38th International Symposium on Theoretical Aspects of Computer Science

STACS 2021, March 16–19, 2021, Saarbrücken, Germany
(Virtual Conference)

Edited by
Markus Bläser
Benjamin Monmege

LIPIcs – Vol. 187 – STACS 2021 www.dagstuhl.de/lipics

Editors

Markus Bläser
Saarland University, Germany
mblaeser@cs.uni-saarland.de

Benjamin Monmege
Aix-Marseille University, France
benjamin.monmege@univ-amu.fr

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 978-3-95977-180-1

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-180-1.

Publication date
March, 2021

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.2021.0

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

Invited Talks

Regular Papers

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege
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 virtually from March 16 to 19 in Saarbrücken) is the 38th 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), and Montpellier (2020).

The interest in STACS has remained at a very high level over the past years. The STACS 2021 call for papers led to 228 submissions with authors from 37 countries. Each paper was assigned to three program committee members who, at their discretion, asked external reviewers for reports. For the seventh 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. In addition, STACS 2021 employed a lightweight double-blind reviewing process for the first time: submissions should not reveal the identity of the authors in any way. However, it was was still possible for authors to disseminate their ideas or draft versions of their paper as they normally would, for instance by posting drafts on the web or giving talks on their results. The committee selected 56 papers during a four-week electronic meeting held in November and December 2020. This means an acceptance rate below 25%. As co-chairs of the program committee, we would like to sincerely thank all its members and the 408 external reviewers for their valuable work. In particular, there were intense and interesting discussions inside the PC committee. The 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: Peter Bürgisser (TU Berlin, Germany), Patrice Ossona de Mendez (CAMS, Paris, France), and Lidia Tendera (Opole University, Poland).

STACS 2020 in Montpellier was one of the last conferences that took place physically before the lockdown happened the next week. We very much hoped that STACS 2021 would be one of the first conferences that takes place physically again, with an option of remote participation for participants who could not come to Saarbrücken due to the Covid-19 situation. Unfortunately, the overall Covid-19 situation got worse again in autumn and winter and therefore, this is not possible. Therefore, STACS 2021 will happen as a virtual conference as many other conferences before, with pre-recorded videos, short online presentations and discussions, and online social events.

We thank the Dagstuhl team for assisting us in the publication process and the final production of the proceedings. These proceedings contain extended abstracts of the accepted contributions and abstracts of the invited talks and the tutorials. The authors retain their rights and make their work available under a Creative Commons license. The proceedings are published electronically by Schloss Dagstuhl – Leibniz-Center for Informatics within their LIPIcs series. Finally we would like to thank Saarland University for its support.

Saarbrücken and Marseilles, March 2021 Markus Bläser and Benjamin Monmege

Conference Organization

Program Committee

C. Aiswarya	Chennai Mathematical Institute, India
Antoine Amarilli	Telecom Paris, France
Christel Baier	Technische Universität Dresden, Germany
Petra Berenbrink	University of Hamburg, Germany
Markus Bläser	Saarland University, Germany, co-chair
Mikołaj Bojańczyk	University of Warsaw, Poland
Nicolas Bousquet	CNRS, Université Lyon 1, France
Dmitry Chistikov	University of Warwick, UK
Radu Curticapean	ITU Copenhagen, Denmark
Jérôme Durand-Lose	University of Orléans, France
Anna Gal	University of Texas, Austin, USA
Eun Jung Kim	CNRS, University Paris Dauphine, France
François Le Gall	Nagoya University, Japan
Erik Jan van Leeuwen	Utrecht University, The Netherlands
Martin Loebl	Charles University, Prague, Czech Republic
Carsten Lutz	University of Bremen, Germany
Irène Marcovici	Université de Lorraine, France
Kitty Meeks	University of Glasgow, UK
Tobias Mömke	University of Augsburg, Germany
Benjamin Monmege	Aix-Marseille University, France, co-chair
Pat Morin	Carleton University, Canada
Igor Potapov	University of Liverpool, UK
Chandan Saha	IISc Bangalore, India
Kai Salomaa	Queen's University, Canada
Ramprasad Saptharishi	TIFR Mumbai, India
Thatchaphol Saranurak	TTI Chicago, USA
Sven Schewe	University of Liverpool, UK
Markus L. Schmid	Humboldt-University Berlin, Germany
Hadas Shachnai	Technion, Israel
Shay Solomon	Tel Aviv University, Israel
Magnus Wahlström	Royal Holloway, UK

