rooted Grundy Coloring.

▶ **Observation 7.** Rooted Grundy Coloring *is W[1]-hard.*

Proof. We design an FPT reduction from k-Multicolored Independent Set to Rooted Grundy Coloring. Let H be an instance of k-MIS with partition $V_1, \ldots, V_k$. We build an equivalent instance G of Rooted Grundy Coloring in the following way. We copy H in G and we add a clique C of size $k+1$. We call v a fixed vertex of C and we add a pendant neighbor v' to v. We number the vertices of $C \setminus \{v\}$, $v_1, \ldots, v_k$, and we make v_i adjacent to all the vertices of V_i for each $i \in [k]$. A greedy coloring can color v by $k+2$ if and only if there is a k-multicolored independent set in H. ◀

Of course this reduction does not imply anything for Grundy Coloring. Indeed the vertices of $V(H)$ could get much higher colors than v. This is precisely the issue with showing the parameterized hardness of Grundy Coloring.



(a) Biclique. **(b)** Anti-matching. **(c)** Half-graph short cycle. **(d)** Half-graph long path.

Figure 2 The barriers in a propagation gadget for Grundy Coloring. The biclique has small Grundy number but do not propagate, nor it imposes a unique choice. The three other propagations have arbitrary large Grundy number, rendering them useless for a parameterized reduction.

A reduction starting from any W[1]-hard problem has to "erase" the potentially large Grundy number of the initial structure. This can be done by isolating it with low-degree vertices. However the degree Δ of the graph should be large, and a large chunk of the instance should have degree unbounded in k since Grundy Coloring is FPT parameterized by $\Delta + k$ [31, 4]. Besides, as it is the case with W[1]-hardness reductions where induced subgraphs of the initial instance have to be tamed, we crucially need to propagate consistently one choice among a number of alternatives unbounded in the parameter.

A natural idea for encoding one choice among $t \ggg k$ is to have a set S of t vertices, one of which, the *selected* vertex, receiving a specific color, say, 1. Then a mechanism should ensure that one cannot color 1 two or more vertices of S. Note that we cannot force that property by cliquifying S, as this would elevate the Grundy number to at least t. Furthermore, by Ramsey's theorem, there will be independent sets of size $2^{\Omega(\frac{\log t}{k})}$ in S. Thus we might as well assume that S is an independent set, and look for another way of preventing two vertices from getting color 1, than by adding edges inside S.

We are now facing the following task: Given a bipartite graph, or a "path" or "cycle" of bipartite graphs whose partite sets are copies of S, ensure that exactly one vertex can receive color 1 in each partite set, and that this corresponds to a single vertex in S. A biclique certainly has low Grundy number (see Figure 2a) but does not propagate nor it actually forces a unique choice. Anything more elaborate seems to have large Grundy number, be it the complement of an induced matching, or *anti-matching*, (see Figure 2b), a "cycle" of half-graphs (see Figure 2c), or even a long "path" of half-graphs (see Figure 2d). We remind the reader that, as detailed in Section 2, half-graphs and anti-matchings are (the) two ways of propagating a consistent independent set.

4 Overcoming the Barriers: Short Path of Half-Graphs

It might be guessed from the previous section that the solution will come from a constant-length "path" of half-graphs. It is easy to see that half-graphs (that can be seen as length-one path of half-graphs) have Grundy number at most 3. Due to the $2K_2$-freeness of the

half-graph, there cannot be both color 1 and color 2 vertices present on both sides of the bipartition, say (A, B). If A is the side missing a 1 or a 2 among its colors, then B in turn cannot have a 3 (nor a 4). The absence of vertices colored 3 in B prevents vertices colored 4 in A. Overall, no vertex with color 4 can exist. It takes more time to realize that a length-two path of half-graphs have constant Grundy number. We crucially use the fact that any constant-length path of half-graphs have constant Grundy number.

▶ **Lemma 8.** *The Grundy number of a length-ℓ path of half-graphs is at most 4^ℓ.*

Proof. Achieving a (more) reasonable upper bound –the Grundy number of such graphs is most likely polynomial or even linear in ℓ– proves to be not so easy. We choose here to give a short proof of an admittingly bad upper bound.

We show this bound by induction on ℓ. Note that the statement trivially holds for $\ell = 0$, and that we previously verified it for $\ell = 1$. Assume that the Grundy number of any length-$(\ell - 1)$ path of half-graphs is at most $4^{\ell-1}$, for any $\ell \geqslant 2$.

Let G be a length-ℓ path of half-graphs, with partition $V(G) = V_0 \uplus V_1 \uplus \cdots \uplus V_\ell$ where $G[V_i \cup V_{i+1}]$ is a half-graph for each $i \in [\ell - 1]$. Observe that $G - V_0$ and $G - V_\ell$ are both length-$(\ell - 1)$ path of half-graphs. Let H be a colored witness of G achieving color $\Gamma(G)$. We distinguish some cases based on the number of colors of H appearing in V_0 or in V_ℓ. In each case, we conclude with Observation 1. No more than $4^{\ell-1}$ colors of H can be missing in V_0 (resp. in V_ℓ). Otherwise by Observation 1, the corresponding color classes form a k-witness $G - V_0$ (resp. in $G - V_\ell$) with some $k > 4^{\ell-1}$, contradicting the induction hypothesis.

So we may assume that at least $\Gamma(G) - 4^{\ell-1}$ colors appear in V_0 (resp. in V_ℓ). Thus at least $(2\Gamma(G) - 2 \cdot 4^{\ell-1}) - \Gamma(G) = \Gamma(G) - 2 \cdot 4^{\ell-1}$ colors appears in both V_0 and V_ℓ. If $\Gamma(G) > 4^\ell$, then $\Gamma(G) - 2 \cdot 4^{\ell-1} > 4^{\ell-1}$. We further claim that the corresponding color classes would form a witness in $G - V_0$, a contradiction. If not, it must be because a vertex $x \in V_1$ colored i was adjacent to a vertex $y \in V_0$ colored $j < i$, and is not adjacent to any vertex colored j in $G - V_0$. But we know that V_0 contains a vertex y' colored i, which in turn must be adjacent to a vertex $x' \in V_1$ colored j, forming an induced $2K_2$ in $G[V_0 \cup V_1]$, a contradiction. Therefore, $\Gamma(G) \leqslant 4^\ell$.

We observe that our proof works for a more general notion of "path of half-graphs" where one does not impose the orders of the successive half-graphs to have the same orientation (see the second paragraph of Section 2.1). ◀

We are now ready to present the hardness construction. We reduce from k-Multicolored Subgraph Isomorphism whose definition is the following.

k-Multicolored Subgraph Isomorphism **Parameter:** k
Input: An integer $k > 0$, a graph G whose vertex-set is partitioned into k sets $V_1, \ldots, V_k$, and a graph H with $V(H) = [k]$.
Question: Is there $\phi : i \in [k] \mapsto v_i \in V_i$ such that for all $ij \in E(H)$, $\phi(i)\phi(j) \in E(G)$?

▶ **Theorem 9.** Grundy Coloring *is W[1]-complete and, unless the ETH fails, cannot be solved in time $f(q)n^{o(2^{q-\log q})}$ (nor in time $f(q)n^{2^{o(q)}}$) for any computable function f, on n-vertex graphs with Grundy number q.*

Proof. The membership to W[1] is given by the framework of Cesati [7], since there is always a witness of size 2^{q-1}. We show the W[1]-hardness of Grundy Coloring by reducing from k-Multicolored Subgraph Isomorphism with 3-regular pattern graphs. Let $(G = (V_1, \ldots, V_k, E), H = ([k], F))$ be an instance of that problem. We further assume that k is a positive even integer and there is no edge between V_i and V_j in G whenever

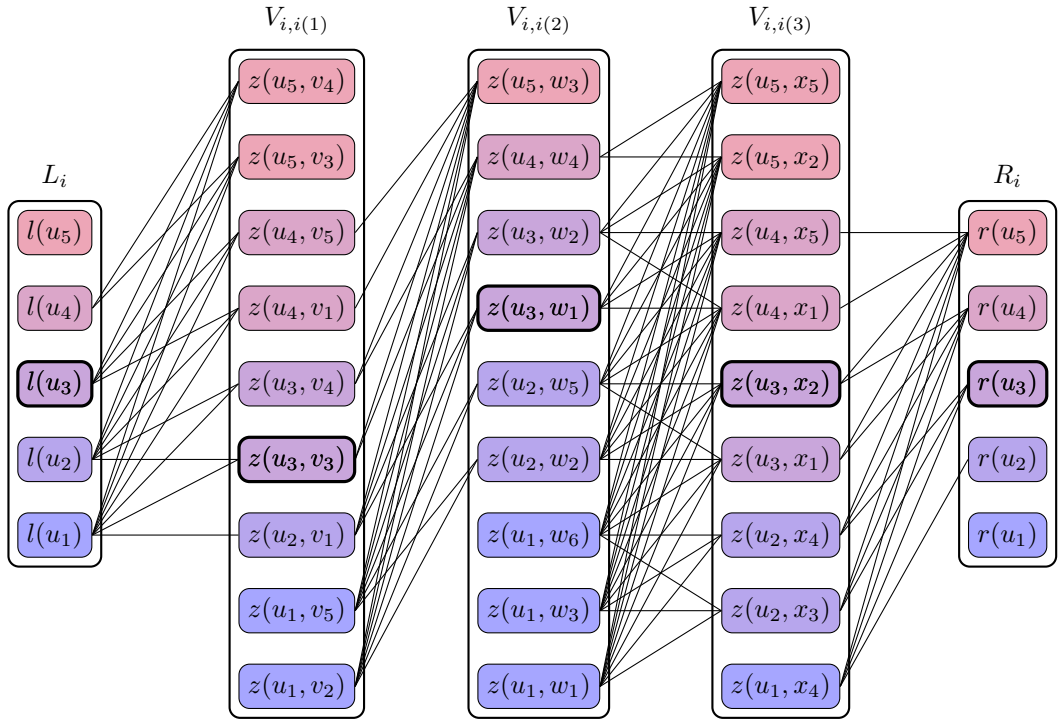


Figure 3 The encoding H_i of one V_i ordered $u_1 < u_2 < u_3 < u_4 < u_5$. In bold, a possible independent set intersecting the five sets and containing a consistent pair $l(u), r(u)$.

$ij \notin E(H)$. The goal is now to find $v_1 \in V_1, \ldots, v_k \in V_k$ such that H is isomorphic to $G[\{v_1, \ldots, v_k\}]$. Even with these restrictions k-Multicolored Subgraph Isomorphism cannot be solved in time $f(k)|V(G)|^{o(k/\log k)} = f(|E(H)|)|V(G)|^{o(|E(H)|/\log|E(H)|)}$, unless the ETH fails (see [26, 27], and Theorem 5.5 in [28]).

We build an equivalent Grundy Coloring-instance (G', q) with $q = \lceil \log k \rceil + 258$ as follows. For each color class V_i, we fix an arbitrary total ordering $\leqslant_i$ on the vertices of V_i, and we write $u <_i u'$ if $u \neq u'$ and $u \leqslant_i u'$. Let $i \in [k]$ and let $i(1), i(2), i(3) \in [k]$ be the three neighbors of i in H. Each V_i is encoded by a length-4 path of half-graphs denoted by H_i (see Figure 3). We now detail the construction of H_i.

We set $V(H_i) := L_i \cup V_{i,i(1)} \cup V_{i,i(2)} \cup V_{i,i(3)} \cup R_i$. The vertices of L_i (resp. R_i) are in one-to-one correspondence with the vertices of V_i. We denote by $l(u)$ (resp. $r(u)$) the vertex of L_i (resp. R_i) corresponding to $u \in V_i$. For each $p \in [3]$, the vertices of $V_{i,i(p)}$ are in one-to-one correspondence with the edges of $E(V_i, V_{i(p)})$. We denote by $z(u, v)$ the vertex of $V_{i,i(p)}$ corresponding to the edge $uv \in E(V_i, V_{i(p)})$ with $u \in V_i$ and $v \in V_{i(p)}$.

We set $E(H_i) := E(L_i, V_{i,i(1)}) \cup E(V_{i,i(1)}, V_{i,i(2)}) \cup E(V_{i,i(2)}, V_{i,i(3)}) \cup E(V_{i,i(3)}, R_i)$:

- $l(u)z(u', v) \in E(L_i, V_{i,i(1)})$ if and only if $u <_i u'$
- for $p \in [2]$, $z(u, v)z(u', v') \in E(V_{i,i(p)}, V_{i,i(p+1)})$ if and only if $u <_i u'$
- $z(u, v)r(u') \in E(V_{i,i(3)}, R_i)$ if and only if $u <_i u'$.

For each pair of vertices $u, u' \in V_i$ such that $u <_i u'$, we add an edge between $l(u)$ and $z(u', v) \in V_{i,i(1)}$, respectively $z(u, v) \in V_{i,i(1)}$ and $z(u', v') \in V_{i,i(2)}$, respectively $z(u, v) \in V_{i,i(2)}$ and $z(u', v') \in V_{i,i(3)}$, respectively $z(u, v) \in V_{i,i(3)}$ and $r(u')$ (see Figure 3).

For each $ij \in E(H)$, we create $|E(V_i, V_j)|$ copies of the binomial tree T_5. So these trees are in one-to-one correspondence with the edges of G between V_i and V_j, and we denote by $T_5(uv)$ the tree corresponding to $uv \in E(V_i, V_j)$. We denote by $\beta(uv)$ and $\gamma(uv)$ the

two children getting color 2 of the only two vertices colored 3, in the Grundy coloring of $T_5(uv)$ which gives color 5 to its root. We remove the pendant neighbor of $\beta(uv)$ and of $\gamma(uv)$ (the two vertices getting color 1 and supporting $\beta(uv)$ and $\gamma(uv)$). This results in a fourteen-vertex tree. We denote this set of trees by $\mathcal{T}_{i,j}$, and the $|E(V_i, V_j)|$ roots of the T_5 by $\mathcal{R}_{i,j}$. For each $ij \in E(H)$ and for every pair $z(u,v) \in V_{i,j}$, $z(v,u) \in V_{j,i}$, we make $z(u,v)$ and $\beta(uv)$ adjacent, and we make $z(v,u)$ and $\gamma(uv)$ adjacent.

For every $i \in [k]$, we create $|V_i|$ copies of the binomial tree T_5. These trees are in one-to-one correspondence with V_i. Similarly as above, we denote by $\beta(u)$ and $\gamma(u)$ the two vertices getting color 2, whose parents are colored 3, in $T_5(u)$ and we remove their pendant neighbor (colored 1). For every pair $l(u) \in L_i$ and $r(u) \in R_i$, we link $l(u)$ and $\beta(u)$, and we link $r(u)$ and $\gamma(u)$. We denote this set of trees by $\mathcal{T}_i$, and the $|V_i|$ roots of the T_5 by $\mathcal{R}_i$.

We finally create one copy of the binomial tree T_q. We observe that there are $|E(H)|$ sets $\mathcal{R}_{i,j}$ and $|V(H)|$ sets $\mathcal{R}_i$. The binomial tree T_q has at least $|V(H)| + |E(H)| = 2.5k$ vertices getting color 7 in the greedy coloring giving color q to the root. Indeed the number of vertices colored 7 is 2^{q-8}, and it holds that $q - 8 \geqslant \log k + \log 2.5$. We map $2.5k$ distinct vertices colored 6 in T_q, that are children of vertices colored 7, in a one-to-one correspondence with $V(H) \cup E(H)$. Let $f(i)$ be the vertex mapped to $i \in V(H)$ and $f(ij)$ be the vertex mapped to $ij \in E(H)$. We further remove the subtree T_5 of each of these $2.5k$ vertices colored 6. For every $i \in V(H)$, we link $f(i)$ to all the vertices in $\mathcal{R}_i$. Similarly for every $ij \in E(H)$, we link $f(ij)$ to all the vertices in $\mathcal{R}_{i,j}$. This finishes the construction of the graph G'. Solving Grundy Coloring in time $f(q)n^{o(2^{q-\log q})} = f(\lceil \log k \rceil + 258)n^{o(k/\log k)}$ would give the same running time for k-Multicolored Subgraph Isomorphism, which is ruled out under the ETH. We now prove that the reduction is correct.

A solution to k-Multicolored Subgraph Isomorphism implies $\Gamma(G') \geqslant q$

Let $v_1 \in V_1, v_2 \in V_2, \ldots, v_k \in V_k$ be a fixed solution to the k-Multicolored Subgraph Isomorphism-instance (the colored isomorphism being $i \in [k] \mapsto v_i$). We say that each edge v_iv_j is *in the solution* (for $i \neq j \in [k]$). We color 1 all the vertices of G' corresponding to edges in the solution, that is, all the vertices $z(v_i, v_j)$, as well as all vertices of G' corresponding to vertices in the solution, that is $l(v_i)$ and $r(v_i)$. This is possible since the five vertices $l(v_i), z(v_i, v_{i(1)}), z(v_i, v_{i(2)}), z(v_i, v_{i(3)}), r(v_i)$ form an independent set since $\neg(v_i <_i v_i)$.

We can now color 2 the vertices $\beta(v_i)$ and $\gamma(v_i)$. Therefore the root of $T_5(v_i)$ can receive color 5. Moreover, for every $ij \in E(H)$ we can color 2 the vertices $\beta(v_iv_j)$ and $\gamma(v_iv_j)$. Therefore the root of $T_5(v_iv_j)$ can receive color 5. Since one vertex in each $\mathcal{R}_i$, and one vertex in each $\mathcal{R}_{i,j}$ get color 5, the vertices $f(i)$ and $f(ij)$ can all get color 6. Finally the root of T_q can receive color q.

$\Gamma(G') \geqslant q$ implies a solution to k-Multicolored Subgraph Isomorphism

We first show that only the two vertices of T_q with degree $q-1$ can get color q. Besides these two vertices, the only vertices of T_q with sufficiently large degree to get color q are the vertices $f(i)$ and $f(ij)$. But these vertices have at most one neighbor of degree more than 5. So according to Corollary 5, they cannot receive a color higher than $7 < q$. Now we use Lemma 8 to bound the color reachable outside of T_q. For every $i \in [k]$, the induced subgraph $G'[H_i]$ is a length-four path of half-graphs. Thus by Lemmas 3 and 8, $\Gamma(G'[H_i]) \leqslant 4^4 = 256$. All the vertices in the open neighborhood of $V_{i,i(1)} \cup V_{i,i(2)} \cup V_{i,i(3)}$ have degree at most 2. So by Lemma 4 vertices outside T_q cannot receive a color beyond $258 < q$.

We now established that if $\Gamma(G') \geqslant q$ (actually $\Gamma(G') = q$), then either one of the two possible roots of T_q shall receive color q. By Lemma 2, this implies that all the vertices $f(i)$ and $f(ij)$ receive color 6, and that in each $\mathcal{R}_i$ and each $\mathcal{R}_{i,j}$ there is at least one vertex receiving color 5. For every $i \in [k]$, let $T_5(u_i)$ be one T_5 of $\mathcal{T}_i$ whose root gets color 5. We will now show that $\{u_1, \ldots, u_i, \ldots, u_k\}$ is a solution to the k-Multicolored Subgraph Isomorphism-instance. Again by Lemma 2, this is only possible if $\beta(u_i)$ and $\gamma(u_i)$ both get color 2, and their unique neighbor outside $T_5(u_i)$ gets color 1. It means that $l(u_i)$ and $r(u_i)$ both get color 1.

Since every $\mathcal{R}_{i,j}$ contains at least one vertex colored 5, Lemma 2 implies that every $V_{i,i(p)}$ (for each $p \in [3]$) gets at least one vertex colored 1. Let $z(u,v) \in V_{i,i(1)}$, $z(u',v') \in V_{i,i(2)}$, and $z(u'',v'') \in V_{i,i(3)}$ three vertices getting color 1. As $\{l(u_i), z(u,v), z(u',v'), z(u'',v''), r(u_i)\}$ should be an independent set, we have $u_i \geqslant_i u \geqslant_i u' \geqslant_i u'' \geqslant_i u_i$. This implies that $u_i = u = u' = u''$. In turn that implies that no vertex $z(u^*, v) \in V_{i,i(1)} \cup V_{i,i(2)} \cup V_{i,i(3)}$ with $u^* \neq u_i$ can get color 1. Indeed $l(u_i)$ prevents a 1 "above" $z(u_i, v) \in V_{i,i(1)}$ and $z(u_i, v') \in V_{i,i(2)}$ prevents a 1 "below" $z(u_i, v)$. The same goes for the color classes $V_{i,i(2)}$ and $V_{i,i(3)}$. Thus the only trees $T_5(uv) \in \mathcal{T}_{i,j}$ that can get color 5 at their root are the ones such that $\{u, v\} \subset \{u_1, \ldots, u_k\}$. As all the $1.5k$ sets $\mathcal{T}_{i,j}$ have such a tree, it implies that $\{u_1, \ldots, u_k\}$ is a solution to the k-Multicolored Subgraph Isomorphism-instance. ◀

5 Parameterized hardness of b-Chromatic Core

A length-two path of half-graphs have arbitrary large b-chromatic core. Nevertheless a simple half-graph only admits b-chromatic cores of bounded size. We show how to still build a W[1]-hardness construction in this furtherly constrained situation.

▶ **Theorem 10 (♣).** *b-Chromatic Core is W[1]-complete.*

Proof. The inclusion in W[1] is immediate by the characterization of Cesati [7], and the facts that minimal witnesses have size at most k^2, and that given the subgraph induced by a minimal witness one can check if it is solution. To show W[1]-hardness, we reduce from k-by-k Grid Tiling. In this problem, given k^2 sets of pairs over $[n]$, say, $(P_{i,j} \subseteq [n] \times [n])_{i,j \in [k] \times [k]}$, "displayed in a k-by-k grid", one has to find one pair $(x_{i,j}, y_{i,j})$ in each $P_{i,j}$ such that $x_{i,j} = x_{i,j+1}$ and $y_{i,j} = y_{i+1,j}$, for every $i, j \in [k-1]$. This problem was introduced and shown W[1]-hard by Marx [24]. It is called Matrix Tiling in [25], although subsequent papers refer to it as Grid Tiling. We assume that k is dividable by 3, $k^2 > 33$, and for the sake of clarity, that each $P_{i,j}$ contains the same number of pairs, say $t \leqslant n^2$. This problem remains W[1]-hard under these assumptions.

Construction

Let $(P_{i,j} \subseteq [n] \times [n])_{i,j \in [k] \times [k]}$ be the instance of Grid Tiling. For each (i,j), we have the set of pairs $P_{i,j}$ with $|P_{i,j}| = t$. For each (i,j), we add a biclique $K_{t,q-9}(i,j) := K_{t,q-9}$, where $q := 14k^2$. The part of $K_{t,q-9}(i,j)$ with size t is denoted by $A_{i,j}$ and the other part by $B_{i,j}$ (see Figure 4). We denote by $A_{i,j}$ the t vertices to the left of $K_{t,q-9}(i,j)$ on Figure 4, and by $B_{i,j}$, the $q-9$ vertices to the right. The vertices of $A_{i,j}$ are in one-to-one correspondence with the pairs of $P_{i,j}$. We denote by $a_{i,j}(x,y) \in A_{i,j}$ the vertex corresponding to $(x,y) \in P_{i,j}$. We make the construction "cyclic", or rather "toroidal". So in what follows, every occurrence of $i+1$ or $j+1$ should be interpreted as 1 in case $i = k$ or $j = k$.

For every vertically (resp. horizontally) consecutive pairs (i,j) and $(i+1,j)$ (resp. (i,j) and $(i,j+1)$) we add a half-graph $H(i \to i+1, j)$ (resp. $H(i, j \to j+1)$) with bipartition $H(\underline{i} \to i+1, j) \cup H(i \to \underline{i+1}, j)$ (resp. $H(i, \underline{j} \to j+1) \cup H(i, j \to \underline{j+1})$). Both sets

$H(\underline{i} \to i+1, j)$ and $H(i, \underline{j} \to j+1)$ are in one-to-one correspondence with the vertices of $A_{i,j}$, while the set $H(i \to \underline{i+1}, j)$ is in one-to-one correspondence with the vertices of $A_{i+1,j}$, and $H(i, j \to \underline{j+1})$, with the vertices of $A_{i,j+1}$. We denote by $h_{\underline{i} \to i+1,j}(x, y)$ (resp. $h_{i \to \underline{i+1},j}(x', y')$) the vertex corresponding to $a_{i,j}(x, y)$ (resp. $a_{i+1,j}(x', y')$). Similarly we denote by $h_{i,\underline{j} \to j+1}(x, y)$ (resp. $h_{i,j \to \underline{j+1}}(x', y')$) the vertex corresponding to $a_{i,j}(x, y)$ (resp. $a_{i,j+1}(x', y')$). Every vertex in a half-graph $H(i \to i+1, j)$ or $H(i, j \to j+1)$ is made adjacent to its corresponding vertex in $A_{i,j} \cup A_{i+1,j} \cup A_{i,j+1}$. Thus $a_{i,j}(x, y)$ is linked to $h_{\underline{i} \to i+1,j}(x, y)$, $h_{i-1 \to \underline{i},j}(x, y)$, $h_{i,\underline{j} \to j+1}(x, y)$, and $h_{i,j-1 \to \underline{j}}(x, y)$. Note that underlined numbers are used to distinguish names, and to give information on its neighborhood. We call *vertical* half-graph an $H(i \to i+1, j)$, and *horizontal* half-graph an $H(i, j \to j+1)$. We now precise the order of the half-graphs. In vertical half-graphs, we put an edge between $h_{\underline{i} \to i+1,j}(x, y)$ and $h_{i \to \underline{i+1},j}(x', y')$ whenever $y < y'$. In horizontal half-graphs, we put an edge between $h_{i,\underline{j} \to j+1}(x, y)$ and $h_{i,j \to \underline{j+1}}(x', y')$ whenever $x < x'$.

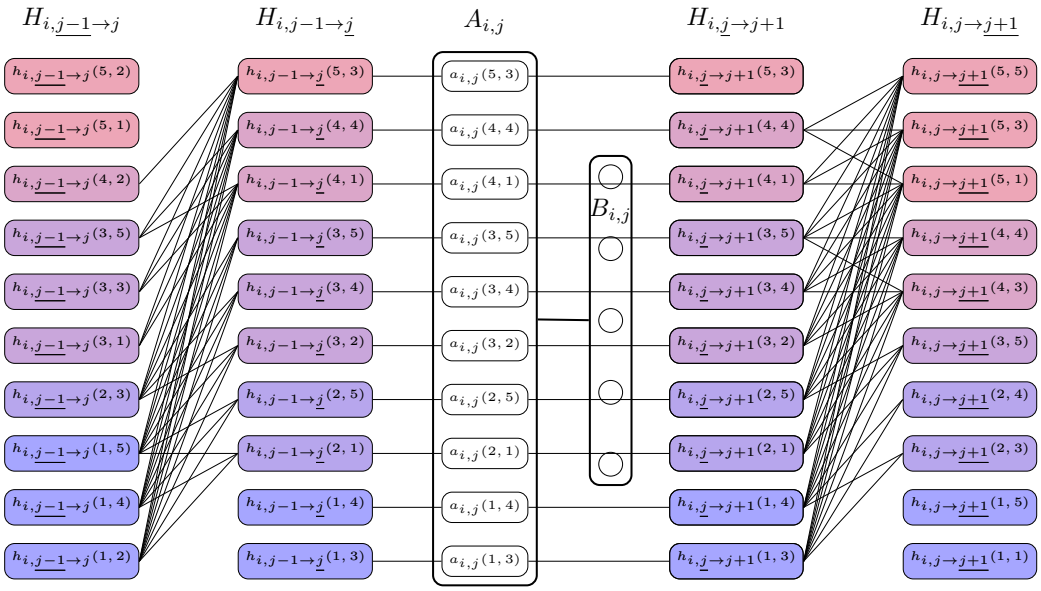


Figure 4 The biclique $K_{t,q-9}(i, j)$ encoding the pairs $P_{i,j}$, and its connection to the two neighboring horizontal half-graphs, with $n = 5$, $t = 10$, and $q = 14$.

We then add a global clique C of size $q - k^2$. We attach k^2 private neighbors to each vertex of C. Among the $q - k^2$ vertices of C, we arbitrarily distinguish 33 vertices: a set $D = \{d_1, \ldots, d_{18}\}$ of size 18, and three sets C', C^-, C^+ each of size 5. We fully link d_z to every $B_{i,j}$ if z takes one of the following values:

- $3(j \bmod 3 - 1) + i \bmod 3$,
- $\text{succ}(3(j \bmod 3 - 1) + i \bmod 3)$,
- $3(i \bmod 3 - 1) + j \bmod 3 + 9$,
- $\text{succ}(3(i \bmod 3 - 1) + j \bmod 3 + 9)$,

where the modulos are always taken in $\{0, 1, 2\}$, and $\text{succ}(x) := x + 1$ if x is not dividable by 3 and $\text{succ}(x) := x - 2$ otherwise (see Figure 5). Note that each $B_{i,j}$ is linked with d_z for two successive (indicated by the operator $\text{succ}(x)$) integers z in the range of $[1, 3]$, $[4, 6]$ or $[7, 9]$ depending on the coordinate j modulo 3. Likewise, each $B_{i,j}$ is linked with d_z for two successive integers z in the range of $[10, 12]$, $[13, 15]$ or $[17, 18]$ depending on the coordinate i modulo 3.

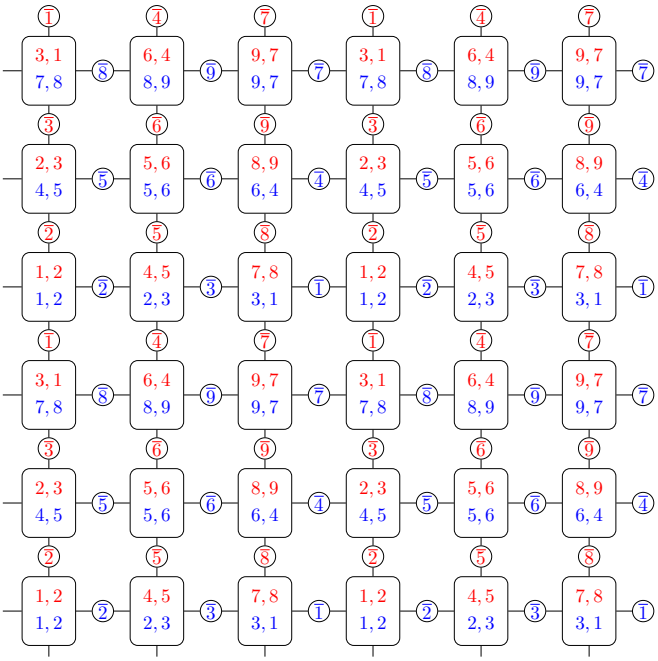


Figure 5 The rounded rectangles represent the $B_{i,j}$, and the numbers therein, the $z \in [18]$ such that d_z is fully linked to it. These are the "colors" that a center in $A_{i,j}$ will have to fight for. The circles represent the half-graphs, and the number therein, the only z such that d_z is *not* linked to it. Red integers are offset by 9 ($1 = 10$, $2 = 11$, and so on). The edges represent the non-empty interaction between the $A_{i,j}$ and the half-graphs. The structure is glued like a torus.

We observe that $B_{i,j}$ and $B_{i+1,j}$ are linked to exactly one common d_z ($z \in [18]$), and we fully link the half-graph $H(i \to i+1, j)$ to $D \setminus \{z\}$. Similarly we fully link the half-graph $H(i, j \to j+1)$ to $D \setminus \{z\}$ where z is the unique integer of [18] such that d_z is fully linked to both $B_{i,j}$ and $B_{i,j+1}$. We fully link each $A_{i,j}$ to C', each $H(\underline{i} \to i+1, j)$ and $H(i, \underline{j} \to j+1)$ to C^-, and each $H(i \to \underline{i+1}, j)$ and $H(i, j \to \underline{j+1})$ to C^+. This is just to prevent that one uses a vertex of a $B_{i,j}$ or of a half-graph as a center. As we will see, intended solutions have all their centers in $C \cup \bigcup_{i,j \in [k]} A_{i,j}$.

This ends our polytime construction. We denote by G the obtained graph. In the long version [2], we show that G has b-chromatic number at least q if and only if the k-BY-k GRID TILING-instance is positive. ◀

6 Partial Grundy Coloring and b-Chromatic Core on $K_{t,t}$-free graphs

In the following subsection, we prove that both b-CHROMATIC CORE and PARTIAL GRUNDY COLORING can be solved in FPT time when all but a bounded number of vertices have bounded degree. This is a preparatory step to show the tractability in $K_{t,t}$-free graphs.

6.1 FPT algorithm on almost bounded-degree graphs

The technique of random separation [6, 5], inspired by the color coding technique [3], comes handy when one wants to separate a latent vertex-subset of small size from the rest of the graph. A derandomize version of random separation can be obtained with splitters by Naor et al. [29] (see also Chitnis et al. [8]) and is available in literature. For two disjoint sets A and B of a universe U, we say that $S \subseteq U$ is an (A, B)-*separating set* if $A \subseteq S$ and $B \cap S = \emptyset$.

▶ **Lemma 11** (Chitnis et al. [8]). *Let a and b be non-negative integers. For an n-element universe U, there exists a family $\mathcal{F}$ of $2^{O(\min(a,b)\log(a+b))}\log n$ subsets of U such that for any disjoint subsets $A, B \subseteq U$ with $|A| \leqslant a$ and $|B| \leqslant b$, there exists an (A,B)-separating set S in $\mathcal{F}$. Furthermore, such a family $\mathcal{F}$ can be constructed in time $2^{O(\min(a,b)\log(a+b))}n\log n$.*

▶ **Theorem 12.** *Let G be a graph in which at most s vertices have degree larger than d. Then whether G has a k-witness for b-Chromatic Core (Partial Grundy Coloring, respectively) can be decided in FPT time parameterized by $k+d+s$.*

Proof. Let X be the set of s vertices of degree larger than d. In order to explain the algorithm and prove its correctness, it is convenient to assume that G does contain a k-witness H for b-Chromatic Core (or Partial Grundy Coloring) as an induced subgraph. We define $I := V(H) \cap X, A := V(H) \setminus X$, and $B := N(A) \setminus X$.

We can guess I by considering at most 2^s subsets of X. To find A, we use Lemma 11. From the fact that every vertex of $V \setminus X$ has degree at most d and that H is a minimal k-witness, we have $|A| \leqslant k^2$ and $|B| \leqslant dk^2$. Hence, by Lemma 11 with universe $V(G) \setminus X$, we can compute in time $2^{O(k^2\log(k^2+dk^2))}n\log n$ a family $\mathcal{F}$ with $2^{O(k^2\log(k^2+dk^2))}\log n$ subsets of $V(G) \setminus X$, that contains an (A,B)-separating set.

We guess this (A,B)-separating set by iterating over all elements of $\mathcal{F}$. Let S be a correct guess, i.e., S is an (A,B)-separating set. So $A \subseteq S$ and $S \cap B = \emptyset$. Observe that every connected component of $G[A]$ appears in $G[S]$ as a connected component.

Let $\mathcal{C}_S$ be the set of connected components of $G[S]$ of size at most k^2. Since $|A| \leqslant k^2$, larger connected component of $G[S]$ are clearly disjoint from A. Moreover, by definition of B and since S is disjoint from B, each connected component of $G[A]$ is an element of $\mathcal{C}_S$.

Since each element of $\mathcal{C}_S$ has at most k^2 vertices, the number of equivalence classes of $\mathcal{C}_S$ under graph isomorphism is bounded by a function of k. In fact, the number of equivalence classes under a stronger form of isomorphism is bounded by a function of k. We define a labeling function $\ell : S \to 2^I$ as $\ell(v) := N(v) \cap I$. Let $\sim_S$ be a relation on $\mathcal{C}_S$ such that, for every $C, C' \in \mathcal{C}'_S$: $C \sim_S C'$ if and only if there is a graph isomorphism $\phi : C \to C'$ with $\ell(v) = \ell(\phi(v))$ for every $v \in C$. Let $[\sim_S]$ be the partition of $\mathcal{C}_S$ into equivalence classes under $\sim_S$. As members of $\mathcal{C}_S$ have cardinality at most k^2 and there are $2^{|I|} \leqslant 2^{k^2}$ labels, $\mathcal{C}_S$ has at most 2^{2k^4} equivalence classes under $\sim_S$. And thus we can compute $[\sim_S]$ in time $2^{2k^4}n$. The definition of $\sim_S$ clearly implies that two equivalent sets C and C' under $\sim_S$ are exchangeable as a connected component of $H - X$. That is, for any induced subgraph D of G with $V(D) \cap X = I$, if C is a connected component of $D - I$, then $G[(V(D) \setminus C) \cup C']$ is isomorphic to D.

We will now guess, by doing an exhaustive search, how many connected components $H - I$ takes from each part of the partition $[\sim_S]$. There are $2^{2k^4k^2}$ possible such guesses and from the fact that the number of connected components in $H - I$ is at most k^2. Choose an element (i.e. a connected vertex-set) from each part of $[\sim_S]$ as many times as the current guess suggests (if this is impossible, then discard the current guess) and let W be the union of the chosen connected vertex-sets. We can now verify by brute-force that $G[W \cup I]$ is a k-witness for b-Chromatic Core or Partial Grundy Coloring, depending on the problem at hand.

To complete the proof of correctness, note that if we find a k-witness for some choice of I, $S \in \mathcal{F}$ and W, the input graph G clearly admits a k-witness. One can easily observe that the running time is FPT in $k+d+s$. ◀

6.2 FPT algorithm on $K_{t,t}$-free graphs

In this subsection, we present an FPT algorithm on graphs which do not contain $K_{t,t}$ as a subgraph. A key element of this algorithm is a combinatorial result (Proposition 16), which states that if there are many vertices of large degree, then one can always find a k-witness.

▶ **Lemma 13 (♠).** *Let t and N be two positive integers with $N \geqslant t$, and let G be a graph on a vertex-set $A \uplus B$ not containing $K_{t,t}$ as a subgraph. If $|A| \geqslant N2^{N+t}$ and $|B| \geqslant N+t$, then there exist two sets $A' \subseteq A$ and $B' \subseteq B$, each of size at least N, such that there is no edge between A' and B'.*

▶ **Lemma 14 (♠).** *For any integers k and t, there exists an integer M such that the following holds: given a $K_{t,t}$-free graph G and a partition $A_1 \uplus \cdots \uplus A_k$ of $V(G)$ such that each A_i contains at least M vertices, there exists either a clique on k vertices, or an independent set of size k^2 which contains k vertices from each A_i.*

The following statement is proved in [1].

▶ **Lemma 15** (Aboulker et al. [1])**.** *Let t be a positive integer and let $\epsilon \in (0,1)$. Then there is an integer $N(t,\epsilon)$ that satisfies the following: if $H = (V,E)$ is a hypergraph on at least $N(t,\epsilon)$ vertices, where all hyperedges have size at least $\epsilon|V|$, and the intersection of any t hyperedges has size at most $t-1$, then $|E| < t/\epsilon^t$.*

We are ready to prove the key combinatorial result on $K_{t,t}$-free graphs.

▶ **Proposition 16.** *Let t, k be positive integers. Let G be a $K_{t,t}$-free graph and let $X \uplus Y$ be a partition of $V(G)$. There exist integers $f(t,k)$ and $g(t,k)$ such that the following holds: If $|X| \geqslant f(t,k)$, and $|N_Y(x)| \geqslant g(t,k)$ for every $x \in X$, then G contains $kK_{1,k}$ as an induced subgraph. In particular, G admits k-witnesses for b-Chromatic Core and thus for Partial Grundy Coloring.*

Proof. We first observe that $kK_{1,k}$ (even $kK_{1,k-1}$) is a k-witness for b-Chromatic Core: color the k centers with distinct colors, and assign colors from $[k] \setminus \{i\}$ to the leaves of the center colored i. Let $N(t,1/k)$ and M be the integers defined in Lemmas 14 and 15 respectively, and set $M' := \max(M, N(t,1/k))$, $f(t,k) := 2^{2t+k(tk^t+t)}$, $g(t,k) := 2^{k(tk^t+t)}M'$.

By Ramsey's theorem, any graph on at least $f(t,k)$ vertices admits either a clique of size $2t$ or an independent set of size $k(tk^t+t)$. Since $G[X]$ (which has at least $f(t,k)$ vertices) is $K_{t,t}$-free, the former outcome is impossible, so it has an independent set of size $k(tk^t+t)$. It should be noted that the inductive proof of Ramsey's theorem yields a greedy linear-time algorithm which outputs a clique or an independent set of the required size. Hence we efficiently find an independent set of size $k(tk^t+t)$ in $G[X]$. Starting from $j=1$, we now prove the following claim inductively for all $j \leqslant k$.

> ($\star$) If $|X| \geqslant j(tk^t+t)$ and $|N_Y(x)| \geqslant 2^{j(tk^t+t)}M'$ for every $x \in X$, then there are j vertices $\{b_1,\ldots,b_j\} \subseteq X$ and a family of j disjoint vertex-sets $A_1,\ldots,A_j \subseteq Y$ each of size at least M', such that each A_i are private neighbors of b_i; that is, the vertices of A_i are adjacent with b_i and not adjacent with any other vertices from $\{b_1,\ldots,b_j\}$.

The claim ($\star$) trivially holds when $j=1$. Suppose it holds for all integers smaller than j, where $2 \leqslant j \leqslant k$. We may assume that X has precisely $j(tk^t+t)$ vertices by discarding some vertices if its size exceeds the bound. For each $\emptyset \neq I \subseteq X$, we define $N_I = \bigcap_{v\in I} N_Y(v) \cap \bigcap_{v\in X\setminus I} Y \setminus N_Y(v)$. Thus N_I corresponds to all the vertices of Y whose neighborhood in X is exactly I. Observe that the N_I's partition Y and that N_I corresponds to the set of vertices of Y that are complete with I and anti-complete with $X-I$.

Choose a vertex $x \in X$ that minimized $|N_Y(x)|$. As there are $2^{j(tk^t+t)}$ possible subsets of X and $|N_Y(x)| \geqslant 2^{j(tk^t+t)}M'$, there exists $I^* \subseteq X$ such that $x \in I^*$ and $|N_{I^*}| \geqslant M'$.

Let X_x be the set of vertices in X adjacent with at least k-th fraction of $N_Y(x)$, that is,

$$X_x = \{v \in X : |N_Y(v) \cap N_Y(x)| \geqslant \frac{|N_Y(x)|}{k}\}.$$

Set $X' = X - (I^* \cup X_x)$, $Y' = Y - N_Y(x)$ and let $G' = G[X' \cup Y']$.

We want to apply the induction hypothesis on G' with respect to X' and Y'. For this, we need to make sure that it satisfies the conditions of $(\star)$ for $j-1$. To prove that $|X'| \geqslant (j-1)(tk^t+t)$, we need to bound the size of I^* and X_x. To bound the size of I^*, notice that N_{I^*} is complete with I^*. Since $|N_{I^*}| \geqslant M \geqslant t$ and G is $K_{t,t}$-free, we conclude that $|I^*| < t$. To bound the size of X_x, we apply Lemma 15 with $\epsilon = 1/k$ to the hypergraph on the vertex-set $N_Y(x)$ and with hyperedge set $\{N_Y(v) \cap N_Y(x) : v \in X\}$. Each hyperedge of size at least $\frac{|N_Y(x)|}{k}$ corresponds to a vertex in X_x which gives us the bound $|X_x| \leqslant tk^t$. Notice that Lemma 15 can be legitimately applied on this hypergraph as $N_Y(x)$ has at least $M' \geqslant N(t, 1/k)$ vertices. Therefore, we have $|X'| \geqslant |X| - t - tk^t \geqslant (j-1)(tk^t+t)$.

It remains to verify that each $v \in X'$ has at least $2^{(j-1)(tk^t+t)}M'$ neighbors in Y'. Indeed

$$\begin{aligned}|N_{Y'}(v)| &\geqslant |N_Y(v)| - |N_Y(v) \cap N_Y(x)| \geqslant |N_Y(v)| - \frac{|N_Y(x)|}{k} \geqslant |N_Y(v)| - \frac{|N_Y(v)|}{k} \\ &\geqslant \frac{k-1}{k} 2^{j(tk^t+t)}M' \geqslant 2^{(j-1)(tk^t+t)}M'.\end{aligned}$$

This proves that G' meets the requirement to apply the induction hypothesis, and thus we can find $\{b_2, \ldots, b_j\}$ and sets $A_2, \ldots, A_j$ in G' as claimed in $(\star)$. Observe now that N_{I^*} is anticomplete to $\{b_2, \ldots, b_j\}$ and recall that $|N_{I^*}| \geqslant M'$. Hence, setting $b_1 = x$ and $A_1 = N_{I*}$ complete the proof of $(\star)$. Now, applying Lemma 14 to the sets $A_1 \uplus \cdots \uplus A_k$ given by $(\star)$ gives us either a clique on k vertices or the announced set of stars. ◀

Combined with the main result of the previous subsection, this implies that b-Chromatic Core and Partial Grundy Coloring can be solved in FPT time on $K_{t,t}$-free graphs. Observe that our algorithm is FPT in the combined parameter $k+t$, which is a stronger than having an FPT algorithm in k when t is a fixed constant.

▶ **Theorem 17.** *There is a function h and an algorithm which, given a graph $G = (V, E)$ not containing $K_{t,t}$ as a subgraph, decides whether G admits k b-Chromatic Core (Partial Grundy Coloring, respectively) in time $h(k,t)n^{O(1)}$.*

Proof. Let $X \subseteq B$ be the set of all vertices whose degree is at least $g(t,k) + f(t,k)$, where $g(t,k)$ and $f(t,k)$ are the integers as in Proposition 16. If X contains at least $f(t,k)$ vertices, then we there exists a k-witness in G by Proposition 16. If X contains less than $f(t,k)$ vertices, the algorithm of Theorem 12 can be applied to correctly decide whether G contains a k-witness. ◀

References

1 Pierre Aboulker, Jørgen Bang-Jensen, Nicolas Bousquet, Pierre Charbit, Frédéric Havet, Frédéric Maffray, and José Zamora. χ-bounded families of oriented graphs. *Journal of Graph Theory*, 89(3):304–326, 2018. doi:10.1002/jgt.22252.

2 Pierre Aboulker, Édouard Bonnet, Eun Jung Kim, and Florian Sikora. Grundy coloring & friends, half-graphs, bicliques. *CoRR*, abs/2001.03794, 2020. arXiv:2001.03794.

3 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

4 Édouard Bonnet, Florent Foucaud, Eun Jung Kim, and Florian Sikora. Complexity of grundy coloring and its variants. *Discrete Applied Mathematics*, 243:99–114, 2018. doi:10.1016/j.dam.2017.12.022.

5 Leizhen Cai. Parameterized complexity of cardinality constrained optimization problems. *Comput. J.*, 51(1):102–121, 2008. doi:10.1093/comjnl/bxm086.

6 Leizhen Cai, Siu Man Chan, and Siu On Chan. Random separation: A new method for solving fixed-cardinality optimization problems. In *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, pages 239–250, 2006. doi:10.1007/11847250_22.

7 Marco Cesati. The turing way to parameterized complexity. *J. Comput. Syst. Sci.*, 67(4):654–685, 2003. doi:10.1016/S0022-0000(03)00073-4.

8 Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michal Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. *SIAM J. Comput.*, 45(4):1171–1229, 2016. doi:10.1137/15M1032077.

9 Claude A. Christen and Stanley M. Selkow. Some perfect coloring properties of graphs. *Journal of Combinatorial Theory, Series B*, 27(1):49–59, 1979.