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Steering committee

Olaf Beyersdorff	Friedrich-Schiller-Universität Jena
Dietmar Berwanger	CNRS, Université Paris-Saclay, France
Martin Dietzfelbinger	Technische Universität Ilmenau, Germany
Arnaud Durand	Université de Paris, France
Christoph Dürr	CNRS, Sorbonne Université, France, co-chair
Henning Fernau	Universität Trier, Germany
Arne Meier	Leibniz Universität Hannover, Germany
Cyril Nicaud	Université Paris-Est, France
Rolf Niedermeier	Technische Universität Berlin, Germany
Natacha Portier	ENS Lyon, France
Thomas Schwentick	Technische Universität Dortmund, Germany, co-chair
Ioan Todinca	Université d'Orléans, France

Local organizing committee (Saarland University)

Markus Bläser, chair
Julian Dörfler
Felix Freiberger
Sabine Nermerich
Sandra Neumann
Kristina Scherbaum
Charilaos Zisopoulos

Subreviewers

408 external subreviewers assisted the PC. We apologize to every subreviewer who does not appear in this list (because his or her review was entered manually into easychair).

Mohammad Ali Abam
Amir Abboud
Pierre Aboulker
Duncan Adamson
Peyman Afshani
Jungho Ahn
Saeed Akhoondian Amiri
Carme Alvarez
Antonios Antoniadis
Simon Apers
Sepehr Assadi
Yossi Azar
Ashwinkumar Badanidiyuru
Oana Balalau
Sayan Bandyapadhyay
Benoit Barbot
Siddharth Barman
Valentin Bartier
Libor Barto
Bruno Bauwens
Ruben Becker
Paul Bell
Thomas Bellitto
Aleksandrs Belovs
Ran Ben Basat
Huxley Bennett
Benjamin Bergougnoux
Christoph Berkholz
Siddharth Bhandari
Arnab Bhattacharyya
Laurent Bienvenu
Felix Biermeier
Ahmad Biniaz
Achim Blumensath
Greg Bodwin
Adrien Boiret
Édouard Bonnet
Pierre Bourhis
Joshua Brakensiek
Cornelius Brand
Karl Bringmann
Caroline Brosse
Benjamin Merlin Bumpus
Elisabet Burjons
Laurine Bénéteau
Martin Böhm
Michaël Cadilhac
Cezar Campeanu
Florent Capelli
Chiara Capresi
Arnaud Carayol
Clément Carbonnel
Lorenzo Carlucci
Nofar Carmeli
Olivier Carton
Katrin Casel
Jerome Casse
Sourav Chakraborty
Jérémie Chalopin
T-H. Hubert Chan
Witold Charatonik
Philippe Chassaing
Evangelos Chatziafratis
Yun Kuen Cheung
Nai-Hui Chia
Rajesh Chitnis
Da-Jung Cho
Keerti Choudhary
Tobias Christiani
Thomas Colcombet
Ágnes Cseh
Murilo da Silva
Konrad K. Dabrowski
Clément Dallard
Victor Dalmau
Amit Daniely
Bireswar Das
Joel Day
Paloma de Lima
Arnaud De Mesmay
Mateus De Oliveira Oliveira
Martin Delacourt
Argyrios Deligkas
Holger Dell
Martin Dietzfelbinger
Cunsheng Ding
Michael Dinitz
Rod Downey
Stephane Durocher
Talya Eden
Eduard Eiben
Hicham El-Zein
Khaled Elbassioni
Ehsan Emamjomeh-Zadeh
David Eppstein
Kousha Etessami
Léo Exibard
Yaron Fairstein
Nazim Fates
Uriel Feige
Moran Feldman
Laurent Feuilloley
Jiri Fiala
Nathanaël Fijalkow
Jirka Fink
Jacob Focke
Viktor Fredslund-Hansen
Tom Friedetzky
Tobias Friedrich
Martin Fürer
Maximilien Gadouleau
Andreas Galanis
Chaya Ganesh
Ankit Garg
Paul Gastin
Zsolt Gazdag
Gilles Geeraerts
Sevag Gharibian
Reza Gheissari
Suprovat Ghoshal
George Giakkoupis
Archontia Giannopoulou
Yuval Gil
Marinus Gottschau
Nicolas Grelier
Benoit Groz
Tom Gur
Vladimir Gusev
Gregory Gutin
Stefan Göller
Christopher Hahn
Magnús M. Halldórsson