10 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

11 Brice Effantin, Nicolas Gastineau, and Olivier Togni. A characterization of b-chromatic and partial grundy numbers by induced subgraphs. *Discrete Mathematics*, 339(8):2157–2167, 2016. doi:10.1016/j.disc.2016.03.011.

12 Paul Erdős, W. R. Hare, Stephen T. Hedetniemi, and Renu Laskar. On the equality of the Grundy and ochromatic numbers of a graph. *Journal of Graph Theory*, 11(2):157–159, 1987. doi:10.1002/jgt.3190110205.

13 Paul Erdős, Stephen T. Hedetniemi, Renu C. Laskar, and Geert C.E. Prins. On the equality of the partial Grundy and upper ochromatic numbers of graphs. *Discrete Mathematics*, 272(1):53–64, 2003. In Honor of Frank Harary.

14 P Erdös. Some combinatorial, geometric and set theoretic problems in measure theory. In *Measure Theory Oberwolfach 1983*, pages 321–327. Springer, 1984.

15 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

16 Nicolas Gastineau. *Partitionnement, recouvrement et colorabilité dans les graphes. (Partitionability, coverability and colorability in graphs)*. PhD thesis, University of Burgundy, Dijon, France, 2014. URL: https://tel.archives-ouvertes.fr/tel-01136691.

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

18 Patrick Michael Grundy. Mathematics and games. *Eureka*, 2:6–8, 1939.

19 András Gyárfás, Zoltán Király, and Jenö Lehel. On-line 3-chromatic graphs - II critical graphs. *Discrete Mathematics*, 177(1-3):99–122, 1997.

20 Frédéric Havet, Ana Karolinna Maia, and Min-Li Yu. Complexity of greedy edge-colouring. *J. Braz. Comp. Soc.*, 21(1):18:1–18:7, 2015. doi:10.1186/s13173-015-0036-x.

21 Frédéric Havet and Leonardo Sampaio. On the Grundy and b-chromatic numbers of a graph. *Algorithmica*, 65(4):885–899, 2013. doi:10.1007/s00453-011-9604-4.

22 Frédéric Havet and Leonardo Sampaio. On the Grundy and b-chromatic numbers of a graph. *Algorithmica*, 65(4):885–899, 2013. doi:10.1007/s00453-011-9604-4.

23 Sandra M. Hedetniemi, Stephen T. Hedetniemi, and Terry Beyer. A linear algorithm for the Grundy (coloring) number of a tree. *Congressus Numerantium*, 36:351–363, 1982.

24 Dániel Marx. Parameterized complexity of independence and domination on geometric graphs. In *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, pages 154–165, 2006. doi:10.1007/11847250_14.

25 Dániel Marx. On the optimality of planar and geometric approximation schemes. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 338–348, 2007. doi:10.1109/FOCS.2007.50.

26 Dániel Marx. Can you beat treewidth? *Theory of Computing*, 6(1):85–112, 2010. doi:10.4086/toc.2010.v006a005.

27 Dániel Marx and Michal Pilipczuk. Optimal parameterized algorithms for planar facility location problems using voronoi diagrams. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 865–877, 2015. doi:10.1007/978-3-662-48350-3_72.

28 Dániel Marx and Michal Pilipczuk. Optimal parameterized algorithms for planar facility location problems using voronoi diagrams. *CoRR*, abs/1504.05476, 2015. arXiv:1504.05476.

29 Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 182–191. IEEE Computer Society, 1995. doi:10.1109/SFCS.1995.492475.

30 Fahad Panolan, Geevarghese Philip, and Saket Saurabh. On the parameterized complexity of b-chromatic number. *J. Comput. Syst. Sci.*, 84:120–131, 2017. doi:10.1016/j.jcss.2016.09.012.

31 Leonardo Sampaio. *Algorithmic aspects of graph colouring heuristics*. PhD thesis, University of Nice-Sophia Antipolis, November 2012.

32 Gustavus J. Simmons. The ochromatic number of graphs. *Graph Theory Newsletters*, 11(6):2–3, 1982.

33 Jan Arne Telle and Andrzej Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM Journal on Discrete Mathematics*, 10(4):529–550, 1997. doi:10.1137/S0895480194275825.

34 Jan Arne Telle and Yngve Villanger. FPT algorithms for domination in biclique-free graphs. In Leah Epstein and Paolo Ferragina, editors, *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, volume 7501 of *Lecture Notes in Computer Science*, pages 802–812. Springer, 2012. doi:10.1007/978-3-642-33090-2_69.

35 Manouchehr Zaker. The Grundy chromatic number of the complement of bipartite graphs. *Australasian Journal of Combinatorics*, 31:325–329, 2005.

36 Manouchehr Zaker. Results on the Grundy chromatic number of graphs. *Discrete Mathematics*, 306(23):3166–3173, 2006. doi:10.1016/j.disc.2005.06.044.

Lower Bounds Against Sparse Symmetric Functions of ACC Circuits: Expanding the Reach of #SAT Algorithms

Nikhil Vyas
MIT, Cambridge, MA, USA
nikhilv@mit.edu

R. Ryan Williams
MIT, Cambridge, MA, USA
rrw@mit.edu

Abstract

We continue the program of proving circuit lower bounds via circuit satisfiability algorithms. So far, this program has yielded several concrete results, proving that functions in Quasi-NP = NTIME$[n^{(\log n)^{O(1)}}]$ and NEXP do not have small circuits (in the worst case and/or on average) from various circuit classes $\mathcal{C}$, by showing that $\mathcal{C}$ admits non-trivial satisfiability and/or #SAT algorithms which beat exhaustive search by a minor amount.

In this paper, we present a new strong lower bound consequence of non-trivial #SAT algorithm for a circuit class $\mathcal{C}$. Say a symmetric Boolean function $f(x_1, \ldots, x_n)$ is *sparse* if it outputs 1 on $O(1)$ values of $\sum_i x_i$. We show that for every sparse f, and for all "typical" $\mathcal{C}$, faster #SAT algorithms for $\mathcal{C}$ circuits actually imply lower bounds against the circuit class $f \circ \mathcal{C}$, which may be *stronger* than $\mathcal{C}$ itself. In particular:

- #SAT algorithms for n^k-size $\mathcal{C}$-circuits running in $2^n/n^k$ time (for all k) imply NEXP does not have $f \circ \mathcal{C}$-circuits of polynomial size.
- #SAT algorithms for 2^{n^ε}-size $\mathcal{C}$-circuits running in 2^{n-n^ε} time (for some $\varepsilon > 0$) imply Quasi-NP does not have $f \circ \mathcal{C}$-circuits of polynomial size.

Applying #SAT algorithms from the literature, one immediate corollary of our results is that Quasi-NP does not have EMAJ $\circ$ ACC^0 $\circ$ THR circuits of polynomial size, where EMAJ is the "exact majority" function, improving previous lower bounds against ACC^0 [Williams JACM'14] and ACC^0 $\circ$ THR [Williams STOC'14], [Murray-Williams STOC'18]. This is the first nontrivial lower bound against such a circuit class.

2012 ACM Subject Classification Theory of computation → Circuit complexity

Keywords and phrases #SAT, satisfiability, circuit complexity, exact majority, ACC

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.59

Related Version A full version of the paper is available at `https://arxiv.org/abs/2001.07788`.

Funding Supported by NSF CCF-1741615 and NSF CCF-1909429.

1 Introduction

Currently, our knowledge of algorithms vastly exceeds our knowledge of lower bounds. Is it possible to bridge this gap, and use the existence of powerful algorithms to give lower bounds for hard functions? Over the last decade, the program of proving lower bounds via algorithms has been positively addressing this question. A line of work starting with Kabanets and Impagliazzo [15] has shown how deterministic subexponential-time algorithms for polynomial identity testing would imply lower bounds against arithmetic circuits. Starting around 2010 [24, 25], it was shown that even *slightly nontrivial* algorithms could imply Boolean circuit lower bounds. For example, a circuit satisfiability algorithm running in $O(2^n/n^k)$

37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020).
Editors: Christophe Paul and Markus Bläser; Article No. 59; pp. 59:1–59:17

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

time (for all k) on n^k-size circuits with n inputs would already suffice to yield the (infamously open) lower bound $\mathsf{NEXP} \not\subset \mathsf{P/poly}$. More generally, a generic connection was found between non-trivial SAT algorithms and circuit lower bounds:

▶ **Theorem 1** ([24, 25], Informal). *Let $\mathcal{C}$ be a circuit class closed under AND, projections, and compositions.*[1] *Suppose for all k there is an algorithm A such that, for every $\mathcal{C}$-circuit of n^k size, A determines its satisfiability in $O(2^n/n^k)$ time. Then* NEXP *does not have polynomial-size $\mathcal{C}$-circuits.*

To illustrate Theorem 1 with two examples, when $\mathcal{C}$ is the class of general fan-in 2 circuits, Theorem 1 says that non-trivial Circuit SAT algorithms imply $\mathsf{NEXP} \not\subset \mathsf{P/poly}$; when $\mathcal{C}$ is the class of Boolean formulas, it says non-trivial Formula-SAT algorithms imply $\mathsf{NEXP} \not\subset \mathsf{NC}^1$. Both are major open questions in circuit complexity. Theorem 1 and related results have been applied to prove several concrete circuit lower bounds: super-polynomial lower bounds for ACC^0 [25], $\mathsf{ACC}^0 \circ \mathrm{THR}$ [21], quadratic lower bounds for depth-two symmetric and threshold circuits [18, 1], and average-case lower bounds as well [7, 5].

Recently, the algorithms-to-lower-bounds connection has been extended to show a trade-off between the running time of the SAT algorithm on large circuits, and the complexity of the hard function in the lower bound. In particular, it is even possible in principle to obtain circuit lower bounds against NP with this algorithmic approach.

▶ **Theorem 2** ([16], Informal). *Let $\mathcal{C}$ be a class of circuits closed under unbounded AND, ORs of fan-in two, and negation. Suppose there is an algorithm A and $\varepsilon > 0$ such that, for every $\mathcal{C}$-circuit C of 2^{n^ε} size, A solves satisfiability for C in $O(2^{n-n^\varepsilon})$ time. Then* Quasi-NP *does not have polynomial-size $\mathcal{C}$-circuits.*[2]

In fact, Theorem 2 holds even if A only distinguishes between unsatisfiable circuits from those with at least 2^{n-1} SAT assignments; we call this easier problem GAP-UNSAT.

Intuitively, the aforementioned results show that as the circuit satisfiability algorithms improve in running time and scope, they imply stronger lower bounds. In all known results, to prove a lower bound against $\mathcal{C}$, one must design a SAT algorithm for a circuit class that is at least as powerful as $\mathcal{C}$. Inspecting the proofs of the above theorems carefully, it is not hard to show that, even if $\mathcal{C}$ did not satisfy the desired closure properties, it would suffice to give a SAT algorithm for a slightly more powerful class than the lower bound. For example, in Theorem 2, a SAT algorithm running in $O(2^{n-n^\varepsilon})$ time for 2^{n^ε}-size AND of ORs of three (possibly negated) $\mathcal{C}$ circuits (on n inputs, of 2^{n^ε} size) would still imply $\mathcal{C}$-circuit lower bounds for Quasi-NP. Our key point here is that *these proof methods require a SAT algorithm for a potentially more powerful circuit class than the class for which we can conclude a lower bound.* A compelling question is whether this requirement is an artifact of our proof method, or is it inherent?

Lower bounds for more powerful classes from SAT algorithms?

We feel it is natural to conjecture that a SAT algorithm for a circuit class $\mathcal{C}$ implies a lower bound against a class that is *more powerful* than $\mathcal{C}$, because checking satisfiability is itself a very powerful ability. Intuitively, a non-trivial SAT algorithm for $\mathcal{C}$ on n-input circuits is computing a *uniform OR* of 2^n $\mathcal{C}$-circuits evaluated on fixed inputs, in $o(2^n)$ time. (Recall

[1] It is not necessary to know precisely what these conditions mean, as we will use different conditions in our paper anyway. The important point is that these conditions hold for most interesting circuit classes that have been studied, such as AC^0, TC^0, NC^1, NC, and general fan-in two circuits.

[2] In this paper, we use the notation $\text{Quasi-NP} := \bigcup_k \mathrm{NTIME}[n^{(\log n)^k}]$.

that a "uniform" circuit informally means that any gate of the circuit can be efficiently computed by an algorithm.) If there were an algorithm to decide the outputs of uniform ORs of $\mathcal{C}$-circuits more efficiently than their actual circuit size, perhaps this implies a lower bound against $\mathrm{OR} \circ \mathcal{C}$ circuits.

Similarly, a #SAT algorithm for $\mathcal{C}$ on n-input circuits can be used to compute the output of any circuit of the form $f(C(x_1), \ldots, C(x_{2^n}))$ where f is a uniform symmetric Boolean function, C is a $\mathcal{C}$-circuit with n inputs, and $x_1, \ldots, x_{2^n}$ is an enumeration of all n-bit strings. Should we therefore expect to prove lower bounds on symmetric functions of $\mathcal{C}$-circuits, using a #SAT algorithm? This question is particularly significant because in many of the concrete lower bounds proved via the program [25, 21, 16], non-trivial #SAT algorithms were actually obtained, not just SAT algorithms. So our question amounts to asking: *how strong of a circuit lower bound we can prove, given the SAT algorithms we already have?* We use SYM to denote the class of Boolean symmetric functions.

▶ **Conjecture 1** (#SAT Algorithms Imply Symmetric Function Lower Bounds, Informal)**.** *Non-trivial #SAT algorithms for circuit classes $\mathcal{C}$ imply size lower bounds against $\mathsf{SYM} \circ \mathcal{C}$ circuits. In particular, all statements in Theorem 1 and Theorem 2 hold when the SAT algorithm is replaced by a #SAT algorithm, and the lower bound consquence for $\mathcal{C}$ is replaced by $\mathsf{SYM} \circ \mathcal{C}$.*

If Conjecture 1 is true, then existing #SAT algorithms would already imply super-polynomial lower bounds for $\mathsf{SYM} \circ \mathsf{ACC}^0 \circ \mathrm{THR}$ circuits, a class that contains depth-two symmetric circuits (for which no lower bounds greater than n^2 are presently known) [18, 1].

More intuition for Conjecture 1 can be seen from a recent paper of the second author, who showed how #SAT algorithms for a circuit class $\mathcal{C}$ can imply lower bounds on *(real-valued) linear combinations of $\mathcal{C}$-circuits* [23]. For example, known #SAT algorithms for ACC^0 circuits imply $\mathsf{Quasi\text{-}NP}$ problems cannot be computed via polynomial-size linear combinations of polynomial-size $\mathsf{ACC}^0 \circ \mathrm{THR}$ circuits. However, the linear combination representation is rather constrained: the linear combination is required to always output 0 or 1. Applying PCPs of proximity, Chen and Williams [6] showed that the lower bound of [23] can be extended to "approximate" linear combinations of $\mathcal{C}$-circuits, where the linear combination does not have to be exactly 0 or 1, but must be closer to the correct value than to the incorrect one, within an additive constant factor. These results show, in principle, how a #SAT algorithm for a circuit class $\mathcal{C}$ can imply lower bounds for a stronger class of representations than $\mathcal{C}$.

1.1 Conjecture 1 Holds for Sparse Symmetric Functions

In this paper, we take a concrete step towards realizing Conjecture 1, by proving it for "sparse" symmetric functions. We say a symmetric Boolean function $f(x_1, \ldots, x_n)$ is *k-sparse* if f is 1 on at most k values of $\sum_i x_i$. The 1-sparse symmetric functions are called the *exact threshold* (ETHR with polynomial weights) or *exact majority* (EMAJ) functions, which have been studied for years in both circuit complexity (e.g. [11, 4, 12, 13, 14]) and structural complexity theory, where the corresponding complexity class (computing an exact majority over all computation paths) is known as $\mathsf{C_{=}P}$ [20].

▶ **Theorem 3.** *Let $\mathcal{C}$ be closed under AND_2, negation, and suppose the all-ones and parity function are in $\mathcal{C}$. Let $f = \{f_n\}$ be a family of k-sparse symmetric functions for some $k = O(1)$.*

- *If there is a #SAT algorithm for n^k-size $\mathcal{C}$-circuits running in $2^n/n^k$ time (for all k), then NEXP does not have $f \circ \mathcal{C}$-circuits of polynomial size.*
- *If there is a #SAT algorithm for 2^{n^ε}-size $\mathcal{C}$-circuits running in 2^{n-n^ε} time (for some $\varepsilon > 0$), then $\mathsf{Quasi\text{-}NP}$ does not have $f \circ \mathcal{C}$-circuits of polynomial size.*

Applying known #SAT algorithms for $\mathsf{AC}^0[m] \circ \mathrm{THR}$ circuits from [22], we obtain:

▶ **Corollary 4.** *For all constant depths $d \geq 2$ and constant moduli $m \geq 2$,* Quasi-NP *does not have polynomial-size EMAJ $\circ$ $\mathsf{AC}^0[m] \circ$ THR circuits.*

1.2 Intuition

Here we briefly explain the new ideas that lead to our new circuit lower bounds.

As in prior work [23, 6], the high-level idea is to show that if (for example) Quasi-NP has polynomial-size EMAJ $\circ \mathcal{C}$, and there is a #SAT algorithm for $\mathcal{C}$ circuits, then we can design a nondeterministic algorithm for verifying GAP Circuit Unsatisfiability (GAP-UNSAT) on generic circuits that beats exhaustive search. In GAP-UNSAT, we are given a generic circuit and are promised that it is either unsatisfiable, or at least half of its possible assignments are satisfying, and we need to nondeterministically prove the unsatisfiable case. (Note this is a much weaker problem than SAT.) As shown in [24, 25, 16], combining a nondeterministic algorithm for GAP-UNSAT with the hypothesis that Quasi-NP has polynomial-size circuits, we can derive that nondeterministic time 2^n can be simulated in time $o(2^n)$, contradicting the nondeterministic time hierarchy theorem.

Our key idea is to use probabilistically checkable proofs (PCPs) in a new way to exploit the power of a #SAT algorithm. First, let's observe a task that a #SAT algorithm for $\mathcal{C}$ can compute on an EMAJ $\circ \mathcal{C}$ circuit. Suppose our EMAJ $\circ \mathcal{C}$ circuit has the form

$$D(x) = \left[\sum_{i=1}^{t} C_i(x) = s \right],$$

where each $C_i(x)$ is a Boolean $\mathcal{C}$-circuit on n inputs, s is a threshold value, and our circuit outputs 1 if and only if the sum of the C_i's equals s.[3] Consider the expression

$$E(x) := \left(\sum_{i=1}^{t} C_i(x) - s \right)^2. \tag{1}$$

Treated as a function, $E(x)$ outputs integers; $E(a) = 0$ when $D(a) = 1$, and otherwise $E(a) \in [1, (t+s)^2]$. We first claim that the quantity

$$\sum_{a \in \{0,1\}^n} E(a) \tag{2}$$

can be compute faster than exhaustive search using a faster #SAT algorithm. To see this, using distributivity, we can rewrite (1) as

$$E(x) = \sum_{i,j} (C_i \wedge C_j)(x) - 2s \sum_i C_i(x) + s^2.$$

Assuming $\mathcal{C}$ is closed under conjunction, each $C_i \wedge C_j$ is also a $\mathcal{C}$-circuit, and we can compute

$$\sum_{a \in \{0,1\}^n} E(a) = \sum_{i,j} \left(\sum_{a \in \{0,1\}^n} (C_i \wedge C_j)(a) \right) - 2s \sum_i \left(\sum_{a \in \{0,1\}^n} C_i(a) \right) + s^2 \cdot 2^n$$

by making $O(t^2)$ calls to a #SAT algorithm. Thus we can compute (2) using a #SAT algorithm.

[3] We are using the standard Iverson bracket notation, where $[P]$ is 1 if predicate P is true, and 0 otherwise.

How is computing (2) useful? This is where PCPs come in. We cannot use (2) to directly solve #SAT for D (otherwise as #SAT algorithms imply SAT algorithms we could apply existing work [25], and be done). But we can use (2) to obtain a *multiplicative approximation* to the number of assignments that *falsify* D. In particular, each satisfying assignment is counted zero times in (2), and each falsifying assignment is counted between 1 and (less than) $(t+s)^2$ times. We want to exploit this, and obtain a faster GAP-UNSAT algorithm. Given a circuit which is a GAP-UNSAT instance, we start by using an efficient hitting set construction [10] to increase the gap of GAP-UNSAT. We obtain a new circuit $C(x)$ which is either UNSAT or has at least $2^n - o(2^n)$ satisfying assignments (Section 2.1). Next (Lemma 15) we apply a PCP of Proximity and an error correcting code to C, yielding a 3-SAT instance over x and extra variables, with constant gap (similar to Chen-Williams [6]), and we amplify this gap using standard serial repetition. Finally, we apply the FGLSS [9] reduction (Lemma 19) to the 3-SAT instance, obtaining Independent Set instances with a large gap between the YES case and NO case. In particular, for all inputs x, when $C(x) = 1$ there is a large independent set in the resulting graph, and when $C(x) = 0$, there are only small independent sets in the resulting graph (see Lemma 14). Returning to the assumption that Quasi-NP has small EMAJ $\circ\, \mathcal{C}$ circuits, and applying an easy witness lemma [16], it follows that the solutions to the independent set instance can be encoded by EMAJ $\circ\, \mathcal{C}$ circuits. Because of the large gap between the YES case and NO case, our multiplicative approximation to the number of UNSAT assignments can be used to distinguish the unsatisfiable case and the "many satisfying assignments" case of GAP-UNSAT, which finishes the argument.

One interesting bottleneck is that we cannot *directly* apply serial repetition and the FGLSS reduction in our argument; we need the PCP machinery we use to behave similarly on all inputs x to the original circuit C. This translates to studying the behavior of these reductions *with respect to partial assignments*. While for these two reductions we are able to prove that they behave "nicely" with respect to partial assignments, it is entirely unclear that this is true for other PCP reductions such alphabet reduction, parallel repetition, and so on.

Our approach is very general; to handle k-sparse symmetric functions, we can simply modify the function E accordingly.

2 Preliminaries and Organization

We assume general familiarity with basic concepts in circuit complexity and computational complexity [2]. In particular we assume familiarity with AC^0, ACC^0, $\mathsf{P}_{/\mathsf{poly}}$, NEXP, and so on.

Circuit Notation

Here we define notation for the relevant circuit classes. By $\text{size}_{\mathcal{C}}(h(n))$ we denote circuits from circuit class $\mathcal{C}$ with size at most $h(n)$.

▶ **Definition 5.** *An EMAJ$\circ\mathcal{C}$ circuit (a.k.a. "exact majority of $\mathcal{C}$ circuit") has the general form $EMAJ(C_1(x), C_2(x), \ldots, C_t(x), u)$, where u is a positive integer, x are the input variables, $C_i \in \mathcal{C}$, and the gate $EMAJ(y_1, \ldots, y_t, u)$ outputs 1 if and only if exactly u of the y_i's output 1.*

▶ **Definition 6.** *A $SUM^{\geq 0} \circ \mathcal{C}$ circuit ("positive sum of $\mathcal{C}$ circuits") has the form*

$$SUM^{\geq 0}(C_1(x), C_2(x), \ldots, C_t(x)) = \sum_{i \in [t]} C_i(x)$$

where C_i is either a $\mathcal{C}$-circuit or -1 times a $\mathcal{C}$-circuit and we are promised that $\sum_{i\in[t]} C_i(x) \geq 0$ over all $x \in \{0,1\}^n$.

Given a set of circuits $\{C_i\}$, we say that $f : \{0,1\}^n \to \{0,1\}$ is represented *by the positive-sum circuit $SUM^{\geq 0}(C_1(x), C_2(x), \ldots, C_t(x))$ if for all x, $f(x) = 1$ when $\sum_{i\in[t]} C_i(x) > 0$, and $f(x) = 0$ when $\sum_{i\in[t]} C_i(x) = 0$.*

▶ **Definition 7.** *A circuit class $\mathcal{C}$ is* typical *if there is a $k > 0$ such that the following hold:*

- ***Closure under negation.*** *For every $\mathcal{C}$ circuit C, there is a circuit C' computing the negation of C where $size(C') \leq size(C)^k$.*
- ***Closure under AND.*** *For every $\mathcal{C}$ circuits C_1 and C_2, there is a circuit C' computing the AND of C_1 and C_2 where $size(C') \leq (size(C_1) + size(C_2))^k$.*
- ***Contains all-ones.*** *The function $\mathbf{1}_n : \{0,1\}^n \to \{0,1\}$ has a $\mathcal{C}$ circuit of size $O(n^k)$.*

The vast majority of circuit classes that are studied (AC^0, ACC^0, TC^0, NC^1, $\mathsf{P}_{/\mathsf{poly}}$) are typical.[4] The next lemma shows that the negation of an exact-majority of $\mathcal{C}$ circuit can be represented as a "positive-sum" of $\mathcal{C}$ circuit, if $\mathcal{C}$ is typical.

▶ **Lemma 8.** *Let $\mathcal{C}$ be typical. If a function f has a $EMAJ \circ \mathcal{C}$ circuit D of size s, then $\neg f$ can be represented by a $SUM^{\geq 0} \circ \mathcal{C}$ circuit D' of size $poly(s)$. Moreover, a description of the circuit D' can be obtained from a description of D in polynomial time.*

Proof. Suppose f is computable by the $\mathrm{EMAJ} \circ \mathcal{C}$ circuit $D = \mathrm{EMAJ}(D_1, D_2, \ldots, D_t, u)$, where $u \in \{0, 1, \ldots, t\}$. Consider the expression

$$E(x) := (\mathrm{SUM}(D_1, D_2, \ldots, D_t) - u)^2.$$

Note that $E(x) = 0$ when $D(x) = 1$, and $E(x) > 0$ when $D(x) = 0$. So in order to prove the lemma, it suffices to show that E can be written as a $\mathrm{SUM}^{\geq 0} \circ \mathcal{C}$ circuit. Expanding the expression E,

$$\begin{aligned} E(x) &= \mathrm{SUM}(D_1, D_2, \ldots, D_t)^2 - 2u \cdot \mathrm{SUM}(D_1, D_2, \ldots, D_t) + u^2 \\ &= \sum_{i,j=1}^{t} (D_i \wedge D_j) - \sum_{j=1}^{2u} \sum_{i=1}^{t} D_i + u^2. \end{aligned}$$

By Definition 7 $\mathrm{AND}_2 \circ \mathcal{C} = \mathcal{C}$, each $D_i \wedge D_j$ is a circuit from $\mathcal{C}$ of size $\mathrm{poly}(s)$. Since the all-ones function is in $\mathcal{C}$, the function $x \mapsto u^2$ also has a $\mathrm{SUM} \circ \mathcal{C}$ circuit of size $O(t^2)$. Therefore there are circuits $D'_i \in \mathcal{C}$ and $t' \leq O(t^2)$ such that by defining $D' := \mathrm{SUM}^{\geq 0}(D'_1, \ldots, D'_{t'})$ we have $D'(x) = E(x)$ for all x. ◀

Error-Correcting Codes

We will need a (standard) construction of binary error correcting codes with constant rate and constant relative distance.

▶ **Theorem 9** ([17])**.** *There are universal constants $c \geq 1$ and $\delta \in (0,1)$ such that for all sufficiently large n, there are linear functions $ENC^n : (\mathbb{F}_2)^n \to (\mathbb{F}_2)^{cn}$ such that for all $x \neq y$ with $|x| = |y| = n$, the Hamming distance between $ENC^n(x)$ and $ENC^n(y)$ is at least δn.*

In what follows, we generally drop the superscript n for notational brevity. Note that each bit of output $\mathrm{ENC}^n_i(x)$ (for $i = 1, \ldots, cn$) is a parity function on some subset of the input bits.

[4] A notable exception (as far as we know) is the class of depth-d *exact* threshold circuits for a fixed $d \geq 2$, because we do not know if such classes are closed under negation. Similarly, we do not know if the class of depth-d threshold circuits is typical. (In that case, the only non-trivial property to check is closure under AND; we can compute the AND of two threshold circuits with a quasi-polynomial blowup using Beigel-Reingold-Spielman [3], but not with a polynomial blowup.)

2.1 Weak CAPP Algorithms Are Sufficient For Lower Bounds

Murray and Williams [16] showed that CAPP/GAP-UNSAT algorithms, i.e., algorithms which distinguish between unsatisfiable circuits and circuits with $\geq 2^{n-1}$ satisfying assignments are enough to give lower bounds. For our results, it is necessary to strengthen the "gap", which can be done using known hitting set constructions.

▶ **Lemma 10** (Corollary C.5 in [10], Hitting Set Construction). *There is a constant $\psi > 0$ and a $\text{poly}(n, \log g)$ time algorithm S such that, given a (uniform random) string r of $n+\psi\cdot\log g$ bits, S outputs $t = O(\log g)$ strings $x_1, x_2, \ldots, x_t \in \{0,1\}^n$ such that for every $f : \{0,1\}^n \to \{0,1\}$ with $\sum_x f(x) \geq 2^{n-1}$, $\Pr_r[OR_{i=1}^t f(x_i) = 1] \geq 1 - 1/g$.*

We will use the following "algorithms to lower bounds" connections as black box:

▶ **Theorem 11** ([16]). *Suppose for some constant $\varepsilon \in (0,1)$ there is an algorithm A that for all 2^{n^ε}-size circuits C on n inputs, $A(C)$ runs in 2^{n-n^ε} time, outputs YES on all unsatisfiable C, and outputs NO on all C that have at least 2^{n-1} satisfying assignments. Then for all k, there is a $c \geq 1$ such that $NTIME[2^{\log^{ck^4/\varepsilon} n}] \not\subset \mathsf{SIZE}[2^{\log^k n}]$.*

Applying Lemma 10 to Theorem 11, we observe that the circuit lower bound consequence can be obtained from a significantly weaker-looking hypothesis. This weaker hypothesis will be useful for our lower bound results.

▶ **Theorem 12.** *Suppose for some constant $\varepsilon \in (0,1)$ there is an algorithm A that for all 2^{n^ε}-size circuits C on n inputs, $A(C)$ runs in $2^n/g(n)^{\omega(1)}$ time, outputs YES on all unsatisfiable C, and outputs NO on all C that have at least $2^n(1-1/g(n))$ satisfying assignments, for $g(n) = 2^{n^{2\varepsilon}}$. Then for all k, there is a $c \geq 1$ such that $NTIME[2^{\log^{ck^4/\varepsilon} n}] \not\subset \mathsf{SIZE}[2^{\log^k n}]$.*

Proof. Our starting point is Theorem 11 ([16]): we are given an m-input, 2^{m^δ}-size circuit D' that is either UNSAT or has at least 2^{m-1} satisfying assignments, and we wish to distinguish between the two cases with a 2^{m-m^δ}-time algorithm. We set $\delta = \varepsilon/2$

We create a new circuit D with n inputs, where n satisfies

$$n = m + \psi \cdot \log g(n),$$

and $\psi > 0$ is the constant from Lemma 10. (Note that, since $g(n)$ is time constructible and $g(n) \leq 2^{o(n)}$, such an n can be found in subexponential time.) Applying the algorithm from Lemma 10, D treats its n bits of input as a string of randomness r, computes $t = O(\log g(n))$ strings $x_1, x_2, \ldots, x_t \in \{0,1\}^m$ with a $\text{poly}(m, \log g)$-size circuit, then outputs the OR of $D'(x_i)$ over all $i = 1, \ldots, t$. Note the total size of our circuit D is $\text{poly}(m, \log g) + O(\log g) \cdot \text{size}(D') = \text{poly}(n) + O(n^{2\varepsilon}) \cdot 2^{m^\delta} < 2^{n^{2\delta}} = 2^{n^\varepsilon}$ as $\varepsilon = 2\delta$.

Clearly, if D' is unsatisfiable, then D is also unsatisfiable. By Lemma 10, if D' has 2^{m-1} satisfying assignments, then D has at least $2^n(1-1/g(n))$ satisfying assignments. As $\text{size}(D) \leq 2^{n^\varepsilon}$, by our assumption we can distinguish the case where D is unsatisfiable from the case where D has at least $2^n(1-1/g(n))$ satisfying assignments, with an algorithm running in time $2^n/g(n)^{\omega(1)}$. This yields an algorithm for distinguishing the original circuit D' on m inputs and 2^{m^δ} size, running in time

$$2^n/g(n)^{\omega(1)} = 2^m g(n)^{O(1)}/g(n)^{\omega(1)} = 2^m/g(n)^{\omega(1)} \leq 2^m 2^{-n^{2\varepsilon}} \leq 2^m 2^{-n^\delta} \leq 2^{m-m^\delta},$$

since $g(n) = 2^{n^{2\varepsilon}}$. By Theorem 11, this implies that for all k, there is a $c \geq 1$ such that $\text{NTIME}[2^{\log^{ck^4/\delta} n}] \not\subset \mathsf{SIZE}[2^{\log^k n}]$. As, $\varepsilon = 2\delta$ we get that $\text{NTIME}[2^{\log^{2ck^4/\varepsilon} n}] \not\subset \mathsf{SIZE}[2^{\log^k n}]$. But as the constant 4 can be absorbed in the constant c hence we get that for all k, there is a $c \geq 1$ such that $\text{NTIME}[2^{\log^{ck^4/\varepsilon} n}] \not\subset \mathsf{SIZE}[2^{\log^k n}]$. ◀

2.2 Organization

In Section 3 we give a reduction from Circuit SAT to "Generalized" Independent Set. Section 4 uses this reduction to prove lower bounds for EMAJ$\circ\mathcal{C}$ assuming #SAT algorithms for $\mathcal{C}$ with running time $2^{n-n^{\varepsilon}}$. Section 4.1 uses this result to give lower bound for EMAJ$\circ$ACC$^0\circ$THR. Section 5 generalizes these results to $f\circ\mathcal{C}$ lower bounds where f is a sparse symmetric function. In the full version of the paper [19] we give lower bounds for EMAJ$\circ\mathcal{C}$ assuming #SAT algorithms for $\mathcal{C}$ with running time $2^n/n^{\omega(1)}$.

3 From Circuit SAT to Independent Set

The goal of this section is to give the main PCP reduction we will use in our new algorithm-to-lower-bound theorem. First we need a definition of "generalized" independent set instances, where some vertices have already been "assigned" in or out of the independent set.

▶ **Definition 13.** *Let $G=(V,E)$ be a graph. Let $\pi: V\to\{0,1,*\}$ be a partial Boolean assignment to V. We define $G(\pi)$ to be a graph with the label function π on its vertices (where each vertex gets the label 0, or 1, or no label). We construe $G(\pi)$ as an* ***generalized independent set instance****, in which any valid independent set (vertex assignment) must be consistent with π: any independent set must contain all vertices labeled 1, and no vertices labeled 0.*

▶ **Lemma 14.** *Let k be a function of n. Given a circuit D on X with $|X|=n$ bits and of size $m>n$, there is a poly$(m,2^{O(k)})$-time reduction from D to a generalized independent set instance on graph $G_D=(V_D,E_D)$, with the following properties.*

- *Each vertex $v\in V_D$ is associated with a set of pairs S_v of the form $\{(i,b)\}\subseteq[O(n)]\times\{0,1\}$. The set $\{S_v\}$ is produced as part of the reduction.*
- *Each assignment x to X defines a partial assignment π_x to V_D such that*

$$\pi_x(v)=\begin{cases}0 & \text{if } \exists(i,b)\in S_v \text{ such that } ENC_i(x)\neq b\\ * & \text{otherwise,}\end{cases}$$

  where ENC is the error-correcting code from Theorem 9.
- *If $D(x)=0$, the maximum independent set in $G_D(\pi_x)$ equals κ for an integer κ, and furthermore given x, it can be found in time poly$(n,m,2^{O(k)})$.*
- *If $D(x)=1$, then the maximum independent set in $G_D(\pi_x)$ has size at most $\kappa/2^k$.*

Intuitively, the use of Lemma 14 is that we will start with a "no satisfying assignment" vs "most assignments are satisfying" GAP-UNSAT instance from Theorem 12. Now in the "no satisfying assignment" case for all x the reduced independent set instance $G_D(\pi_x)$ has a large independent set instance. Counting the sum of independent sets over x gives a high value. On the other hand in the "most assignments are satisfying" case for most x the reduced independent set instance $G_D(\pi_x)$ has a small independent set and for a very few x, $G_D(\pi_x)$ can have a large independent set. Hence in this case counting the sum of independent sets over all x gives a low value. The difference between the high value and low value is big enough that even a approximate counting of these values as outlined in Section 1.2 is enough to distinguish and hence solve the GAP-UNSAT instance.

The remainder of this section is devoted to the proof of Lemma 14.

Let us set up some notation for variable assignments to a formula. Let F be a SAT instance on a variable set Z, and let $\tau: Z\to\{0,1,\star\}$ be a partial assignment to Z. Then we define $F(\tau)$ to be the formula obtained by setting the variables in F according to τ. Note that we do not perform further reduction rules on the clauses in $F(\tau)$: for each clause in F that becomes false (or true) under τ, there is a clause in $F(\tau)$ which is always false (true).

For every subsequence Y of variables from Z, and every vector $y \in \{0,1\}^{|Y|}$, we define $F(Y = y)$ to be the formula F in which the i^{th} variable in Y is assigned y_i, and all other variables are left unassigned.

▶ **Lemma 15** (PCPP+ECC, [6]). *There is a polynomial-time transformation that, given a circuit D on n inputs of size $m \geq n$, outputs a 3-SAT instance F on the variable set $Y \cup Z$, where $|Y| \leq poly(n)$, $|Z| \leq poly(m)$, and the following hold for all $x \in \{0,1\}^n$:*

- *If $D(x) = 0$ then $F(Y = ENC(x))$ on variable set Z has a satisfying assignment z_x. Furthermore, there is a $poly(m)$-time algorithm that given x outputs z_x.*
- *if $D(x) = 1$ then there is no assignment to the Z variables in $F(Y = ENC(x))$ satisfying more than a $(1 - \Omega(1))$-fraction of the clauses.*

where $ENC : \{0,1\}^n \to \{0,1\}^{O(n)}$ is the linear encoding function from Theorem 9. As it is a linear function, the i^{th} bit of output $ENC_i(x)$ satisfies $ENC_i(x) = \oplus_{j \in U_i} x_j$ for some set U_i.

Serial Repetition [8] is a basic operation on CSPs/PCPs, in which a new CSP is created whose constraints are ANDs of k uniformly sampled clauses from the original CSP. Serial repetition is usually done for the purpose of reducing soundness, i.e., reducing the fraction of satisfiable clauses. We now state a derandomized version of serial repetition.

▶ **Lemma 16** (Serial repetition [8]). *Given a 3-SAT instance F on n variables denoted by Y with m clauses we can construct a $O(k)$-SAT formula F' on the same n variables with $m2^{O(k)}$ clauses such that:*

1. *If $Y = y$ satisfies F then y satisfies F'.*
2. *If $F(Y = y)$ is at most $1 - \Omega(1)$ satisfiable then $F'(Y = y)$ is at most $1/2^k$ satisfiable.*

Next we prove a stronger version of derandomized serial repetition with guarantees for partial assignments. The proof directly follows from the guarantees of standard Serial Repetition (Lemma 16).

▶ **Lemma 17** (Serial repetition with partial assignments). *Let k be a function of n. Given a 3-SAT instance F on n variables denoted by Y, Z with m clauses we can construct a $O(k)$-SAT formula F' on the same n variables with $m \cdot 2^{O(k)}$ clauses such that:*

1. *If $Y, Z = y, z$ satisfies F then y, z satisfies F'.*
2. *If $F(Y = y)$ is at most $1 - \Omega(1)$ satisfiable then $F'(Y = y)$ is at most $1/2^k$ satisfiable.*

Proof. We prove that just standard serial repetition from Lemma 16 suffices for proving this stronger property.

Property 1 directly follows from Property 1 in Lemma 16.

Define $F_y = F(Y = y)$ where we treat any clauses that became FALSE or TRUE under $Y = y$ as normal clauses. Let F'_y be the $O(k)$-SAT formula obtained by applying serial repetition to f_y from Lemma 17.

In Serial Repetition [8] it is clear that clauses in F' are just ANDs of clauses in F and which clauses are part of the "AND" is only dependent on their index.

Due to this $F'(Y = y)$ i.e. first applying serial repetition then setting $Y = y$ is equivalent to first setting $Y = y$ and then applying serial repetition i.e. F'_y.

By our assumption F_y is at most $1 - \Omega(1)$ satisfiable and hence by Property 2 of Lemma 16 F'_y is at most $1/2^k$ satisfiable. As $F'_y = F'(Y = y)$ we have that $F'(Y = y)$ is at most $1/2^k$ satisfiable. ◀

The FGLSS reduction [9] maps a CSP Φ to a graph G_Φ such that the MAX-SAT value in Φ is equal to the size of the maximum independent set in G_Φ.

▶ **Lemma 18** (FGLSS [9]). *Let F be a k-SAT instance on variable set Y with $|Y| = n$ and m clauses. There exists a poly$(n, m, 2^{O(k)})$ time reduction graph from F to a graph $G_F = (V_F, E_F)$ such that: the size of maximum independent set in G_F is exactly equal to maximum clauses satisfiable in F.*

We note that a stronger version of the FGLSS reduction [9] holds with guarantees for partial assignments. The proof is very similar to the proof of the standard FGLSS reduction (Lemma 18).

▶ **Lemma 19** (FGLSS with partial assignments). *Let F be a k-SAT instance on variable set Y, Z with $|Y| + |Z| = n$ and m clauses. There exists a poly$(n, m, 2^{O(k)})$ time reduction graph from F to an independent set instance on graph $G_F = (V_F, E_F)$. Each vertex $v \in V_F$ is a associated to a set T_v of $(i \in [|Y|], b \in \{0, 1\})$ pairs. For each partial assignment of the form $\tau : Y \to \{0, 1\}$ define a partial assignment π_τ to V_F such that:*

$$\pi_\tau(v) = \begin{cases} 0 & \text{if } \exists (i, b) \in T_v \text{ such that } \tau(Y_i) \neq b \\ * & \text{otherwise,} \end{cases}$$

Then the max independent set in $G_F(\pi_\tau)$ equals the max number of clauses satisfiable in $F(\tau)$.

Proof. Let w be a clause in F and w_i denote the i^{th} variable in w. Let ℓ denote a satisfying assignment to w. For every w, ℓ pair create a vertex in V_F. Let v be the vertex associated with a particular w, ℓ. Let $T_v = \{(w_i, \ell_i\}$ represent the assignment $w_i = \ell_i$ for $1 \leq i \leq k$.

Make an edge between vertex u and vertex v if the assignment T_u and T_v contradict each other. Note that this means that there is always an edge between two vertices associated to the same clause but different satisfying assignments i.e. vertices associated with the same clause form a clique.

Let x be a assignment for F satisfying κ clauses. We now give an independent set in G_F of size κ. For every satisfied clause w and and ℓ the assignment to variables of w in x we choose the vertex w, ℓ in the independent set. As there are κ satisfied clauses we choose κ vertices. These vertices form and independent set as if two of these vertices u, v had an edge between them it would mean that the assignments T_u and T_v contradict each other. This is not possible as all these assignments are partial assignments of x.

Consider S to be an independent set in G_F of size κ. We now give an assignment to F which satisfies κ clauses. Note that from vertices corresponding to the same clauses only 1 vertex can be a part of independent set as they all form a clique. Hence vertices associated with κ different clauses must be part of the independent set. For a vertex u associated with w, ℓ the partial assignment T_u satisfies w. For two vertices u, v in the independent set the partial assignments from T_v and T_u do not contradict as otherwise there would be an edge between u and v. Hence we can join all the partial assignments T_v for vertices v in the independent set to get a partial assignment which satisfies κ clauses in $F(\tau)$. Hence the maximum independent set in $G_F(\pi_\tau)$ has size at most the maximum number clauses satisfied in $F(\tau)$. ◀

We next present the proof of Lemma 14 which just follows by combining Lemma 15, 17, and 19 sequentially.

Proof of Lemma 14. The proof follows by applying Lemma 15, 17 and 19 sequentially.

We start from a circuit D with input variables X ($|X| = n$) and size $m > n$. Lemma 15 transform this into a 3-SAT instance F with poly(m) clauses on the variable set $Y \cup Z$, where $|Y| \leq \text{poly}(n)$, $|Z| \leq \text{poly}(m)$, and the following hold for all $x \in \{0, 1\}^n$:

- If $D(x) = 0$ then $F(Y = \mathrm{ENC}(x))$ on variable set Z has a satisfying assignment z_x. Furthermore, there is a $\mathrm{poly}(m)$-time algorithm that given x outputs z_x.
- if $D(x) = 1$ then there is no assignment to the Z variables in $F(Y = \mathrm{ENC}(x))$ satisfying more than a $(1 - \Omega(1))$-fraction of the clauses.

where $\mathrm{ENC} : \{0,1\}^n \to \{0,1\}^{O(n)}$ is the linear encoding function from Theorem 9.

Applying Lemma 17 on F gives us a $O(k)$-SAT formula F' on the same $Y \cup Z$ variables with $\mathrm{poly}(m) \cdot 2^{O(k)}$ clauses such that:

1. If $Y, Z = y, z$ satisfies F then y, z satisfies F'.
2. If $F(Y = y)$ is at most $1 - \Omega(1)$ satisfiable then $F'(Y = y)$ is at most $1/2^k$ satisfiable.

which implies that:

- If $D(x) = 0$ then $F'(Y = \mathrm{ENC}(x))$ on variable set Z has a satisfying assignment z_x. Furthermore, there is a $\mathrm{poly}(m)$-time algorithm that given x outputs z_x.
- if $D(x) = 1$ then there is no assignment to the Z variables in $F'(Y = \mathrm{ENC}(x))$ satisfying more than a $1/2^k$-fraction of the clauses.

Finally applying Lemma 19 to F' where we consider partial assignments τ which assign Y to $\mathrm{ENC}(x)$ for some x. Hence $\tau(Y_i) = \mathrm{ENC}_i(x)$. As τ is fixed by fixing x we rename π_τ to π_x. S_v is just a renaming of T_v. Size of the graph is $\mathrm{poly}(n + m, \mathrm{poly}(m) \cdot 2^{O(k)}, 2^{O(k)}) = \mathrm{poly}(m, 2^k)$ as $m > n$. ◀

4 Main Result

We now turn to the proof of the main result, Theorem 3. We will prove the result for $\mathrm{EMAJ} \circ \mathcal{C}$ first, and sketch how to extend to $f \circ \mathcal{C}$ for sparse symmetric f in Section 5. Below we prove $\mathrm{EMAJ} \circ \mathcal{C}$ lower bounds for Quasi-NP when we have 2^{n-n^ε} time algorithms for #SAT on $\mathcal{C}$ circuits of size 2^{n^ε}. For the other parts of Theorem 3 (on #SAT algorithms with running time $2^n/n^{\omega(1)}$), see the full version of the paper [19].

We note here that in Theorem 3 we mentioned polynomial size lower bounds for $\mathrm{EMAJ} \circ \mathcal{C}$ we in fact prove quasi-polynomial size lower bounds below.