Thekla Hamm
Yassine Hamoudi
Tero Harju
David Harris
Prahladh Harsha
Meng He
Marc Heinrich
Benjamin Hellouin de Menibus
Milan Hladík
Petr Hlineny
Markus Holzer
Juha Honkala
Hamed Hosseinpour
Mathieu Hoyrup
Jing Huang
Marcus Hutter
Rupert Hölzl
John Iacono
Tanmay Inamdar
Davis Issac
Louis Jachiet
Meena Jagadeesan
Damien Jamet
Rajesh Jayaram
Mark Jerrum
Artur Jeż
Zhengfeng Ji
Vincent Jugé
Marcin Jurdzinski
Dominik Kaaser
Tomas Kaiser
Ida Kantor
Akitoshi Kawamura
Yasushi Kawase
Neeraj Kayal
Alexandr Kazda
Chris Keeler
Thomas Kesselheim
Azadeh Khaleghi
Arindam Khan
Kamyar Khodamoradi
Evangelos Kipouridis
Elena Kirshanova
Sándor Kisfaludi-Bak
Martin Klazar
Peter Kling
Dušan Knop
Jakob Bæk Tejs Knudsen
Sang-Ki Ko
Tomohiro Koana
Yusuke Kobayashi
Petr Kolman
Dennis Komm
Christian Komusiewicz
Stavros Konstantinidis
Bahram Kouhestani
Martin Koutecky
Lukasz Kowalik
Alexander Kozachinskiy
Laszlo Kozma
Jan Kratochvil
Stefan Kratsch
Andrei Krokhin
Manfred Kufleitner
Ariel Kulik
Mrinal Kumar
Dietrich Kuske
Antti Kuusisto
O-Joung Kwon
Marvin Künnemann
Pierre L'Ecuyer
Arnaud Labourel
Victor Lagerkvist
Michael Lampis
Martin Lange
Sophie Laplante
Hung Le
Euiwoong Lee
Engel Lefaucheux
Daniel Lemire
Nathan Lhote
Jason Li
Jian Li
Yinan Li
Mathieu Liedloff
Nutan Limaye
Vincent Limouzy
Zhiheng Liu
William Lochet
Théodore Lopez
Anand Louis
Vladimir Lysikov
Christof Löding
Ramanujan M. Sridharan
Hugh Macpherson
Manfred Madritsch
Meena Mahajan
Soumen Maity
Guillaume Malod
Nikhil Mande
Silviu Maniu
David Manlove
Pasin Manurangsi
Nicolas Markey
Eric Martin
Arnaud Mary
Fionn Mc Inerney
Samuel McCauley
Arne Meier
Paul Melotti
Stefan Mengel
Neeldhara Misra
Pranabendu Misra
Dieter Mitsche
Takaaki Mizuki
Tulasimohan Molli
Mikael Monet
Benoît Monin
Matthew Moore
Antoine Mottet
Shay Mozes
Partha Mukhopadhyay
Filip Murlak
Ahad N. Zehmakan
Tigran Nagapetyan
Vineet Nair
Jonathan Narboni
Guyslain Naves
Joe Neeman
Rian Neogi
Timothy Ng
Valtteri Niemi
Matthias Niewerth
Andrey Nikolaev
Harumichi Nishimura
Reino Niskanen
Thomas Nowak
Krzysztof Nowicki
Pascal Ochem
Joanna Ochremiak
Alexander Okhotin
Karolina Okrasa
Aurélien Ooms
Jakub Opršal