▶ **Theorem 20.** *Suppose $\mathcal{C}$ is typical, and the parity function has poly(n)-sized $\mathcal{C}$ circuits. Then for every k,* quasi-NP *does not have $\mathrm{EMAJ} \circ \mathcal{C} = \mathcal{H}$ circuits of size $O(n^{\log^k n})$, if for some $\epsilon \in (0,1)$ there is a #SAT algorithm running in time 2^{n-n^ε} for all circuits from class $\mathcal{C}$ of size at most 2^{n^ε}.*

Proof. Let us assume that for a fixed $k > 0$, quasi-NP has $\mathcal{H} = \mathrm{EMAJ} \circ \mathcal{C}$ circuits of size $O(n^{(\log^k n)})$ which implies that quasi-NP $\in \mathrm{size}(n^{O(\log^k n)})$ for general circuits. By Theorem 12, we obtain a contradiction if for some constant $\delta \in (0,1)$ and $g(n) = 2^{n^{2\delta}}$ we can give a $2^n/g(n)^{\omega(1)}$ time nondeterministic algorithm for distinguishing between:

1. YES case: D has no satisfying assignments.
2. NO case: D has at least $2^n\,(1 - 1/g(n))$ satisfying assignments

given a generic fan-in 2 circuit D with n inputs and size $m \leq h(n) := 2^{n^\delta}$. Under the hypothesis, we will give such an algorithm for $\delta = \varepsilon/4$.

Using Lemma 14, we reduce the circuit D to an independent set instance G_D (with $k = \log h(n)$) on $n_2 = \mathrm{poly}(m, 2^{O(k)}) = \mathrm{poly}(m, 2^{O(k)}) = \mathrm{poly}(m, h(n)^{O(1)}) = \mathrm{poly}(h(n))$ vertices. We also find subsets S_i for every vertex $i \in [n_2]$. Let π_x be the partial assignment which assigns a vertex i to 0 if there exist $(j', b) \in S_i$ such that $\mathrm{ENC}_{j'}(x) \neq b$. Note that π_x does not assign any vertex to 1. By Lemma 14, G_D has the following properties:

1. If $D(x) = 0$, then $G_D(\pi_x)$ has an independent set of size κ. Furthermore, given x we can find this independent set in $\text{poly}(h(n))$ time.
2. If $D_1(x) = 1$, then in $G_D(\pi_x)$, all independent sets have size at most $\kappa/h(n)$.

This means it suffices for us to distinguish between the following two cases:
1. YES case: For all x, $G_D(\pi_x)$ has an independent set of size κ.
2. NO case: For at most $2^n/g(n)$ values of $x, G_D(\pi_x)$ has an independent set of size $\geq \kappa/h(n)$.

Guessing a succinct witness circuit: As guaranteed by Lemma 14 given an x such that $D(x) = 0$ we can find the assignment $A(x)$ to G_D which is consistent with π_x and represents an independent set of size κ in $\text{poly}(h(n))$ time. Let $A(x, i)$ denote the assignment to the i^{th} vertex in $A(x)$. Given x and vertex $i \in [n_2]$, in time $\text{poly}(h(n))$ we can produce $\neg A(x, i)$.

▷ Claim 21. Under the hypothesis, there is a $h(n)^{o(1)}$-sized EMAJ $\circ\, \mathcal{C}$ circuit U of size $h(n)^{o(1)}$ with x, i as input representing $\neg A(x, i)$.

Proof. Under the hypothesis, for some constant k, we have quasi-NP $\subseteq \text{size}_{\mathcal{H}}[n^{\log^k n}]$. Specifically, for $p(n) = n^{\log^{k+1} n}$ we have $\text{NTIME}[p(n)] \subseteq \text{size}_{\mathcal{H}}[p(n)^{1/\log n}] \subseteq \text{size}_{\mathcal{H}}[p(n)^{o(1)}]$. As $h(n) = 2^{n^\varepsilon} \gg p(n)$, a standard padding argument implies $\text{NTIME}[\text{poly}(h(n))] \subseteq \text{size}_{\mathcal{H}}[(\text{poly}(h(n)))^{o(1)}] = \text{size}_{\mathcal{H}}[h(n)^{o(1)}]$. Since $\neg A(x, i)$ is computable in $\text{poly}(h(n))$ time, we have that $\neg A(x, i)$ can be represented by a $h(n)^{o(1)}$-sized $\mathcal{H} = \text{EMAJ} \circ \mathcal{C}$ circuit. ◁

Our nondeterministic algorithm for GAP-UNSAT begins by guessing U guaranteed by Claim 21 which is supposed to represent $\neg A$. Then by the reduction in Lemma 8 we can covert U to a $\text{SUM}^{\geq 0} \circ \mathcal{C}$ circuit R for $A(x, i)$ of size $\text{poly}(h(n)^{o(1)}) = h(n)^{o(1)}$. Note that if our guess for U is correct, i.e., $U = \neg A$, then R represents A.

Let the subcircuits of R be $R_1, R_2, \ldots, R_t$, so that $R(x) = \sum_{j\in[t]} R_j$, where $R_j \in \mathcal{C}$ and $t \leq h(n)^{o(1)}$. The number of inputs to R_j is $n' = |x| + \log n_2 = n + O(\log h(n))$, and the size of R_j is $h(n)^{o(1)}$.

Note that $R(x, i) = 0$ represents that the i^{th} vertex is not in the independent set of G_D in a solution corresponding to x, while $R(x, i) > 0$ represents that it is in the independent set of G_D in a solution corresponding to x. For all x and i we have $0 \leq R(x, i) \leq t \leq h(n)^{o(1)}$.

Verifying that R encodes valid independent sets: We can verify that the circuit R produces an independent set on all x by checking each edge over all x. To check the edge between vertices i_1 and i_2 we need to verify that at most one of them is in the independent set. Equivalently, for all x we check that $R(x, i_1) \cdot R(x, i_2) = 0$. As $R(x, i) \geq 0$ for all x and i we can just verify

$$\sum_{x\in\{0,1\}^n} R(x, i_1) \cdot R(x, i_2) = 0.$$

Since $R(x, i) = \sum_{j\in[t]} R_j(x, i)$ it suffices to verify that

$$\sum_{x\in\{0,1\}^n} \sum_{j_1,j_2\in[t]} R_{j_1}(x, i_1) \cdot R_{j_2}(x, i_2) = 0.$$

Let $R_{j_1,j_2}(x, i_1, i_2) = R_{j_1}(x, i_1) \cdot R_{j_2}(x, i_2)$. Since $\mathcal{C}$ is closed under AND (upto polynomial factors) R_{j_1,j_2} also has a $\text{poly}(h(n)^{o(1)}) = h(n)^{o(1)}$ sized $\mathcal{C}$ circuit. Exchanging the order of summations is suffices for us to verify

$$\sum_{j_1,j_2\in[t]} \left(\sum_{x\in\{0,1\}^n} R_{j_1,j_2}(x, i_1, i_2) \right) = 0.$$

For fixed i_1, i_2, j_1, j_2 the number of inputs to R_{j_1,j_2} is $|x| = n$ and its size is $h(n)^{o(1)} \leq 2^{n^\varepsilon}$. Hence, for fixed i_1, i_2, j_1, j_2 we can compute $\sum_x R_{j_1,j_2}(x, i_1, i_2)$ using the #SAT algorithm from our assumption, in time 2^{n-n^ε}. Summing over all j_1, j_2 pairs only adds another multiplicative factor of $t^2 = h(n)^{o(1)}$. This allows us to verify that the edge (i_1, i_2) is satisfied by R. Checking all edges of G_D only adds another multiplicative factor of $\text{poly}(h(n))$. Hence the total running time for verifying that R encodes valid independent sets on all x is still $2^{n-n^\varepsilon}\text{poly}(h(n))$.

Verifying consistency of independent set produced by R with π_x: As we care about the sizes of independent sets in $G_D(\pi_x)$ over all x we need to check if the assignment by R is consistent with π_x. As π_x only assigns vertices to 0, we need to verify that all vertices assigned to 0 in π_x are in fact assigned to 0 by the assignment given by $R(x, \cdot)$. From Lemma 14, we know that π_x assigns a vertex i to 0 if for some $(j', b) \in S_i$, $\text{ENC}_{j'}(x) \neq b$. To check this condition we need to verify that $R(x, i) = 0$ if for some $(j', b) \in S_i$, $\text{ENC}_{j'}(x) \neq b$. Equivalently, we cn check $(\text{ENC}_{j'}(x) \oplus b) \cdot R(x, i) = 0$ for all $x, i, (j', b) \in S_i$. Since $(\text{ENC}_j(x) \oplus b)R(x, i) \geq 0$ for all possible inputs we can just check that

$$\sum_{x \in \{0,1\}^n} (\text{ENC}_{j'}(x) \oplus b) \cdot R(x, i) = 0$$

for all $i, (j', b) \in S_i$. As $R(x, i) = \sum_{j \in [t]} R_j(x, i)$ we can equivalently verify that

$$\sum_{x \in \{0,1\}^n} \sum_{j \in [t]} (\text{ENC}_{j'}(x) \oplus b) \cdot R_j(x, i) = 0$$

for all $i, (j', b) \in S_i$. Note that $R_{j'}(x, i)$ has a $h(n)^{o(1)}$ sized $\mathcal{C}$ circuit. By our assumption parity has a poly(n)-sized $\mathcal{C}$-circuit so $(\text{ENC}_j(x) \oplus b)$ also has a poly(n)-sized $\mathcal{C}$ circuit. Hence $(\text{ENC}_j(x) \oplus b) \cdot R_{j'}(x, i)$ has a $\text{poly}(n, h(n)^{o(1)}) = h(n)^{o(1)}$-sized $\mathcal{C}$ circuit, since $\mathcal{C}$ is closed under AND.

For fixed (i, j, j'), $(\text{ENC}_{j'}(x) \oplus b) \cdot R_j(x, i) \in \mathcal{C}$ has $|x| = n$ inputs and size $h(n)^{o(1)} < 2^{n^\varepsilon}$. Hence we can use our assumed #SAT algorithm to calculate $\sum_{x \in \{0,1\}^n} (\text{ENC}_{j'}(x) \oplus b) \cdot R_j(x, i)$ in time 2^{n-n^ε}. Summing over all $j \in [t]$ introduces another multiplicative factor of $h(n)^{o(1)}$. This allows us to verify the desired condition for a fixed $i, (j', b) \in S_i$. To check it for all $i, (j', b) \in S_i$ (recall $|S_i| = O(n)$ by Theorem 9) only introduces another multiplicative factor of $\text{poly}(h(n)) \cdot O(n) = \text{poly}(h(n))$ in time. Therefore the total running time for verifying consistency w.r.t. π_x is $2^{n-n^\varepsilon}\text{poly}(h(n))$.

At this point, we now know that R represents an independent set, and that R is consistent with π_x. We need to distinguish between:

1. YES case: For all x, $R(x, \cdot)$ represents an independent set of size κ.
2. NO case: For at most $2^n/g(n)$ values of $x, R(x, \cdot)$ represents an independent set of size $\geq \kappa/h(n)$.

▶ **Lemma 22.** *For all x such that $R(x, \cdot)$ represents an independent set of size a. we have $a \leq \sum_{i \in [n_2]} R(x, i) \leq at$.*

Proof. For every vertex i in the independent set, $1 \leq R(x, i) \leq t$. For all vertices i not in the independent set, we have $R(x, i) = 0$. Hence $a \leq \sum_{i \in [n_2]} R(x, i) \leq at$. ◀

Distinguishing between the YES and NO cases: To distinguish between the YES and NO cases, we now compute

$$\sum_{x \in \{0,1\}^n} \sum_{i \in [n_2]} R(x, i) \tag{3}$$

This allows us to distinguish between the YES case and NO case as:

1. YES case: We have for at least $2^n(1-1/g(n))$ values of x we have an independent set of size at most $\kappa/h(n)$. By Lemma 22 for such x, $\sum_{i\in[n_2]} R(x,i) \leq t\kappa/h(n)$. for the rest of $2^n/g(n)$ values of x the independent set could be all the vertices in the graph G_D. Hence by Lemma 22 for such values of x, $\sum_{i\in[n_2]} R(x,i) \leq tn_2 = \text{poly}(h(n))$. Hence

$$\begin{aligned}\sum_{x\in\{0,1\}^n}\sum_{i\in[n_2]} R(x,i) &\leq (2^n/g(n))\text{poly}(h(n)) + 2^n t\kappa/h(n)\\ &\leq o(2^n) + 2^n t\kappa/h(n) \quad [\text{As } h(n) = g(n)^{o(1)}]\\ &\leq o(2^n) + o(2^n\kappa) \quad [\text{As } t = h(n)^{o(1)}]\\ &\leq 2^n\kappa \quad [\text{As } \kappa > 1]\end{aligned}$$

2. NO case: We have for all $x \in \{0,1\}^n$ the independent set is at least of size κ. Hence by Lemma 22 the sum is $\sum_{x\in\{0,1\}^n}\sum_{i\in[n_2]} R(x,i) > 2^n\kappa$.

   All that remains is how to compute (3). As $R(x,i) = \sum_{j\in[t]} R_j(x,i)$, we can compute

$$\sum_{x\in\{0,1\}^n}\sum_{i\in[n_2]}\sum_{j\in[t]} R_j(x,i) = \sum_{j\in[t]}\sum_{i\in[n_2]}\sum_{x\in\{0,1\}^n} R_j(x,i)$$

For a fixed i,j, $R_j(x,i) \in \mathcal{C}$, it has $|x| = n$ inputs and size $\leq \text{poly}(h(n)^{o(1)}) = h(n)^{o(1)} < 2^{n^\varepsilon}$. Hence we can use the assumed #SAT algorithm to calculate $\sum_{x\in\{0,1\}^n} R_j(x,i)$ in time 2^{n-n^ε}. Summing over all $j \in [t], i \in [n_2]$ only introduces another $h(n)^{o(1)}\text{poly}(h(n)) = \text{poly}(h(n))$ multiplicative factor. Thus the running time for distinguishing the two cases is $2^{n-n^\varepsilon}\text{poly}(h(n))$.

In total our running time comes to $2^{n-n^\varepsilon}\text{poly}(h(n)) = 2^{n-n^{4\delta}+O(n^\delta)} \leq 2^{n-n^{3\delta}} = 2^n/g(n)^{\omega(1)}$ as $g(n) = 2^{n^{2\delta}}$ and $\varepsilon = 4\delta$. By Theorem 12, this gives us a contradiction which completes our proof. ◀

The above theorem when combined with known #SAT algorithms for $\mathsf{ACC}^0 \circ \text{THR}$ gives an quasi-NP lower bound for $\text{EMAJ} \circ \mathsf{ACC}^0 \circ \text{THR}$.

4.1 EMAJ ∘ ACC⁰ ∘ THR Lower bound

We will apply a known #SAT algorithm for $\mathsf{ACC} \circ \text{THR}$ circuits.

▶ **Theorem 23** ([22]). *For every pair of constants d, m, there exists a constant $\varepsilon \in (0,1)$ such that #SAT can be solved in time 2^{n-n^ε} time for $\mathsf{AC}^0[m] \circ$ THR circuits of depth d and size 2^{n^ε}.*

▶ **Theorem 24.** *For constants k, d, m,* quasi-NP *does not have size$(n^{\log^k n})$ EMAJ$\circ \mathsf{ACC}^0 \circ$ THR circuits of depth d.*

Proof. We first note that $\mathsf{ACC}^0 \circ \text{THR}$ is indeed typical and can represent $\text{ENC}(x)$ by $\text{poly}(n)$-sized circuits as $\text{ENC}(x) : \{0,1\}^n \to \{0,1\}^{O(n)}$ is a linear function.

By Theorem 23 we know that for all constants d there exists some constant $\epsilon \in (0,1)$ such that there exists a #SAT algorithm running in time 2^{n-n^ε} for all circuits from class $\mathsf{ACC}^0 \circ \text{THR}$ of size $\leq 2^{n^\varepsilon}$ and depth d.

The above properties imply that $\mathsf{ACC}^0 \circ \text{THR}$ satisfies the preconditions of Theorem 20 and hence for every pair of constant k, d, quasi-NP does not have size$(n^{\log^k n})$ $\text{EMAJ} \circ \mathsf{ACC}^0 \circ \text{THR}$ circuits of depth d. ◀

The above theorem can be rewritten as: For constants k, d, m, there exists a constant e such that NTIME$[n^{\log^e n}]$ does not have $n^{\log^k n}$-size EMAJ $\circ$ ACC^0 $\circ$ THR circuits of depth d. Here the constant e depends on d and m. Using a standard trick (as in [16]) this dependence can be removed as we show below.

▶ **Corollary 25.** *There exists an e such that NTIME$[n^{\log^e n}]$ does not have polynomial size EMAJ $\circ$ ACC^0 $\circ$ THR circuits.*

Proof. Assume for contradiction that for all e, there exists constants d, m such that NTIME$[n^{\log^e n}]$ has poly-sized EMAJ $\circ$ AC$^0[m]$ $\circ$ THR circuit of depth d. This implies that P has poly-sized EMAJ $\circ$ AC$^0[m]$ $\circ$ THR circuits, which further implies that *CIRCUIT EVALUATION* problem has poly-sized EMAJ $\circ$ AC$^0[m_0]$ $\circ$ THR circuit of a fixed constant depth d_0 and fixed constant m_0. Hence any circuit of size s has an equivalent poly(s)-sized EMAJ$\circ$AC$^0[m_0]\circ$THR circuit of depth d_0. Combining this with our assumption yields: For all e, there exists constants d, m such that NTIME$[n^{\log^e n}]$ has poly-sized EMAJ$\circ$AC$^0[m_0]\circ$THR circuit of depth d_0. This contradicts Theorem 24 and hence our assumption was wrong, which completes the proof. ◀

5 Extension to All Sparse Symmetric Functions

Our lower bounds extend to circuit classes of the form $f \circ \mathcal{C}$ where f denotes a family of symmetric functions that only take the value 1 on a small number of slices of the hypercube. Formally, let $f : \{0,1\}^n \to \{0,1\}$ be a symmetric function, and let $g : \{0,1,\ldots,n\} \to \{0,1\}$ be its "companion" function, where for all x, $f(x) = g(\sum_i x_i)$ (here, x_i denotes the i-th bit of x). For $k \in \{0,1,\ldots,n\}$, we say that a symmetric function f is *k-sparse* if $|g^{-1}(1)| = k$. For example, the all-zeroes function is 0-sparse, the all-ones function is n-sparse, and the EMAJ function is 1-sparse.

▶ **Theorem 26.** *Let $k < n/2$. Every k-sparse symmetric function $f : \{0,1\}^n \to \{0,1\}$ can be represented as an exact majority of $n^{O(k)}$ ANDs on k inputs.*

Proof. Given a k-sparse f and its companion function g, consider the polynomial expression

$$E(x) := \prod_{v \in g^{-1}(1)} \left(\sum_i x_i - v \right).$$

Then $E(x) = 0$ whenever $f(x) = 1$, and $E(x) \neq 0$ otherwise. Expanding E into a sum of products, we can write E as a multilinear n-variate polynomial of degree at most k, with integer coefficients of magnitude at most $n^{O(k)}$ (since each $v \leq n$). We can therefore write E as the EMAJORITY of $n^{O(k)}$ distinct ANDs on up to k inputs. ◀

The above theorem immediately implies that for every k-sparse symmetric function f_m, any circuit with an f_m at the output gate can be rewritten as a circuit with an EMAJ of fan-in at most $m^{O(k)}$ at the output gate (and ANDs of fan-in up to k below that).

▶ **Corollary 27.** *For every fixed k, and every k-sparse symmetric function family $f = \{f_n\}$,* Quasi-NP *does not have polynomial-size $f \circ$ ACC^0 $\circ$ THR circuits.*

References

1 Josh Alman, Timothy M. Chan, and R. Ryan Williams. Polynomial representations of threshold functions and algorithmic applications. In *FOCS*, pages 467–476, 2016.

2 Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. URL: `http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264`.

3 Richard Beigel, Nick Reingold, and Daniel A. Spielman. PP is closed under intersection. *J. Comput. Syst. Sci.*, 50(2):191–202, 1995. `doi:10.1006/jcss.1995.1017`.

4 Richard Beigel, Jun Tarui, and Seinosuke Toda. On probabilistic ACC circuits with an exact-threshold output gate. In *Algorithms and Computation, Third International Symposium, ISAAC '92, Nagoya, Japan, December 16-18, 1992, Proceedings*, pages 420–429, 1992.

5 Lijie Chen. Non-deterministic quasi-polynomial time is average-case hard for ACC circuits. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1281–1304, 2019.

6 Lijie Chen and R. Ryan Williams. Stronger connections between circuit analysis and circuit lower bounds, via pcps of proximity. In *34th Computational Complexity Conference, CCC 2019, July 18-20, 2019, New Brunswick, NJ, USA.*, pages 19:1–19:43, 2019.

7 Ruiwen Chen, Igor Carboni Oliveira, and Rahul Santhanam. An average-case lower bound against acc^0. In *LATIN 2018: Theoretical Informatics - 13th Latin American Symposium, Buenos Aires, Argentina, April 16-19, 2018, Proceedings*, pages 317–330, 2018.

8 Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the PCP theorem. *SIAM J. Comput.*, 36(4):975–1024, 2006. `doi:10.1137/S0097539705446962`.

9 Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Approximating clique is almost np-complete. In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 2–12, 1991.

10 Oded Goldreich. A sample of samplers: A computational perspective on sampling. 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 302–332. Springer, 2011.

11 Frederic Green. A complex-number fourier technique for lower bounds on the mod-m degree. *Computational Complexity*, 9(1):16–38, 2000. `doi:10.1007/PL00001599`.

12 Kristoffer Arnsfelt Hansen. Computing symmetric boolean functions by circuits with few exact threshold gates. In *Computing and Combinatorics, 13th Annual International Conference, COCOON 2007, Banff, Canada, July 16-19, 2007, Proceedings*, pages 448–458, 2007. `doi:10.1007/978-3-540-73545-8_44`.

13 Kristoffer Arnsfelt Hansen. Depth reduction for circuits with a single layer of modular counting gates. In *Computer Science - Theory and Applications, Fourth International Computer Science Symposium in Russia, CSR 2009, Novosibirsk, Russia, August 18-23, 2009. Proceedings*, pages 117–128, 2009. `doi:10.1007/978-3-642-03351-3_13`.

14 Kristoffer Arnsfelt Hansen and Vladimir V. Podolskii. Exact threshold circuits. In *Proceedings of the 25th Annual IEEE Conference on Computational Complexity, CCC 2010, Cambridge, Massachusetts, USA, June 9-12, 2010*, pages 270–279, 2010. `doi:10.1109/CCC.2010.33`.

15 Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004. `doi:10.1007/s00037-004-0182-6`.

16 Cody Murray and R. Ryan Williams. Circuit lower bounds for nondeterministic quasi-polytime: an easy witness lemma for NP and NQP. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 890–901, 2018.

17 Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Trans. Information Theory*, 42(6):1723–1731, 1996. doi:10.1109/18.556668.

18 Suguru Tamaki. A satisfiability algorithm for depth two circuits with a sub-quadratic number of symmetric and threshold gates. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:100, 2016.

19 Nikhil Vyas and Ryan Williams. Lower bounds against sparse symmetric functions of acc circuits: Expanding the reach of #SAT algorithms, 2020. URL: https://drive.google.com/open?id=1NjlPk9FZPY3DHvxpbB7nWE3fz71E16mf.

20 Klaus W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Inf.*, 23(3):325–356, 1986. doi:10.1007/BF00289117.

21 R. Ryan Williams. New algorithms and lower bounds for circuits with linear threshold gates. *Theory of Computing*, 14(1):1–25, 2018.

22 R. Ryan Williams. New algorithms and lower bounds for circuits with linear threshold gates. *Theory of Computing*, 14(1):1–25, 2018. doi:10.4086/toc.2018.v014a017.

23 Richard Ryan Williams. Limits on representing boolean functions by linear combinations of simple functions: Thresholds, relus, and low-degree polynomials. In *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*, pages 6:1–6:24, 2018. doi:10.4230/LIPIcs.CCC.2018.6.

24 Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM Journal on Computing*, 42(3):1218–1244, 2013.

25 Ryan Williams. Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1):2:1–2:32, 2014. doi:10.1145/2559903.

Reversible Pebble Games and the Relation Between Tree-Like and General Resolution Space

Jacobo Torán
Universität Ulm, Ulm, Germany
jacobo.toran@uni-ulm.de

Florian Wörz[1]
Universität Ulm, Ulm, Germany
florian.woerz@uni-ulm.de

Abstract

We show a new connection between the space measure in tree-like resolution and the reversible pebble game in graphs. Using this connection, we provide several formula classes for which there is a logarithmic factor separation between the space complexity measure in tree-like and general resolution. We show that these separations are not far from optimal by proving upper bounds for tree-like resolution space in terms of general resolution clause and variable space. In particular we show that for any formula F, its tree-like resolution space is upper bounded by $\text{space}(\pi)\log\big(\text{time}(\pi)\big)$, where π is any general resolution refutation of F. This holds considering as $\text{space}(\pi)$ the clause space of the refutation as well as considering its variable space. For the concrete case of Tseitin formulas, we are able to improve this bound to the optimal bound $\text{space}(\pi)\log n$, where n is the number of vertices of the corresponding graph.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases Proof Complexity, Resolution, Tree-like Resolution, Pebble Game, Reversible Pebbling, Prover-Delayer Game, Raz–McKenzie Game, Clause Space, Variable Space

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.60

Related Version An extended version of the paper is available at [26], `https://eccc.weizmann.ac.il/report/2019/097/`.

Funding This research was supported by the Deutsche Forschungsgemeinschaft (DFG) under project number 430150230, "Complexity measures for solving propositional formulas".

Acknowledgements The authors would like to thank the reviewers for many insightful comments.

1 Introduction

Resolution is one of the best-studied systems for refuting unsatisfiable propositional formulas. This is due to its theoretical simplicity, as well as its practical importance since it is the proof system at the root of many modern SAT solvers. Several complexity measures for the analysis of resolution refutations have been used in the last decades. In this paper, we will mainly concentrate on space bounds, which measure the amount of memory that is needed in a resolution refutation. Intuitively, the clause space (CS) measures the number of clauses required simultaneously in a refutation, while the variable space (VS) measures the maximum number of distinct variables kept simultaneously in memory during this process. Experimental results have shown that space measures for resolution correlate well with the hardness of refuting unsatisfiable formulas with SAT solvers in practice [2, 18].

[1] Corresponding author.

Tree-like resolution is a restricted kind of resolution that is especially important since the original DPLL algorithm [13, 12] on which many SAT solvers are based, is equivalent to this restriction of the resolution system. Contrary to general resolution, in tree-like resolution, if a clause is needed more than once in a refutation, it has to be rederived each time. It is known that general resolution can be exponentially more efficient than tree-like resolution in terms of length (number of clauses in a refutation) [8, 3]. In [3], the authors give an almost optimal separation between general and tree-like resolution. They show that for each natural number n, there are unsatisfiable formulas in $\mathrm{O}(n)$ variables that have resolution refutations of length L, linear in n, but for which any tree-like resolution refutation of the formula requires length $\exp\big(\Omega(\frac{L}{\log L})\big)$. They also give an almost matching upper bound of $\exp\big(\mathrm{O}\big(\frac{L \log\log L}{\log L}\big)\big)$ for the tree-like resolution length of any formula that can be refuted in length L by general resolution.

In this paper we study space separations between general and tree-like resolution. Space separations are much more modest than the ones for length. It is known from [15] that all space measures considered here for a formula with n variables are between constant and $n+2$. Also, it is not hard to see that variable space coincides in general and tree-like resolution. Therefore, we only consider the clause space measure for the case of tree-like resolution. The first space separation between general and tree-like resolution was given in [16]. There, a family of formulas $(F_n)_{n=1}^{\infty}$ was presented which require tree-resolution clause space s_n but have a general resolution refutation in clause space $c \cdot s_n$, for some constant $c < 1$. More recently, in [18], a family of formulas $(F_n)_{n=1}^{\infty}$ is presented with $\mathrm{O}(n)$ variables that can be refuted by general resolution in constant clause space but requires $\Theta(\log n)$ tree-like resolution space, thus showing that both measures are fundamentally different.

In this paper, we present a systematic study of tree-like resolution space providing upper bounds for this measure, which show that the logarithmic factor in the separation of [18] as well as in other separations provided here are basically optimal. Our main tools are several versions of pebbling games played on graphs, which have been extensively used in the past for analysing different computation models and in particular for analysing proof systems (see [20] for an excellent survey). We formally define these games in the preliminaries. Intuitively, the idea of the pebble game is to measure the number of pebbles needed by a single player in order to place a pebble on the sink of a directed acyclic graph following certain rules. In the standard game, pebbles can only be placed on a vertex if it is a source or if all its direct predecessors already have a pebble, but they can be removed at any time. In the reversible pebble game, pebbles can only be placed or removed from a vertex if all the direct predecessors of the vertex contain a pebble. Based on the pebble game, a class of contradictory formulas, called pebbling formulas, was introduced in [6]. These formulas have been extremely useful for analysing several proof systems. The reason for this is that some of the pebbling properties of the underlying graphs can be translated into parameters for the complexity of their corresponding pebbling contradictions. Known results of pebbling can therefore be translated into proof complexity results.

Our main contribution is a new connection between tree-like resolution clause space and the reversible pebble game. We show that for any graph G, the tree-like resolution space of a (certain kind of) pebbling contradiction of the graph is at least the reversible pebbling number of G and at most twice this number. More interestingly, we show that for *any* unsatisfiable CNF formula F, the tree-like resolution clause space of a refutation of F is at most the reversible pebbling number of any refutation graph of F, not necessarily a tree-like refutation. This result adds one more connection to the rich set of interrelations between pebbling and resolution [20]. A central tool in the proofs of these results is the Raz–McKenzie game [23], a

two-player game on graphs, and the fact that this game is equivalent to reversible pebbling in a precise sense [10]. The clause space measure for any formula can be exactly characterised in terms of the black pebble game on a refutation graph of the formula [15]. We find the fact that tree-like clause space is upper bounded by the reversible pebble game quite surprising.

Using these bound and known results on reversible pebbling [11, 30], we show in Section 3 that there are families of pebbling formulas $(F_n)_{n=1}^{\infty}$ with $\mathrm{O}(n)$ variables, that have general clause space $\mathrm{O}(s)$ and tree-like resolution space $\Omega(s \log n)$ for any function s smaller than $n^{1/2-\varepsilon}$. This separation (as well as the one in [18]) is almost optimal since we also show that for any pebbling formula F, its tree-like clause space is at most $\min_{\mathcal{P}} \big(\mathsf{space}(\mathcal{P}) \cdot \log \mathsf{time}(\mathcal{P})\big)$, where $\mathcal{P}$ is a black pebbling of the underlying graph of F. This means that for graphs of size n where the smallest black pebbling space is achieved in a one-shot pebbling strategy, that is, a strategy in which every vertex in the graph is pebbled at most once, the $\log n$ factor in the separation is optimal and the only room for improvement is with graph families in which the space-optimal black pebbling is not one-shot. It is possible that for one such family, the $\log n$ separation factor can be improved to a $\log \mathsf{time}(\mathcal{P})$ factor. We provide, however, for the first time a family of graphs for which the minimum pebbling space is obtained in a strategy that is not one-shot, but for which the clause space separation between general and tree-like resolution is also only a $\log n$ factor. We conjecture that this is optimal, and that this separation cannot be improved for other graph classes. This question is closely related to proving optimal upper bounds for reversible pebbling in terms of black pebbling. Another motivation for providing this new graph family is to increase the set of examples of formulas with concrete resolution space bounds that can be used for the testing of SAT solvers, as done for example in [18].

In Section 4, we prove upper bounds on the tree-like clause space for *any* unsatisfiable CNF formula F in terms of the variable space and clause space for general resolution of the formula. We use the *amortised space measures* for resolution introduced by Razborov in [24], that penalise configurational proofs for being unreasonably long. In his paper he defined the notations $\mathrm{VS}^*(F \vdash \square) := \min_{\pi: F \vdash \square} \big(\mathrm{VS}(\pi) \cdot \log \mathrm{L}(\pi)\big)$ and $\mathrm{CS}^*(F \vdash \square) := \min_{\pi: F \vdash \square} \big(\mathrm{CS}(\pi) \cdot \log \mathrm{L}(\pi)\big)$, where $\mathrm{L}(\pi)$ is the length of the configurational proof π. We show the upper bounds $\text{Tree-CS}(F \vdash \square) \leq \mathrm{VS}^*(F \vdash \square) + 2$ and $\text{Tree-CS}(F \vdash \square) \leq \mathrm{CS}^*(F \vdash \square) + 2$. The first inequality is especially interesting since it shows that clause space can be meaningfully bounded in terms of variable space, a question posed by Razborov in [24]: $\mathrm{CS}(F \vdash \square) \leq \mathrm{VS}^*(F \vdash \square) + 2$. Again, from the separations in Sections 3 and 5, the only room for improvement in this upper bounds is to decrease the $\log \mathrm{L}(\pi)$ factor to a $\log n$ factor, where n is the size of the formula F.

Finally, in Section 5, we give optimal separations for the space in tree-like resolution for the class of Tseitin formulas. We show that for any graph G with n vertices and odd marking χ, the inequalities $\text{Tree-CS}\big(\mathrm{Ts}(G,\chi) \vdash \square\big) \leq \mathrm{CS}\big(\mathrm{Ts}(G,\chi) \vdash \square\big) \cdot \log n + 2$ and $\text{Tree-CS}\big(\mathrm{Ts}(G,\chi) \vdash \square\big) \leq \mathrm{VS}\big(\mathrm{Ts}(G,\chi) \vdash \square\big) \cdot \log n + 2$ hold, thus improving the upper bound from the previous sections from logarithmic in the resolution length down to a $\log n$ factor. We also provide a class of formulas with a matching space separation showing that this is optimal.

2 Preliminaries

For a positive integer n we let $[n] := \{1, 2, \ldots, n\}$. The base of all logarithms in this paper is 2. The *size of a graph* is the number of vertices of it. Given a directed acyclic graph (*DAG*) $G = (V, E)$, we say that a vertex u is a *direct predecessor* of a vertex v, if there is a directed

edge from u to v. We denote by $\text{pred}_G(v)$ the *set of all direct predecessors* of v in G. The *maximal in-degree* of a graph G is defined to be $\max_{v \in V} |\text{pred}_G(v)|$. A vertex in a DAG with no incoming edges is called a *source* and a vertex with no outgoing edges is called a *sink*.

2.1 Pebble Games

Black pebbling was first mentioned implicitly in [21]. Note, that there exist several variants of the pebble game in the literature. In this paper, we focus on the variant without *sliding* and requiring the sink of the graph to be pebbled at the end. For differences between these variants, we refer to the survey [20], from which we borrowed most of our notation. For the following definitions, let $G = (V, E)$ be a DAG with a unique sink vertex z.

▶ **Definition 1** (Black pebble game). *The* black pebble game *on G is the following one-player game: At any time i of the game, we have a* pebble configuration $\mathbb{P}_i$*, where $\mathbb{P}_i \subseteq V$ is the set of black pebbles. A pebble configuration $\mathbb{P}_{i-1}$ can be changed to $\mathbb{P}_i$ by applying exactly one of the following rules:*

Black pebble placement on v: *If all direct predecessors of an empty vertex v have pebbles on them, a black pebble may be placed on v. More formally, letting $\mathbb{P}_i = \mathbb{P}_{i-1} \cup \{v\}$ is allowed if $v \notin \mathbb{P}_{i-1}$ and $\text{pred}_G(v) \subseteq \mathbb{P}_{i-1}$. In particular, a black pebble can always be placed on an empty source vertex s, since $\text{pred}_G(s) = \varnothing$.*

Black pebble removal from v: *A black pebble may be removed from any vertex at any time. Formally, if $v \in \mathbb{P}_{i-1}$, then we can set $\mathbb{P}_i = \mathbb{P}_{i-1} \setminus \{v\}$.*

A black pebbling *of G is a sequence of pebble configurations $\mathcal{P} = (\mathbb{P}_0, \mathbb{P}_1, \ldots, \mathbb{P}_t)$ such that $\mathbb{P}_0 = \varnothing$, $\mathbb{P}_t = \{z\}$, and for all $i \in [t]$ it holds that $\mathbb{P}_i$ can be obtained from $\mathbb{P}_{i-1}$ by applying exactly one of the above-stated rules. It is* one-shot *if each $v \in V$ is pebbled at most once.*

Finally, we mention the reversible pebble game introduced in [7]. In the reversible pebble game, the moves performed in reverse order should also constitute a legal black pebbling, which means that the rules for pebble placements and removals have to become symmetric.

▶ **Definition 2** (Reversible pebble game). *The* reversible pebble game *on G is the following one-player game: At any time i of the game, we have a pebble configuration $\mathbb{P}_i \subseteq V$. A pebble configuration $\mathbb{P}_{i-1}$ can be changed to $\mathbb{P}_i$ by applying exactly one of the following rules:*

Pebble placement on v: *Letting $\mathbb{P}_i = \mathbb{P}_{i-1} \cup \{v\}$ is allowed if $v \notin \mathbb{P}_{i-1}$ and $\text{pred}_G(v) \subseteq \mathbb{P}_{i-1}$. In particular, a pebble can always be placed on an empty source vertex.*

Reversible pebble removal from v: *Letting $\mathbb{P}_i = \mathbb{P}_{i-1} \setminus \{v\}$ is allowed if $v \in \mathbb{P}_{i-1}$ and $\text{pred}_G(v) \subseteq \mathbb{P}_{i-1}$. In particular, a pebble can always be removed from a source vertex.*

A reversible pebbling *of G is a sequence of pebble configurations $\mathcal{P} = (\mathbb{P}_0, \mathbb{P}_1, \ldots, \mathbb{P}_t)$ such that $\mathbb{P}_0 = \varnothing$, $\mathbb{P}_t = \{z\}$, and for all $i \in [t]$ it holds that $\mathbb{P}_i$ can be obtained from $\mathbb{P}_{i-1}$ by applying exactly one of the above-stated rules.*

▶ **Definition 3** (Time, space, and price of pebblings). *The* time *of a pebbling $\mathcal{P} = (\mathbb{P}_0, \mathbb{P}_1, \ldots, \mathbb{P}_t)$ is $\mathsf{time}(\mathcal{P}) := t$ and the* space *of it is $\mathsf{space}(\mathcal{P}) := \max_{i \in [t]} |\mathbb{P}_i|$. The* black pebbling price *(or* number*) of G, denoted by $\mathsf{Black}(G)$, is the minimum space of any black pebbling of G, whereas the* reversible pebbling price *of G, which we will denote by $\mathsf{Rev}(G)$, is the minimum space of any reversible pebbling of G.*

2.2 Resolution

A *literal* over a Boolean variable x is either x itself (also denoted as x^1) or its negation $\overline{x}$ (also denoted as x^0). A *clause* $C = a_1 \vee \cdots \vee a_\ell$ is a (possibly empty) disjunction of literals a_i over pairwise disjoint variables. The *set of variables occurring in a clause* C will be denoted by

$\mathrm{Vars}(C)$. A clause C is called *unit* if $|\mathrm{Vars}(C)| = 1$. We let $\square$ denote the contradictory *empty clause* (the clause without any literals). A *CNF formula* $F = C_1 \wedge \cdots \wedge C_m$ is a conjunction of clauses. It is often advantageous to think of clauses and CNF formulas as sets. The notion of the set of variables in a clause is extended to CNF formulas by taking unions. A CNF formula is a *k-CNF*, if all clauses in it have at most k variables. An *assignment/restriction* α for a CNF formula F is a function that maps some subset of $\mathrm{Vars}(F)$ to $\{0,1\}$. It is applied to F, which we denote by $F\restriction_\alpha$, in the usual way (see e. g. [6, 25]). We denote the *empty assignment* with $\varnothing$.

The standard definition of a *resolution derivation* of a clause D from a CNF formula F (denoted by $\pi : F \vdash D$) is an ordered sequence of clauses $\pi = (C_1, \ldots, C_t)$ such that $C_t = D$, and each clause C_i, for $i \in [t]$, is either an *axiom clause* $C_i \in F$, or is derived from clauses C_j and C_k, with $j, k < i$, by the *resolution rule* $\frac{B \vee x \quad C \vee \overline{x}}{B \vee C}$. In the resolution rule, we call $B \vee x$ and $C \vee \overline{x}$ the *parents* and $B \vee C$ the *resolvent*. A derivation $\pi : F \vdash \square$ of the empty clause from an unsatisfiable CNF formula F is called *refutation*. Note, that resolution is a sound and complete proof system for unsatisfiable formulas in CNF.

To study space in resolution, we consider the following definitions of the resolution proof system from [15, 1].

▶ **Definition 4** (Configuration-style resolution). *A* resolution refutation $\pi : F \vdash \square$ *of an unsatisfiable CNF formula F is an ordered sequence of* memory configurations *(sets of clauses) $\pi = (\mathbb{M}_0, \ldots, \mathbb{M}_t)$ such that $\mathbb{M}_0 = \varnothing$, $\square \in \mathbb{M}_t$ and for each $i \in [t]$, the configuration $\mathbb{M}_i$ is obtained from $\mathbb{M}_{i-1}$ by applying exactly one of the following rules:*

Axiom Download: *$\mathbb{M}_i = \mathbb{M}_{i-1} \cup \{C\}$ for some axiom clause $C \in F$.*

Erasure: *$\mathbb{M}_i = \mathbb{M}_{i-1} \setminus \{C\}$ for some $C \in \mathbb{M}_{i-1}$.*

Inference: *$\mathbb{M}_i = \mathbb{M}_{i-1} \cup \{D\}$ for some resolvent D inferred from $C_1, C_2 \in \mathbb{M}_i$ by the resolution rule.*

The proof π is said to be tree-like*, if we replace the inference rule with the following rule [15]:*

Tree-like Inference: *$\mathbb{M}_i = \big(\mathbb{M}_{i-1} \cup \{D\}\big) \setminus \{C_1, C_2\}$ for some resolvent D inferred from $C_1, C_2 \in \mathbb{M}_i$ by the resolution rule, i. e., we delete both parent clauses immediately.*

To every configurational refutation π we can associate a *refutation-DAG* G_π, with the clauses of the refutation labelling the vertices of the DAG and with edges from the parents to the resolvent for each application of the resolution rule. There might be several different derivations of a clause C during the course of the refutation, but if so, we can label each occurrence of C with a timestamp when it was derived and keep track of which copy of C is used where (cf. [20]). Using this representation, if π is tree-like, then G_π is a tree.

▶ **Definition 5** (Complexity measures for resolution). *The* length *of a resolution refutation $\pi = (\mathbb{M}_0, \ldots, \mathbb{M}_t)$ is defined to be $\mathrm{L}(\pi) := t$.*

The clause space *of a memory configuration $\mathbb{M}$ is defined as $\mathrm{CS}(\mathbb{M}) := |\mathbb{M}|$, i. e., the number of clauses in $\mathbb{M}$. The* variable space *of a memory configuration $\mathbb{M}$ is defined as $\mathrm{VS}(\mathbb{M}) := \big|\bigcup_{C \in \mathbb{M}} \mathrm{Vars}(C)\big|$, i.e., the number of distinct variables mentioned in $\mathbb{M}$.*

The clause space (variable space) of a refutation $\pi = (\mathbb{M}_0, \ldots, \mathbb{M}_t)$ is defined by $\mathrm{CS}(\pi) := \max_{i \in [t]} \mathrm{CS}(\mathbb{M}_i)$ and $\mathrm{VS}(\pi) := \max_{i \in [t]} \mathrm{VS}(\mathbb{M}_i)$, respectively.

Taking the minimum over all refutations of a formula F, we define $\mathrm{L}(F \vdash \square) := \min_{\pi : F \vdash \square} \mathrm{L}(\pi)$, $\mathrm{CS}(F \vdash \square) := \min_{\pi : F \vdash \square} \mathrm{CS}(\pi)$ and $\mathrm{VS}(F \vdash \square) := \min_{\pi : F \vdash \square} \mathrm{VS}(\pi)$ as the length, clause space and variable space of refuting F in resolution, respectively. We define $\text{Tree-CS}(F \vdash \square) := \min_{\pi' : F \vdash \square} \mathrm{CS}(\pi')$, where the minimum is taken over all tree-like refutations π' of the formula F.

▶ **Proposition 6** ([15]). *Let F be an unsatisfiable formula. Then it holds* $\mathrm{CS}(F \vdash \square) = \min_{\pi: F \vdash \square} \mathsf{Black}(G_\pi)$.

Razborov introduced *amortised space measures* for resolution in [24], that penalise configurational proofs for being unreasonably long.

▶ **Definition 7** (Amortised space measures for resolution). *The* amortised clause space *(*amortised variable space*) of a resolution refutation π is defined by* $\mathrm{CS}^*(\pi) := \mathrm{CS}(\pi) \cdot \log \mathrm{L}(\pi)$ *and* $\mathrm{VS}^*(\pi) := \mathrm{VS}(\pi) \cdot \log \mathrm{L}(\pi)$*, respectively.*

Taking the minimum over all refutations of a formula F, we define $\mathrm{CS}^*(F \vdash \square) := \min_{\pi: F \vdash \square} \mathrm{CS}^*(\pi)$ *and* $\mathrm{VS}^*(F \vdash \square) := \min_{\pi: F \vdash \square} \mathrm{VS}^*(\pi)$.

2.3 Formula Families

Pebbling Formulas and Their XORification

In the last years, there has been renewed interest in pebbling in the context of proof complexity. This is so, because pebbling results can be partially translated into proof complexity results by studying so-called pebbling formulas [6, 5]. These are unsatisfiable CNF formulas encoding the pebble game played on a DAG G. We define them next.

▶ **Definition 8** (Pebbling formulas). *Let $G = (V, E)$ be a DAG with a set of sources $S \subseteq V$ and a unique sink z. We identify every vertex $v \in V$ with a Boolean variable v. The* pebbling contradiction *over G, denoted* Peb_G*, is the conjunction of the following clauses:*

- *for all sources $s \in S$, a unit clause s,* (source axioms)
- *for all non-source vertices v, the clause $\bigvee_{u \in \mathrm{pred}_G(v)} \overline{u} \vee v$,* (pebbling axioms)
- *for the unique sink z, the unit clause $\overline{z}$.* (sink axiom)

Often, it turns out, that the formulas in Definition 8 are a bit too easy to refute. A good way to make them slightly harder is to substitute some suitable Boolean function $f(x_1, \ldots, x_d)$ of arity d for each variable x and expand the result into CNF. This general case is discussed in [20]. We restrict ourselves to the special case of the second degree XORification.

For notational convenience, we assume that the formula F we are trying to make harder only has variables x, y, z, et cetera, without subscripts, so that $x_1, x_2, y_1, y_2, z_1, z_2$, et cetera, are new variables not occurring in F.

▶ **Definition 9** (Substitution formulas, [4]). *For a positive literal x define the* XORification *of x to be $x[\oplus_2] := \{x_1 \vee x_2, \overline{x_1} \vee \overline{x_2}\}$. For a negative literal $\overline{y}$, the XORification is $\overline{y}[\oplus_2] := \{y_1 \vee \overline{y_2}, \overline{y_1} \vee y_2\}$. The XORification of a clause $C = a_1 \vee \cdots \vee a_k$ is the CNF formula*

$$C[\oplus_2] := \bigwedge_{C_1 \in a_1[\oplus_2]} \cdots \bigwedge_{C_k \in a_k[\oplus_2]} (C_1 \vee \cdots \vee C_k)$$

and the XORification of a CNF formula F is $F[\oplus_2] := \bigwedge_{C \in F} C[\oplus_2]$.

▶ Remark 10 ([4]). If G has n vertices and maximal in-degree ℓ, then $\mathrm{Peb}_G[\oplus_2]$ is an unsatisfiable $2(\ell+1)$-CNF formula with at most $2^{\ell+1} \cdot n$ clauses over $2n$ variables.