Sebastian Ordyniak
Charles Paperman
Pavel Paták
Anthony Perez
Michał Pilipczuk
Ilia Ponomarenko
Thomas Prest
Jaikumar Radhakrishnan
Venkatesh Raman
Mayank Rathee
Ansis Rosmanis
Paweł Rzążewski
Bryce Sandlund
Thomas Sauerwald
Martin Schirneck
Steffen Schuldenzucker
Pierre Senellart
C. Seshadhri
Sebastian Siebertz
Michael Skotnica
Michiel Smid
Andreas Spillner
Piyush Srivastava
Giannos Stamoulis
Manon Stipulanti
Donald Stull
Warut Suksompong
Suguru Tamaki
Sébastien Tavenas
Kevin Tian
Csaba Toth
Konstantinos Tsakalidis
Duru Türkoğlu
André van Renssen
José Verschae
Jan Vondrak
Stefan Walzer
Kunihiro Wasa
Andreas Wiese
Marcin Wrochna
Kuan Yang
Se-Young Yun
Amir Zandieh
Hang Zhou

Yota Otachi
Julie Parreaux
Vincent Penelle
William Pettersson
Michael Pinsker
Amaury Pouly
Gabriele Puppis
Saladi Rahul
Michael Rao
Malin Rau
Peter Rossmanith
Prakash Saivasan
Swagato Sanyal
Rosario Scatamacchia
Sylvain Schmitz
Roy Schwartz
Maria Serna
Alexander Shen
Florian Sikora
Michał Skrzypczak
Taylor Smith
Joachim Spoerhase
B Srivathsan
Rafał Stefański
Milos Stojakovic
C. R. Subramanian
John Sylvester
Till Tantau
Sam Thomas
Hans Raj Tiwary
Behrouz Touri
Madhur Tulsiani
Ryuhei Uehara
Daniel Vaz
Pavel Veselý
Polina Vytnova
Haitao Wang
Nicole Wein
Sebastian Wild
Christian Wulff-Nilsen
Eylon Yogev
Viktor Zamaraev
Meirav Zehavi
Pawel Zielinski

Prashant Pandey
Paweł Parys
Pan Peng
Astrid Pieterse
Jakob Piribauer
M. Praveen
Karin Quaas
Patrick Rall
Cyrus Rashtchian
Liam Roditty
Aviad Rubinstein
Sai Sandeep
Ignasi Sau
Kevin Schewior
Philipp Schneider
Pavel Semukhin
Olivier Serre
Suhail Sherif
Alexander Skopalik
Friedrich Slivovsky
Krzysztof Sornat
Srikanth Srinivasan
Katherine Staden
Frank Stephan
Yann Strozecki
Yuichi Sudo
Jean-Marc Talbot
Jakub Tarnawski
Michaël Thomazo
Ioan Todinca
Meng-Tsung Tsai
Mykhaylo Tyomkyn
Pavel Valtr
Rahul Vaze
Tiphaine Viard
Michael Walter
Justin Ward
Albert H. Werner
Damien Woods
Jan Philipp Wächter
Yuichi Yoshida
Or Zamir
Peter Zeman
Jakub Łącki

Optimization, Complexity and Invariant Theory

Peter Bürgisser ✉
Institut für Mathematik, Technische Universität Berlin, Germany

Abstract

Invariant and representation theory studies symmetries by means of group actions and is a well established source of unifying principles in mathematics and physics. Recent research suggests its relevance for complexity and optimization through quantitative and algorithmic questions. The goal of the talk is to give an introduction to new algorithmic and analysis techniques that extend convex optimization from the classical Euclidean setting to a general geodesic setting. We also point out surprising connections to a diverse set of problems in different areas of mathematics, statistics, computer science, and physics.