Tseitin Formulas

Tseitin formulas encode the combinatorial principle that for all graphs the sum of the degrees of the vertices is *even*. This class of formulas was introduced in [28] and has been extremely useful for the analysis of proof systems.

▶ **Definition 11** (Tseitin formulas). *Let $G = (V, E)$ be a connected undirected graph and let $\chi\colon V \to \{0,1\}$ be a* marking *of the vertices of G. A marking χ is called* odd *if it satisfies the property $\sum_{v\in V} \chi(v) \equiv 1 \pmod 2$ otherwise it is called* even*. Associate to every edge $e \in E$ a propositional variable e. The CNF formula $\mathrm{PARITY}_{v,\chi(v)}$ states that the parity of the values of the edges that have vertex v as endpoint coincides with $\chi(v)$, i. e.,*

$$\mathrm{PARITY}_{v,\chi(v)} := \bigwedge \left\{ \bigvee_{e\ni v} e^{a(e)} \,:\, a(e) \in \{0,1\}, \text{ such that } \bigoplus_{e\ni v} \big(a(e) \oplus 1\big) \not\equiv \chi(v) \right\}.$$

Then, the Tseitin formula *associated to the graph G and the marking χ is the CNF formula defined by $\mathrm{Ts}(G,\chi) := \bigwedge_{v\in V} \mathrm{PARITY}_{v,\chi(v)}$.*

For a partial truth assignment α, applying α to $\mathrm{Ts}(G,\chi)$ corresponds to the following simplification of the underlying graph: Setting a variable $e = \{u, v\}$ to 0 corresponds to deleting the edge e in the graph, and setting it to 1 corresponds to deleting the edge from the graph and toggling the value of $\chi(u)$ and $\chi(v)$ in G. We denote by $G{\restriction_\alpha}$ and by $\chi{\restriction_\alpha}$ the remaining graph and marking after applying α according to this process.

▶ **Fact 12** ([28, 29, 15]). *Let χ be an odd marking of of a connected graph G and e an edge in G that, when deleted divides G in two connected components G_1 and G_2. Then for $i \in \{1,2\}$ there is a partial assignment α_i of variable e so that $\chi{\restriction_{\alpha_i}}$ is an odd marking of G_i.*

2.4 Combinatorial Games for Tree-Like Clause Space in Resolution

Important tools for our results are two two-player combinatorial games. The Prover-Delayer game is played on *formulas* and was introduced in [22] in order to prove lower bounds for tree-like resolution length. Later it was shown in [16] that the game exactly characterises tree-like resolution space. The Raz–McKenzie game is played on *DAGs* and was introduced in [23] as a tool for studying the depth complexity of decision trees for search problems.

▶ **Definition 13** (Prover-Delayer game, [22, 16, 3]). *The* Prover-Delayer game *is a game between two players, called* Prover *(he), and* Delayer *(she), played on an unsatisfiable CNF formula F. The game is played in rounds. Each round starts with Prover querying the value of a variable. Delayer can give one of three answers: 0, 1, or $*$. If 0 or 1 is chosen by Delayer, no points are scored by her and the queried variable is set to the chosen bit. If Delayer answers $*$, then Prover gets to decide the value of that variable, and Delayer scores one point. The game finishes when any clause in F has been falsified by the partial assignment constructed this way. If this is not the case, the next round begins. The aim of Delayer is to win as many points as possible, while Prover aims to minimise this quantity.*

▶ **Definition 14** (Game value of the Prover-Delayer game). *Let F be an unsatisfiable CNF formula. The game value of the Prover-Delayer game played on F, denoted by $\mathsf{PD}(F)$, is the greatest number of points Delayer can score on F against an optimal strategy of Prover.*

The Prover-Delayer game exactly characterises the tree-like clause space of a formula. The constant term of the original result in [16, Theorem 2.2] was slightly modified to match our definitions of clause space and the pebble game (without sliding).

▶ **Theorem 15** ([16]). *If F is an unsatisfiable CNF formula,* $\text{Tree-CS}(F \vdash \square) = \mathsf{PD}(F) + 2$.

▶ **Definition 16** (Raz–McKenzie game). *The* Raz–McKenzie game *is played on a single-sink DAG G by two players,* Pebbler *and* Colourer*. The game is played in rounds. In the first round, Pebbler places a pebble on the sink and Colourer colours it red. In all subsequent*

rounds, Pebbler places a pebble on an arbitrary empty vertex of G and Colourer colours this new pebble either red or blue. The game ends when there is a vertex with a red pebble that is either a source vertex or all its direct predecessors in the graph have blue pebbles.

▶ **Definition 17** (Raz–McKenzie price)**.** *The* Raz–McKenzie price $\mathsf{R\text{-}Mc}(G)$ *of a single sink DAG G is the smallest number r such that Pebbler has a strategy to make the game end in at most r rounds against an optimal strategy of Colourer.*

▶ **Theorem 18** ([10])**.** *For any single-sink DAG G we have* $\mathsf{R\text{-}Mc}(G) = \mathsf{Rev}(G)$.

3 Separations Between Tree-Like and General Resolution Space for Pebbling Formulas Using the Raz–McKenzie Game

We will now establish a connection between tree-like clause space in resolution and the Raz–McKenzie price. We simplify the proof by following the intuition behind the game and identify the colour blue with 1 and the colour red with 0.

▶ **Theorem 19.** *For any single-sink DAG G it holds*

$$\mathsf{R\text{-}Mc}(G) + 2 \leq \text{Tree-CS}(\text{Peb}_G[\oplus_2] \vdash \square) \leq 2 \cdot \mathsf{R\text{-}Mc}(G) + 2.$$

Proof. Let G be a fixed DAG with a unique sink. We prove that $\mathsf{R\text{-}Mc}(G) \leq \mathsf{PD}(\text{Peb}_G[\oplus_2])$ and $\mathsf{PD}(\text{Peb}_G[\oplus_2]) \leq 2 \cdot \mathsf{R\text{-}Mc}(G)$. The result then follows from Theorem 15. We first show the inequality $\mathsf{PD}(\text{Peb}_G[\oplus_2]) \leq 2 \cdot \mathsf{R\text{-}Mc}(G) =: 2r$ by giving a strategy for Prover, such that Delayer can score at most $2r$ points. Prover basically simulates the strategy of Pebbler in the Raz–McKenzie game: If Pebbler pebbles a vertex v of G, Prover will query the variables v_1 and v_2 of $\text{Peb}_G[\oplus_2]$ in this order. The Raz–McKenzie game ends after at most r rounds. We will argue, that the Prover-Delayer game also ends after at most $2r$ queries. Thus, Delayer only gets a chance to score $2r$ points (if a variable pair gets queried for the first time, she can always answer $*$; only the second variable of the pair matters due to the XORification). In case the second variable of a pair gets queried, the best choice Delayer has is to follow the strategy of Colourer (Colourer is following an optimal strategy, thus, if Delayer had a better answer, this would correspond to a better answer for Colourer) and to ensure that $v_1 \oplus v_2$ is true under her constructed assignment if v is coloured 1; and false if v is coloured 0. At the end of the Raz–McKenzie game either a source vertex s in G is coloured 0, or a vertex v of G is coloured 0, while all its direct predecessors are coloured 1. In the first case, the source s being coloured 0 leads to the falsification of the corresponding source axiom $s[\oplus_2]$ by Delayer. In the second case, Delayer will falsify a clause of the corresponding pebbling axiom $\left(\bigwedge_{u \in \text{pred}_G(v)} \overline{u} \vee v\right)[\oplus_2]$.

Next, we show the inequality $\mathsf{PD}(\text{Peb}_G[\oplus_2]) \geq \mathsf{R\text{-}Mc}(G) =: r$ by giving a strategy for Delayer, such that under any strategy of Prover, she scores at least r points. By Definition 17, there is a strategy of Colourer, such that Pebbler has to pebble r vertices to end the game. Delayer will essentially copy this strategy: The first time a variable pair gets queried, she can answer $*$. The second time, she can copy the response of Colourer. Thus, she scores at least r points. ◀

From the equivalence between the Raz–McKenzie game and reversible pebbling we get:

▶ **Corollary 20.** *It holds* $\mathsf{Rev}(G) + 2 \leq \text{Tree-CS}(\text{Peb}_G[\oplus_2] \vdash \square) \leq 2 \cdot \mathsf{Rev}(G) + 2$ *for all DAGs G with a unique sink.*

The result that for any DAG G it holds $\mathrm{CS}(\mathrm{Peb}_G[\oplus_2] \vdash \square) = \mathrm{O}\big(\mathsf{Black}(G)\big)$ is considered as folklore (the idea behind it is that the pebbling formula can be resolved following the order in which the vertices of the graph are being pebbled). Combining this fact with Corollary 20, it follows that for any graph G with a gap between its black and reversible pebbling prices, the same separation can be obtained between the general and tree-like clause space of the corresponding pebbling formula. We mention some examples for which such a separation is known:

- The *path graphs*. Consider P_n to be a directed path with n vertices. Bennett [7] noticed, that these graphs provide a separation between black and reversible pebbling, proving that $\mathsf{Rev}(P_n) = \lceil \log n \rceil$. It was shown in [18], using a direct proof, that $\mathrm{CS}(\mathrm{Peb}_{P_n}[\oplus_2] \vdash \square) = \mathrm{O}(1)$ while $\text{Tree-CS}(\mathrm{Peb}_{P_n}[\oplus_2] \vdash \square) = \Theta(\log n)$.
- The *road graphs* from [11] provide a class of graphs for which the black pebbling price is non-constant and the reversible pebbling number is larger by a logarithmic factor.

▶ **Theorem 21** ([11]). *For any function $s(n) = \mathrm{O}\big(n^{1/2-\varepsilon}\big)$ with $0 < \varepsilon < \frac{1}{2}$ constant there is a family of DAGs $(G_n)_{n=1}^{\infty}$ of size $\Theta(n)$ with a single sink and maximal in-degree 2 such that $\mathsf{Black}(G_n) = \mathrm{O}\big(s(n)\big)$ and $\mathsf{Rev}(G_n) = \Omega\big(s(n) \log n\big)$.*

▶ **Corollary 22.** *For any function $s(n) = \mathrm{O}\big(n^{1/2-\varepsilon}\big)$ with $0 < \varepsilon < \frac{1}{2}$ constant there is a family of pebbling formulas $(\mathrm{Peb}_{G_n}[\oplus_2])_{n=1}^{\infty}$ with $\Theta(n)$ variables such that $\mathrm{CS}(\mathrm{Peb}_{G_n}[\oplus_2] \vdash \square) = \mathrm{O}\big(s(n)\big)$ and $\text{Tree-CS}(\mathrm{Peb}_{G_n}[\oplus_2] \vdash \square) = \Omega\big(s(n) \log n\big)$.*

The logarithmic factor in the number of vertices is almost the largest separation that can be obtained using this method since it is known that the reversible pebbling price can be upper bounded in terms of black pebbling space and time:

▶ **Theorem 23** ([19]). *If a DAG G has a black pebbling of time t and space s, the graph G has a reversible pebbling price of at most $s\lceil \log t \rceil$.*

By virtue of this result and Corollary 20 we obtain:

▶ **Corollary 24.** *For any DAG G with a unique sink vertex it holds*

$$\text{Tree-CS}(\mathrm{Peb}_G[\oplus_2] \vdash \square) = \mathrm{O}\left(\min_{\mathcal{P}} \big(\mathsf{space}(\mathcal{P}) \cdot \log \mathsf{time}(\mathcal{P})\big)\right),$$

where the minimum is taken over all black pebblings $\mathcal{P}$ of G.

This shows that the given separations cannot be improved for graphs for which the minimum black pebbling space is obtained with a one-shot strategy as it is the case for the path and road graphs, since the pebbling time for such a strategy is n. We present the first graph class for which the best pebbling strategy is not one-shot with a separation between black and reversible pebbling space. We do not obtain, however, any better separation than the $\log n$ factor obtained in the previous examples. We conjecture that this is in fact optimal. Our graphs $\hat{G}(c,k)$ are simplified versions of the original Carlson–Savage graphs [9]. Another adaptation of the original graphs is the family $\Gamma(c,r)$ studied in [20], for which an upper bound on the reversible pebble price was recently shown in [14]. We have simplified the graphs, eliminating the original pyramids since we are not analysing the black-white pebbling price, but our lower bound on reversible pebbling can be adapted to the original graphs or those in the family $\big(\Gamma(c,r)\big)_{c,r=1}^{\infty}$.

▶ **Definition 25** (Simplified Carlson–Savage graphs). *The class of DAGs $\big(G(c,k)\big)_{c,k=1}^{\infty}$ with parameters $c, k \geq 1$ is inductively defined in k. The base case $G(c,1)$ is the graph with one source node connected to c sink nodes. The graph $G(c,k+1)$ is composed of the graph*

$G(c,k)$ and c spines. A spine *is just a path of length $2c^2k$. The last node of each of the spines is a sink for $G(c,k+1)$. A spine is divided into* $2ck$ sections *of c consecutive vertices each. For each section and for each i with $1 \leq i \leq c$, there is an edge from the i-th sink of $G(c,k)$ to the i-th vertex in the section. In order to have single sink graphs, for $k \geq 2$ we also define $\hat{G}(c,k)$ exactly as $G(c,k)$ but with just one spine at the k-th level (all other levels have c spines). The last vertex of this spine is the only sink of $\hat{G}(c,k)$. For all c, the graph $\hat{G}(c,1)$ consists of just one edge.*

▶ **Lemma 26.** *The following claims hold:*

(i) *$\hat{G}(c,k)$ has $\Theta(c^3k^2)$ vertices,*

(ii) $\mathsf{Black}\big(\hat{G}(c,k)\big) \leq k+1$ *for any $c,k \geq 1$, while*

(iii) $\mathsf{Rev}\big(\hat{G}(c,k)\big) \geq \min\big\{c, (k-1)\log c + \log(k!)\big\}$ *for any $c,k \geq 1$.*

Proof. The first part follows easily by inductive counting.

For part (ii) of the lemma, we show inductively over k that any sink of $G(c,k)$ can be pebbled using $k+1$ pebbles. The result follows since $\hat{G}(c,k)$ is a subgraph of $G(c,k)$. The claim is trivial for $k=1$. For bigger values of k, the first vertex in any of the spines in $G(c,k)$ can be pebbled by placing a pebble on the corresponding sink of $G(c,k-1)$, removing all the pebbles except this one, and then pebbling the first vertex in the spine. The following strategy can be used for any other vertex v in the spine once its direct predecessor in the spine is pebbled: remove all the pebbles in the graph except the one on the direct spine predecessor of v, pebble the sink connected to v in $G(c,k-1)$, remove all the pebbles except the 2 on the direct predecessors of v, and then place a pebble on v. For this, by the induction hypothesis, at most $k+1$ pebbles are needed.

Part (iii) is more involved. We use the equivalence between reversible pebbling and the Raz–McKenzie game and show, also by induction over k, that the number of rounds to finish a game on $\hat{G}(c,k)$ starting from a configuration in which less than c vertices have been coloured blue, and no vertex in the unique spine of $\hat{G}(c,k)$ (except the sink) is coloured, is at least $\min\big\{c, (k-1)\log c + \log(k!)\big\}$. We give a strategy for Colourer obtaining this bound on the number of rounds. The base case is trivial. For $k \geq 2$, initially the only vertex coloured red is the unique sink of $\hat{G}(c,k)$. Let us denote the unique spine from $\hat{G}(c,k)$ as the *k-spine*. The game is divided in k stages (starting at stage k and finishing at stage 1). Stage k finishes when there is a blue vertex in the k-spine at a distance less than $2c$ from a red vertex. In stage k, if Colourer gives the colour red to a vertex v, this vertex has to be in the k-spine. If some vertex in $G(c,k-1)$ is queried by Pebbler, Colourer always answers with the blue colour. Because of this, the game cannot finish before the end of stage k. For simplicity we may assume that the first vertex of the k-spine has been coloured blue (for free, this can only make the strategy of Colourer harder), also for the clarity of exposition let us say that the k-spine is directed from left to right. The strategy of Colourer on the k-spine is to keep the gap between the rightmost blue vertex a (initially the initial node of the spine) and the leftmost red vertex b (initially the sink) as large as possible. That is, for any queried vertex v in the k-spine, if v lies at the left of a, it is coloured blue, if it is at the right of b it is coloured red and otherwise (i. e., if v is between a and b) if the distance from a to v is smaller that or equal to the distance from v to b, then v is coloured blue, otherwise it is coloured red. This strategy is followed by Colourer as long as the gap between a and b is at least $2c$. Once it is smaller than $2c$, stage k ends. If at this moment at least c vertices have been queried, there have been at least c rounds and the result follows. Otherwise there has to be a spine in $G(c,k-1)$ without any coloured vertex on it (there are c spines). Let us call t the sink of this spine and t' its rightmost uncoloured successor in the k-spine. We

can suppose that at this moment Colourer colours (for free) t, t' as well as all uncoloured vertices to the right of t' in the k-spine with colour red, and all the uncoloured vertices to the left of t' in the k-spine with blue. Again this only makes the strategy of Colourer harder since we are not counting these rounds. But now the game has been reduced to the instance of the graph $\hat{G}(c, k-1)$ containing the sink t. The number of rounds in stage k is at least $\log(\frac{2c^2k}{2c}) = \log c + \log k$ (this would happen with a binary search strategy of Pebbler on the k-spine). If in all the stages less than c vertices are queried, by induction, the rounds to finish the game on $\hat{G}(c, k-1)$ are at least $(k-2)\log c + \log\big((k-1)!\big)$. Adding these rounds to those from stage k we get the result. ◀

▶ **Theorem 27.** *For any function $s(n) = \Theta\big(n^{1/5-\varepsilon}\big)$ with $0 < \varepsilon < \frac{1}{5}$ constant there is a family of pebbling formulas $(\mathrm{Peb}_{G_n}[\oplus_2])_{n=1}^{\infty}$ with $\mathrm{O}(n)$ variables such that $\mathrm{CS}(\mathrm{Peb}_{G_n}[\oplus_2] \vdash \square) = \mathrm{O}\big(s(n)\big)$ and $\text{Tree-CS}(\mathrm{Peb}_{G_n}[\oplus_2] \vdash \square) = \Omega\big(s(n)\log n\big)$, and the best strategy for pebbling the graphs G_n is not one-shot.*

Proof. We show that for any such function s there is a graph family $\big(\hat{G}\big(c(n), \lceil s(n)\rceil\big)\big)_{n=1}^{\infty}$ with the corresponding gap between its black and reversible pebbling prices. The result follows from Corollary 20.

Given any such space function $s(n) = \Theta\big(n^{1/5-\varepsilon}\big)$ with $0 < \varepsilon < \frac{1}{5}$ constant, we define $c(n) := \lceil s(n) \cdot \log n\rceil$. This allows us to consider the graphs $\hat{G}\big(c(n), \lceil s(n)\rceil\big)$. By Lemma 26 (i), this graph has $\mathrm{O}\big(c(n)^3 \cdot \lceil s(n)\rceil^2\big) = \mathrm{O}\big(s(n)^5 \cdot \log^3 n\big) = \mathrm{O}\big(n^{1-5\varepsilon} \cdot \log^3 n\big) = \mathrm{O}(n)$ vertices. By Lemma 26 (ii), the graph has a black pebbling number upper bounded by $\lceil s(n)\rceil + 1 = \mathrm{O}(s(n))$. It only remains to show, that the reversible pebbling number of the graph is asymptotically lower bounded by $s(n)\log n$. For this, we consider two cases.

Case 1: $\min\Big\{c(n), \big(s(n)-1\big)\log c(n) + \log\big(s(n)!\big)\Big\} = c(n)$. In this case, Lemma 26 (iii) implies, that the reversible pebbling number of the graph is lower bounded by $c(n)$, which, by definition, is greater than or equal to $s(n)\log n$.

Case 2: $\min\Big\{c(n), \big(s(n)-1\big)\log c(n) + \log\big(s(n)!\big)\Big\} = \big(s(n)-1\big)\log c(n) + \log\big(s(n)!\big)$. In this case, one can notice, that already the first term, i. e., $\big(s(n)-1\big)\log c(n)$ is in

$$\begin{aligned}\Omega\Big(\big(s(n)-1\big)\log\big(s(n)\log n\big)\Big) &= \Omega\Big(\big(s(n)-1\big)\log\big(s(n)\big) + \big(s(n)-1\big)\log\log n\Big)\\ &= \Omega\Big(\big(s(n)-1\big)\big(1/5-\varepsilon\big)\log n + \big(s(n)-1\big)\log\log n\Big) = \Omega\big(s(n)\log n\big).\end{aligned}$$

◀

4 Upper Bounds for Tree-CS for General Formulas

Next, we provide generalisations of Corollary 24 for general formulas.

▶ **Theorem 28.** *For any unsatisfiable formula F it holds*

$$\text{Tree-CS}(F \vdash \square) \le \mathrm{VS}^*(F \vdash \square) + 2 = \min_{\pi : F \vdash \square}\big(\mathrm{VS}(\pi)\cdot\log \mathrm{L}(\pi)\big) + 2.$$

Proof of Theorem 28. Consider a configurational refutation $\pi = (\mathbb{M}_0, \ldots, \mathbb{M}_t)$ of F. Let α be the current partial assignment constructed in the Prover-Delayer game played on the formula F. At the beginning we have $\alpha = \varnothing$. We give a strategy for Prover that allows him to finish the game with at most $\mathrm{VS}(\pi)\cdot\log \mathrm{L}(\pi)$ points scored by Delayer regardless of her answers. The strategy of Prover proceeds in *bisection steps* indexed with k. Prover keeps as an invariant in these steps an interval $I_k = [a_k, b_k] \subseteq [0, t]$ such that $\pi_{[a_k,b_k]\restriction\alpha} := \big(\mathbb{M}_{a_k}\restriction\alpha, \ldots, \mathbb{M}_{b_k}\restriction\alpha\big)$ is a

configurational refutation of $F\restriction_\alpha$ for all k. Initially, $I_0 := [0,t]$, thus $\pi_{[0,t]}\restriction_\varnothing = \pi$ is obviously a refutation of $F\restriction_\varnothing = F$. In each bisection step, Prover starts querying the variables present in the configuration $\mathbb{M}_{m_k}$, with $m_k = \lfloor \frac{a_k+b_k}{2} \rfloor$, that have not been assigned yet, in any order. If Delayer answers $*$ to some variable, Prover will assign 0 to it (actually, Prover could assign any value). In this way α is extended to all the variables in the configuration $\mathbb{M}_{m_k}$. Prover then proceeds according to the following cases:

- **(i)** If after the assignment to the queried variables, a clause in the configuration $\mathbb{M}_{m_k}$ is falsified, Prover continues with the upper half of the proof (i. e., he sets $I_{k+1} = [a_{k+1}, b_{k+1}] := [a_k, m_k]$) and proceeds with the next bisection step.
- **(ii)** If after the assignment to the queried variables, all the clauses in $\mathbb{M}_{m_k}$ are satisfied, Prover continues with the lower half of the proof (i. e., he sets $I_{k+1} = [a_{k+1}, b_{k+1}] := [m_k, b_k]$) and proceeds with the next bisection step.

Prover queries at most $\mathrm{VS}(\pi)$ variables in each bisection step. It remains to show that the invariant is indeed kept and that Prover wins the game by following this strategy.

First, we show inductively, that the invariant is kept. In case (i) this is true by following the Resolution Restriction Lemma (see e.g. [25]) because $\mathbb{M}_{b_{k+1}}\restriction_\alpha = \mathbb{M}_{m_k}\restriction_\alpha$ contains the empty clause and thus $(\mathbb{M}_{a_{k+1}}\restriction_\alpha, \ldots, \mathbb{M}_{m_k}\restriction_\alpha)$ is a configurational refutation of $F\restriction_\alpha$. In case (ii) we have $\mathbb{M}_{m_k}\restriction_\alpha = \varnothing$ and $\mathbb{M}_{b_k}\restriction_\alpha \ni \square$ by the induction hypothesis, yet π was a refutation for F. Hence, for $i \in (a_k, b_k)$ the axioms contained in the memory configurations $\mathbb{M}_i\restriction_\alpha$ must be downloaded from $F\restriction_\alpha$. Thus, $(\mathbb{M}_{a_{k+1}}\restriction_\alpha, \ldots, \mathbb{M}_{b_{k+1}}\restriction_\alpha)$ is a legal refutation of $F\restriction_\alpha$.