2012 ACM Subject Classification Theory of computation → Continuous optimization

Keywords and phrases geometric invariant theory, geodesic optimization, non-commutative optimization, null cone, operator scaling, moment polytope, orbit closure intersection, geometric programming

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.1

Category Invited Talk

Related Version *Full Version*: `https://arxiv.org/abs/1910.12375`

Funding *Peter Bürgisser*: Supported by the ERC under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 787840.)

Acknowledgements The talk (and this write-up) are mainly based on the joint articles [14, 15] with Cole Franks, Ankit Garg, Rafael Oliveira, Michael Walter and Avi Wigderson. I am grateful to my coauthors for inspiration and enlightening discussions and thank Levent Dogan, Yinan Li, Visu Makam, Harold Nieuwboer and Philipp Reichenbach for valuable feedback.

1 Introduction

Consider a group G that acts by linear transformations on the complex Euclidean space $V = \mathbb{C}^m$. This partitions V into *orbits*: For a vector $v \in V$, the orbit $\mathcal{O}_v$ consists of all vectors of the form $g \cdot v$ to which the action of a group element $g \in G$ can map v.

The most basic algorithmic question in this setting is as follows. Given a vector $v \in V$, compute (or approximate) the smallest ℓ_2-norm of any vector in the orbit of v, that is, $\inf\{\|w\|_2 : w \in \mathcal{O}_v\}$. Remarkably, this simple question, for different groups and actions, captures natural important problems in computational complexity, algebra, analysis, statistics and quantum information. When restricted to commutative groups G, this amount to unconstrained geometric programming (see Section 2). In particular, this already captures all linear programming problems!

Starting with [37, 39], a series of recent works including [38, 16, 34, 62, 2, 13] designed algorithms and analysis tools to handle this basic and other related optimization problems over *non-commutative* groups G. In all these works, the groups at hand are products of at least two copies of rather specific linear groups (SL(n)'s or tori), to support the algorithms and analysis. These provided efficient solutions for some applications, and *through algorithms*, the resolution of some purely structural mathematical open problems.

We mention here some of the diverse applications of the paradigm of optimization over non-commutative groups.

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 1; pp. 1:1–1:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1. **Algebraic identities:** Given an arithmetic formula (with inversion gates) in non-commuting variables, is it identically zero?
2. **Quantum information:** Given density matrices describing local quantum states of various parties, is there a global pure state consistent with the local states?
3. **Eigenvalues of sums of Hermitian matrices:** Given three real n-vectors, do there exist three Hermitian $n \times n$ matrices A, B, C with these *prescribed* spectra, such that $A + B = C$?
4. **Analytic inequalities:** Given m linear maps $A_i : \mathbb{R}^n \to \mathbb{R}^{n_i}$ and $p_1, \ldots, p_m \geq 0$, does there exist a finite constant C such that for all integrable functions $f_i : \mathbb{R}^{n_i} \to \mathbb{R}_+$ we have

   $$\int_{x \in \mathbb{R}^n} \prod_{i=1}^m f_i(A_i x) dx \ \leq C \prod_{i=1}^m \|f_i\|_{1/p_i} ?$$

   These inequalities are the celebrated Brascamp-Lieb inequalities, which capture the Cauchy-Schwarz, Hölder, Loomis-Whitney, and many further inequalities.
5. **Maximum Likelihood Estimation** Consider a centered Gaussian random variable $Y \in \mathbb{R}^n$ with a covariance matrix Ψ being an element of the matrix normal model $\mathcal{M}(p, q) = \{\Psi_1 \otimes \Psi_2 \mid \Psi_1 \in \mathrm{PD}_p, \Psi_2 \in \mathrm{PD}_q\}$. What is the number of samples needed to achieve almost surely the existence and uniqueness of maximum likelihood estimators?