Prover has to win the game since for every k, the formula $F\restriction_\alpha$ has a configurational refutation, namely $\pi_{I_k}\restriction_\alpha$, of length upper bounded by $\frac{1}{2}\mathrm{L}(\pi_{I_{k-1}})$. The strategy proceeds until $F\restriction_\alpha$ has a configurational refutation of length 1. Then, $\square \in F\restriction_\alpha$. In other words, the constructed assignment α falsifies a clause in F and Prover wins the game.

Summarising, Prover queries at most $\mathrm{VS}(\pi)$ variables in each bisection step and since there are at most $\lceil \log \mathrm{L}(\pi) \rceil$ configurations that get queried, Prover in total queries at most $\mathrm{VS}(\pi) \cdot \log \mathrm{L}(\pi)$ variables. Theorem 15 yields the desired inequality. ◀

We prove now that Theorem 28 also works for clause space. For this, we show that the tree-like clause space of a formula F is always upper bounded by the reversible pebble game played on a refutation of F. Note, that the minimum in the theorem is taken over all possible refutations of F, not only over the tree-like ones. The inequality in Theorem 29 works only in one direction. For example the formula with a clause with n negated variables and n unit clauses containing one of the variables each, has constant tree-resolution space while the reversible pebbling price for any refutation graph is at least $\log n$.

▶ **Theorem 29.** *For any unsatisfiable formula F with n variables it holds*

$$\text{Tree-CS}(F \vdash \square) \leq \min_{\pi : F \vdash \square} \mathsf{Rev}(G_\pi) + 2, \quad \textit{and}$$
$$\min_{\pi : F \vdash \square} \mathsf{Rev}(G_\pi) \leq \text{Tree-CS}(F \vdash \square)\big(\lceil \log n \rceil + 1\big).$$

Proof. Let F be an unsatisfiable formula with n variables.

For proving the first inequality, let π be a resolution refutation of F with a refutation-graph G_π and $\mathsf{Rev}(G_\pi) =: k$. We will use Theorem 15, as well as Theorem 18 applied to G_π: It suffices to give a strategy for Prover in the Prover-Delayer game played on F under which he has to pay at most k points. Prover basically simulates the strategy of Pebbler in the Raz–McKenzie game played on G_π, which coincides with reversible pebbling. By doing so, a partial assignment α falsifying an initial clause of F will be produced. The game is divided

in stages. Initially the partial assignment is the empty assignment. In each stage, if Pebbler chooses a clause $C \in V(G_\pi)$, Prover queries the variables in C not yet assigned by α one by one, extending the partial assignment α with the answers of Delayer, until either:

(i) the clause C is satisfied or falsified by α, or

(ii) a variable x in C is given value $*$ by Delayer.

In case (i), Prover moves to the next stage, simulating the strategy of Pebbler assuming Colourer has given clause C the colour $C\restriction_\alpha$. In case (ii), Prover extends α by assigning x with the value that satisfies C and moves to the next stage, simulating the strategy of Pebbler, assuming Colourer has given clause C the colour 1. The game is played until α falsifies a clause in F. After at most k stages the Raz–McKenzie games finishes and therefore Delayer can score at most k points. It is only left to show that at the end of the game a clause in F is falsified by α. When the Raz–McKenzie game finishes, either a source in G_π is assigned colour 0 by Colourer, or a vertex with all its direct predecessors being coloured 1 is coloured 0. Since α defines Colourer's answers, the first situation corresponds to α falsifying a clause in F. The second situation is not possible since for any partial assignment α it cannot be that α satisfies two parent clauses in a resolution proof, while falsifying their resolvent.

For the proof of the second inequality, let $k := \text{Tree-CS}(F \vdash \square)$. By Proposition 6, we know that there is a refutation π of F whose underlying graph G_π is a tree with black pebbling price k. We can suppose that the refutation is *regular*, that is, in every path from the empty clause to a clause in F in the refutation tree, each variable is resolved at most once [15, Theorem 5.1]. This implies that the depth of the tree is at most n. For any node v in the refutation tree let T_v be the subtree of G_π rooted at v. For the sake of convenience, we refer to $\mathsf{Black}(T_v)$ as the *pebbling number of* v.

We show by induction on κ that for any vertex v in G_π, if $\mathsf{Black}(T_v) = \kappa$ then there is a strategy for Pebbler in the Raz–McKenzie game on T_v with most $\kappa(\lceil \log n \rceil + 1)$ rounds. For the base case $\kappa = 1$, the vertex v must be a leaf node and the game needs only one round. For $\kappa > 1$, the game starts, according to the rules, by Pebbler querying the root v of the subtree and Colourer answering 0. We consider two cases, depending on whether for both predecessors v_1 and v_2 of v in G_π, $\mathsf{Black}(T_{v_1}) = \mathsf{Black}(T_{v_2}) = \kappa - 1$ or not. In the former case, Pebbler queries one of them, say v_1. If the answer is 0, he continues on T_{v_1} and otherwise continues on T_{v_2}. By induction, the number of rounds in this case is at most $2 + (\kappa - 1)(\lceil \log n \rceil + 1) \leq \kappa(\lceil \log n \rceil + 1)$. In case, it is not true, that $\mathsf{Black}(T_{v_1}) = \mathsf{Black}(T_{v_2}) = \kappa - 1$, since $\mathsf{Black}(T_v) = \kappa$, and G_π is a tree, one of the trees T_{v_1} or T_{v_2} leading to v must have pebbling number κ and the other one must have pebbling number smaller than κ. Pebbler considers the path of nodes starting at v and going towards the leaves, having all the nodes in the path pebbling number κ, until a node u is reached, for which both predecessors have pebbling number $\kappa - 1$. Such a node u must exist because G_π is a tree. Let u_1 be one of the predecessors of u. The length of the path from v to u_1 is at most n since the refutation is regular. Pebbler queries the vertices in the path between v and u_1 with binary search, until a vertex t is found that is coloured with colour 0 by Colourer, while its predecessor in the path $v \rightsquigarrow u_1$ has been coloured 1. At this point, Pebbler continues playing the game on the tree rooted at the uncoloured predecessor of t. It is also possible that all the queried nodes in the path from v to u_1 (including u_1) are coloured 0 by Colourer. In this case Pebbler continues with T_{u_1}. In all situations at most $1 + \lceil \log n \rceil$ vertices have been queried and the game has been reduced to a subgraph with smaller pebbling number. ◀

▶ **Corollary 30.** *For any unsatisfiable formula F it holds*

$$\text{Tree-CS}(F \vdash \square) \leq \text{CS}^*(F \vdash \square) + 2 = \min_{\pi : F \vdash \square} \big(\text{CS}(\pi) \cdot \log \text{L}(\pi)\big) + 2.$$

Proof. By Theorem 23, $\min_{\pi:F\vdash\square}\mathsf{Rev}(G_\pi)+2\le\min_{\mathcal{P}}\big(\mathsf{space}(\mathcal{P})\cdot\log\mathsf{time}(\mathcal{P})\big)+2$, where the minimum is taken over all black pebblings $\mathcal{P}$ of G_π. The result follows with (a slight adaption of) Proposition 6 since every black pebbling $\mathcal{P}$ of G_π defines a configurational refutation of F with clause space equal to $\mathsf{space}(\mathcal{P})$ and length $\mathsf{time}(\mathcal{P})$. ◀

5 Optimal Separations for Tseitin Formulas

In this section, we prove optimal separations between tree-like clause space and variable space, as well as clause space in the context of Tseitin formulas. This complements the relations between clause space and variable space of Tseitin formulas recently given in [17].

▶ **Theorem 31.** *For any connected graph G with n vertices and odd marking χ we have*

$$\text{Tree-CS}\big(\mathrm{Ts}(G,\chi)\vdash\square\big)\le\mathrm{CS}\big(\mathrm{Ts}(G,\chi)\vdash\square\big)\cdot\log n+2,\quad \textit{and}$$
$$\text{Tree-CS}\big(\mathrm{Ts}(G,\chi)\vdash\square\big)\le\mathrm{VS}\big(\mathrm{Ts}(G,\chi)\vdash\square\big)\cdot\log n+2.$$

Proof. The proof is based on the one for the lower bound for CS of Tseitin formulas from [27]. Let $G=(V,E)$ be a connected graph with n vertices, χ an odd marking, and $\pi=(\mathbb{M}_0,\dots,\mathbb{M}_t)$ a refutation of $\mathrm{Ts}(G,\chi)$ with $\mathrm{CS}(\pi)=:k$. We use π to give a strategy for Prover in the Prover-Delayer game for which he has to pay at most $k\log n$ points. We say that a partial assignment α of some of the variables in $\mathrm{Ts}(G,\chi)$ is *non-splitting* if after applying α to the formula, the resulting graph still has a connected component with an odd marking (*odd component*) of size at least $\lceil\frac{|V|}{2}\rceil$, and the rest are components with even markings. Consider the last configuration $\mathbb{M}_s$ in π for which there is a partial assignment α fulfilling:

(i) α simultaneously satisfies all clauses in $\mathbb{M}_s$, and
(ii) α is non-splitting.

This stage must exist since before the initial step the empty truth assignment is trivially a non-splitting partial assignment satisfying the clauses in $\mathbb{M}_0=\varnothing$. At the end, the last configuration $\mathbb{M}_t$ in the refutation contains the empty clause which cannot be satisfied by any assignment. Thus, stage s must exist in between.

The step from s to $s+1$ was no deletion step (otherwise this would be a contradiction to the maximality of s). The only new clause in $\mathbb{M}_{s+1}$ must be an axiom C of $\mathrm{Ts}(G,\chi)$ since any other clause that could be added to the list of clauses in memory at stage $s+1$ would be a resolvent of two clauses from stage s, but in this case any partial assignment satisfying the clauses at stage s would also satisfy those at $s+1$. For some vertex v in G, this axiom clause C introduced at stage $s+1$ belongs to the formula $\mathrm{PARITY}_{v,\chi(v)}$. Let α be a partial assignment of minimal size satisfying the conditions at stage s. It is possible to extend α to satisfy the clause $C{\restriction_\alpha}$ since v either belongs to an even component in $(G{\restriction_\alpha},\chi{\restriction_\alpha})$ or to the large odd component in this graph and therefore $C{\restriction_\alpha}\neq\square$. Because of this, vertex v must belong to the unique odd component since otherwise α could be extended in a non-splitting way.

Let $C{\restriction_\alpha}=(\ell_1,\dots,\ell_m)$, $m\ge1$, where the ℓ_i's are literals corresponding to the edges with endpoint v in $G{\restriction_\alpha}$. Observe that deleting any of these edges e_i in $C{\restriction_\alpha}$ cuts the connected component of v in two pieces because otherwise assigning any value to the corresponding edge would not modify the size of the connected components in $C{\restriction_\alpha}$ and there would be a non-splitting way to extend α to e satisfying C. Also, any component remaining after assigning all the literals in $C{\restriction_\alpha}$ must have size at most $\lfloor\frac{|V|}{2}\rfloor$ since otherwise there would be a way to extend α satisfying C and producing an odd marking for the largest such component (Fact 12), and this extension would be non-splitting.

The strategy of Prover is to query the variables assigned in α thus paying at most $k-1$ points and obtaining a partial assignment γ from Delayer. If at this point one of the connected components of size at most $\lfloor \frac{|V|}{2} \rfloor$ is odd then Prover moves to this component and starts playing the game on it. Otherwise Prover queries the variables in $C\restriction_\gamma$ one by one. If for a variable e_i Delayer answers with $*$, Prover just has to assign e_i so that the smallest of the two components that appear in $G\restriction_\gamma$ after assigning e_i is odd (not necessarily satisfying $C\restriction_\gamma$). This is always possible because of Fact 12. If no $*$ is answered, Prover queries the next variable until no variable in $C\restriction_\gamma$ is left. Let γ' be the assignment obtained this way. Either γ' falsifies C and the game ends, or it satisfies the clause and in this case all the components (odd or even) remaining after applying γ' have size at most $\lfloor \frac{|V|}{2} \rfloor$. In every case, after applying γ' Prover wins the game or there is an odd connected component of size at most half as large as the initial graph. The original problem has been reduced to another in a graph with at most $\frac{n}{2}$ many vertices. Also Prover has to pay at most k for obtaining γ'. After repeating this process at most $\log n$ times, an initial clause is falsified.

The second part of the theorem is a little simpler and follows by considering a configurational proof π of variable space k. Everything in the above proof works in the same way, observing that the partial assignment α satisfying all clauses in memory at stage s, when extended to all the variables in the new clause at stage $s+1$ needs to assign at most k variables (all those included in the configuration) and is either splitting or falsifies the axiom. Observe that this implies that in every configurational proof there is a point in which every assignment to the variables in the configuration is spliting. ◀

Next, we show, that the upper bounds in Theorem 31 are tight by proving that there is a family of Tseitin formulas that provide matching lower bounds. These are formulas corresponding to grid graphs with constant width, which can be considered as the Tseitin version of path graphs.

▶ **Definition 32** (Grid graphs). *For a natural number $\ell \geq 1$, the* grid graphs $G_{2\times\ell}$ *are given by the vertex set $V(G_{2\times\ell}) := [2] \times [\ell]$ and the edge set*

$$E(G_{2\times\ell}) := \Big\{ \big\{(i,j),(i',j')\big\} \,:\, i,i' \in [2],\ j,j' \in [\ell],\ \text{and } |i-i'|+|j-j'|=1 \Big\}.$$

▶ **Theorem 33.** *For the family of Tseitin formulas $\big(\mathrm{Ts}(G_{2\times\ell},\chi_\ell)\big)_{\ell=1}^{\infty}$ with $3\ell-2$ variables it holds* Tree-CS$\big(\mathrm{Ts}(G_\ell,\chi_\ell)\vdash\square\big) = \Theta(\log\ell)$, *and* CS$\big(\mathrm{Ts}(G_\ell,\chi_\ell)\vdash\square\big) = \mathrm{O}(1)$, *as well as* VS$\big(\mathrm{Ts}(G_\ell,\chi_\ell)\vdash\square\big) = \mathrm{O}(1)$.

Proof. To show the lower bound on tree-like clause space with Theorem 15, we give a strategy for Delayer such that he scores $\Omega(\log\ell)$ points playing on $G_{2\times\ell}$. In the following, for a subgraph G' of $G_{2\times\ell}$, we define $\mathrm{Block}(G') := \max\big\{b\in\mathbb{N} \,:\, \text{there is a subgraph of } G' \text{ that is isomorphic to } G_{2\times b}\big\}$. The strategy of Delayer is as follows:

(a) If an edge e in an even component is queried, Delayer should answer according to some assignment satisfying this component.

(b) If an edge e in an odd component is queried, Delayer proceed as follows:

- **(i)** If the deletion of e does not increase the number of connected components in G, Delayer should answer $*$.
- **(ii)** If the deletion of e cuts the graph and both endpoints of e are separated in different connected components, Delayer should answer in a way, that from these two components, the component G' with largest $\mathrm{Block}(G')$ receives the odd marking.

At the beginning of the Prover-Delayer game we have $\mathrm{Block}(G_{2\times\ell}) = \ell$. After each assignment of a variable in the game we have $\mathrm{Block}(G') \geq \lfloor \frac{1}{2}\,\mathrm{Block}(G) \rfloor$, where we let G denote the underlying graph before the assignment and G' the graph after the assignment: Notice, that

rule (b)(ii) guarantees that the component with the largest Block-value always receives an odd marking. If Delayer plays according to this strategy, we must have $\mathrm{Block}(G) = 0$ at the beginning of some round. This means that the Block-value, starting the game with $G_{2\times\ell}$, has to change at least $\Omega(\log \ell)$ times before the game can end. It is easy to see, that if the Block-value changes in a step, the number of connected components does not increase in this step. According to rule (b) (i), Delayer has answered $*$ in this round and has scored a point.

For the second part, consider the variables (edges) ordered (from left to right) with $\big\{(1,j),(2,j)\big\} \prec \big\{(1,j),(1,j+1)\big\} \prec \big\{(2,j),(2,j+1)\big\}$ and edges with lower j defined to be smaller (with respect to $\prec$) than those with higher j for $1 \leq j \leq \ell - 1$, and consider a resolution refutation completely resolving the variables in decreasing order (from right to left). That is, the clauses containing variable $\big\{(1,\ell),(2,\ell)\big\}$ will be first resolved with all clauses containing this variable in negated form (in case it is possible to resolve), and so on. Since the graph has degree at most 3, there is a small number of clauses containing this variable. Also observe that after resolving the last three variables in the ordering in this way, the set of derived clauses plus the initial clauses contain a subset of clauses encoding the formula $\mathrm{Ts}(G_{2\times(\ell-1)}, \chi')$ for some odd marking χ'. The set of newly derived clauses in this subset has constant size, and the number of clauses in all the resolution configurations until this point is also constant. Continuing in this order with the complete resolution of all the variables, we obtain a refutation of $\mathrm{Ts}(G_{2\times\ell}, \chi)$ with constant clause and variable space. ◀

6 Conclusions and Open Problems

By introducing a new connection between tree-like resolution space and the reversible pebble game, we have studied the relation between tree-like space and space measures for general resolution, obtaining almost optimal separations between these measures. We conjecture that these separations are optimal and that in fact, the $\log\big(\mathrm{time}(\pi)\big)$ factors in the upper bounds of Theorems 23 and 28 and Corollaries 24 and 30 can be improved to a $\log n$ factor (n being the number of graph vertices or formula size, depending on the setting). We have been able to prove this for the restricted case of the Tseitin contradictions.

We have seen that a source for obtaining space separations between tree-like and general resolution are graph classes with a gap between their reversible and black pebbling prices and we have provided a new class of such graphs. An interesting question is whether there exists a graph class with such a separation for a space function larger than $n^{1/2}$.

References

1 Michael Alekhnovich, Eli Ben-Sasson, Alexander A. Razborov, and Avi Wigderson. Space complexity in propositional calculus. *SIAM Journal on Computing*, 31(4):1184–1211, 2002. Preliminary version in *STOC '00*.

2 Carlos Ansótegui, María Luisa Bonet, Jordi Levy, and Felip Manyà. Measuring the hardness of SAT instances. In *Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI '08)*, pages 222–228, July 2008.

3 Eli Ben-Sasson, Russell Impagliazzo, and Avi Wigderson. Near optimal separation of tree-like and general resolution. *Combinatorica*, 24(4):585–603, September 2004.

4 Eli Ben-Sasson and Jakob Nordström. Short proofs may be spacious: An optimal separation of space and length in resolution. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS '08)*, pages 709–718, October 2008.

5 Eli Ben-Sasson and Jakob Nordström. Understanding space in proof complexity: Separations and trade-offs via substitutions. In *Proceedings of the 2nd Symposium on Innovations in Computer Science (ICS '11)*, pages 401–416, January 2011. Full-length version available at `http://eccc.hpi-web.de/report/2010/125/`.

6 Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow—resolution made simple. *Journal of the ACM*, 48(2):149–169, March 2001. Preliminary version in *STOC '99*.

7 Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM Journal on Computing*, 18(4):766–776, August 1989.

8 María Luisa Bonet, Juan Luis Esteban, Nicola Galesi, and Jan Johannsen. Exponential separations between restricted resolution and cutting planes proof systems. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS '98)*, pages 638–647, November 1998.

9 David A. Carlson and John E. Savage. Extreme time-space tradeoffs for graphs with small space requirements. *Information Processing Letters*, 14(5):223–227, 1982.

10 Siu Man Chan. Just a pebble game. In *Proceedings of the 28th Annual IEEE Conference on Computational Complexity (CCC '13)*, pages 133–143, June 2013.

11 Siu Man Chan, Massimo Lauria, Jakob Nordström, and Marc Vinyals. Hardness of approximation in PSPACE and separation results for pebble games (Extended abstract). In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS '15)*, pages 466–485, October 2015. Full-length version in [30].

12 Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.

13 Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

14 Susanna F. de Rezende. *Lower Bounds and Trade-offs in Proof Complexity*. PhD thesis, KTH Royal Institute of Technology, June 2019. Available at `http://kth.diva-portal.org/smash/record.jsf?pid=diva2%3A1318061&dswid=-4968`.

15 Juan Luis Esteban and Jacobo Torán. Space bounds for resolution. *Information and Computation*, 171(1):84–97, 2001. Preliminary versions of these results appeared in *STACS '99* and *CSL '99*.

16 Juan Luis Esteban and Jacobo Torán. A combinatorial characterization of treelike resolution space. *Information Processing Letters*, 87(6):295–300, 2003.

17 Nicola Galesi, Navid Talebanfard, and Jacobo Torán. Cops-robber games and the resolution of Tseitin formulas. In *Proceedings of the 21th International Conference on Theory and Applications of Satisfiability Testing (SAT '18)*, volume 10929 of *Lecture Notes in Computer Science*, pages 311–326. Springer, 2018.

18 Matti Järvisalo, Arie Matsliah, Jakob Nordström, and Stanislav Živný. Relating proof complexity measures and practical hardness of SAT. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP '12)*, volume 7514 of *Lecture Notes in Computer Science*, pages 316–331. Springer, October 2012.

19 Richard Královič. Time and space complexity of reversible pebbling. *RAIRO – Theoretical Informatics and Applications*, 38(02):137–161, April 2004.

20 Jakob Nordström. New wine into old wineskins: A survey of some pebbling classics with supplemental results. Manuscript in preparation. Current draft version available at `http://www.csc.kth.se/~jakobn/research/PebblingSurveyTMP.pdf`, 2015.

21 Michael S. Paterson and Carl E. Hewitt. Comparative schematology. In *Record of the Project MAC Conference on Concurrent Systems and Parallel Computation*, pages 119–127, 1970.

22 Pavel Pudlák and Russell Impagliazzo. A lower bound for DLL algorithms for k-SAT (preliminary version). In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '00)*, pages 128–136, January 2000.

23 Ran Raz and Pierre McKenzie. Separation of the monotone NC hierarchy. *Combinatorica*, 19(3):403–435, March 1999. Preliminary version in *FOCS '97*.

24 Alexander A. Razborov. On space and depth in resolution. *Computational Complexity*, 27(3):511–559, 2018.

25 Uwe Schöning and Jacobo Torán. *The Satisfiability Problem: Algorithms and Analyses*, volume 3 of *Mathematics for Applications (Mathematik für Anwendungen)*. Lehmanns Media, 2013.

26 Jacobo Torán and Florian Wörz. Reversible pebble games and the relation between tree-like and general resolution space. Technical Report TR19-097, Electronic Colloquium on Computational Complexity (ECCC), 2019. URL: `https://eccc.weizmann.ac.il/report/2019/097`.

27 Jacobo Torán. Lower bounds for space in resolution. In *Proceedings of the 13th International Workshop on Computer Science Logic (CSL '99)*, volume 1683 of *Lecture Notes in Computer Science*, pages 362–373. Springer, 1999.

28 Grigori Tseitin. The complexity of a deduction in the propositional predicate calculus. *Zapiski Nauchnyh Seminarov Leningradskogo Otdelenija matematicheskogo Instituta im. V. A. Steklova akademii Nauk SSSR (LOMI)*, 8:234–259, 1968. In Russian.

29 Alasdair Urquhart. Hard examples for resolution. *Journal of the ACM*, 34(1):209–219, January 1987.

30 Marc Vinyals. *Space in Proof Complexity*. PhD thesis, KTH Royal Institute of Technology, May 2017. Available at `http://www.csc.kth.se/~jakobn/project-proofcplx/docs/MV_PhDthesis.pdf`.