At first glance, it is far from obvious that solving any of these problems has any relation to either optimization or groups. However, not only symmetries naturally exist in all of them, but they also help both in formulating them as optimization problems over groups, suggesting natural algorithms (or at least heuristics) for them, and finally in providing tools for analyzing these algorithms. It perhaps should be stressed that symmetries exist in many examples in which the relevant groups are commutative (e.g., perfect matching in bipartite graphs, matrix scaling, and more generally in linear, geometric, and hyperbolic programming); however in these cases, standard convex optimization or combinatorial algorithms can be designed and analyzed without any reference to these existing symmetries.

Polynomial time algorithms were first given in [37] for Problem 1 (the works [50, 27, 49] later discovered completely different algebraic algorithms), in [13] for Problem 2 (cf. [79] and the structural results [57, 25, 22, 21, 81, 80, 19]), in [59, 64, 68, 17, 34] for Problem 3 (the celebrated structural result in [59] and the algorithmic results of [64, 68, 17] solved the decision problem, while [34] solved the search problem), and in [38] for Problem 4. However the algorithms in [38, 34, 13] remain exponential time in various input parameters, exemplifying only one aspect of many in which the current theory and understanding is lacking. Problem 5 was recently solved by in [4, 29], proving a conjecture in [32] and generalized to tensor normal models in [30].

The unexpected connections revealed in this study are far richer than the mere relevance of optimization and symmetries to such problems. One type are connections between problems in disparate fields. For example, the analytic Problem 4 turns out to be a special case of the algebraic Problem 1. Moreover, Problem 1 has (well-studied) differently looking but equivalent formulations in quantum information theory and in invariant theory, which are automatically solved by the same algorithm. Another type of connections are of purely structural open problems solved through such geodesic algorithms, reasserting the importance of the computational lens in mathematics. One was the first dimension-independent bound on the Paulsen problem in operator theory, obtained ingeniously through such an algorithm in [62] (this work was followed by [46], who gave a strikingly simpler proof and stronger bounds). Another was a quantitative bound on the continuity of the best constant C in

Problem 4 (in terms of the input data), important for non-linear variants of such inequalities. This bound was obtained through the algorithm in [38] and relies on its efficiency; previous methods used compactness arguments that provided no bounds.

We have no doubt that more unexpected applications and connections will follow. The most extreme and speculative perhaps among such potential applications is to develop a deterministic polynomial-time algorithm for the polynomial identity testing (PIT) problem. Such an algorithm will imply major algebraic or Boolean lower bounds, namely either separating VP from VNP, or proving that NEXP has no small Boolean circuits [51]. We note that this goal was a central motivation of the initial work in this sequence [37], which provided such a deterministic algorithm for Problem 1 above, the non-commutative analog of PIT. The "real" PIT problem (in which variables commute) also has a natural group of symmetries acting on it, which does not quite fall into the frameworks developed so far. Yet, the hope of proving lower bounds via optimization methods is a fascinating (and possibly achievable) one. This agenda of hoping to shed light on the PIT problem by the study of invariant theoretic questions was formulated in the fifth paper of the Geometric Complexity Theory (GCT) series [66, 67], but see [40].

In this talk, we describe the main results of the paper [15], which unifies and generalizes the above mentioned works. A key to all of them are the notions of *geodesic convexity* (which generalizes the familiar Euclidean notion of convexity) and the *moment map* (which generalizes the familiar Euclidean gradient) in the curved space and new metrics induced by the group action. The paper [15] naturally extends the familiar first and second order methods of standard convex optimization. Geodesic analogs of these methods are designed, which, respectively, have oracle access to first and second order "derivatives" of the function being optimized, and apply to any (reductive) group action. The first order method developed (which is a non-commutative version of gradient descent) replaces and extends the use of "alternate minimization" in most past works, and thus can accommodate more general group actions. For instance, this covers the cases of symmetric tensors (bosons) and antisymmetric tensors (fermions) with the standard action of $\mathrm{SL}(n)$, where alternating minimization does not apply. The second order method developed in [15] greatly generalizes the one used for the particular group action corresponding to operator scaling in [2]. It may be thought of as a geodesic analog of the "trust region method" [24] or the "box-constrained Newton method" [23, 3] applied to a regularized function. For both methods, in this non-commutative setting, we recover the familiar convergence behavior of the classical commutative case: to achieve "proximity" ε to the optimum, the first order method converges in $O(1/\varepsilon)$ iterations and the second order method in $O(\operatorname{poly}\log(1/\varepsilon))$ iterations.

As in the commutative case, the fundamental challenge is to understand the "constants" hidden in the big-O notation of each method. These depend on "smoothness" properties of the function optimized, which in turn are determined by the action of the group G on the space V that defines the optimization problem. The main technical contributions of the theory developed in [15] is to identify the key parameters which control this dependence, and to bound them for various actions to obtain concrete running time bounds. These parameters depend on a combination of algebraic and geometric properties of the group action, in particular the irreducible representations occurring in it. As mentioned, despite the technical complexity of defining (and bounding) these parameters, the way they control convergence of the algorithms is surprisingly elegant. The paper [15] also develops important technical tools which naturally extend ones common in the commutative theory, including regularizers, diameter bounds, numerical stability, and initial starting points, which are key to the design and analysis of the presented (and hopefully future) algorithms in the geodesic setting. As in previous works, we also address other optimization problems beyond the basic "norm minimization" question above, in particular the minimization of the moment map, and the membership problem for *moment polytopes*; a very rich class of polytopes (typically with exponentially many vertices and facets) which arises magically from any such group action.

2 Non-commutative optimization

We now give an introduction to non-commutative optimization and contrast its geometric structure and convexity properties with the familiar commutative setting. The basic setting is that of a continuous group G acting linearly on an m-dimensional complex vector space $V \cong \mathbb{C}^m$. Think of G as either the group of $n \times n$ complex invertible matrices, denoted $\mathrm{GL}(n)$, or the group of diagonal such matrices, denoted $\mathrm{T}(n)$.[1] The latter corresponds to the commutative case and the former is a paradigmatic example of the non-commutative case. An (linear) *action* (also called *representation*) of a group G on a complex vector space V is a group homomorphism $\pi : G \to \mathrm{GL}(V)$, that is, an association of an invertible linear map $\pi(g)\colon V \to V$ for every group element $g \in G$ satisfying $\pi(g_1 g_2) = \pi(g_1)\pi(g_2)$ for all $g_1, g_2 \in G$.[2] Further suppose that V is also equipped with a Hermitian inner product $\langle \cdot, \cdot \rangle$ and hence a norm $\|v\| := \langle v, v \rangle$.

Given a vector $v \in V$ one can consider the optimization problem of taking the infimum of the norm in the orbit of the vector v under the action of G. More formally, we define the *capacity* of v (with respect to π) by

$$\mathrm{cap}(v) := \inf_{g \in G} \|\pi(g)v\|.$$

This notion generalizes the matrix and operator capacities developed in [45, 42] (to see this, carry out the optimization over one of the two group variables) as well as the polynomial capacity of [44]. It turns out that this simple-looking optimization problem is already very general in the commutative case and, in the non-commutative case, captures all examples discussed in the introduction.

Let us first consider the commutative case, $G = \mathrm{T}(n)$ acting on V. In this simple case, all actions π have a very simple form. We give two equivalent descriptions, first of how any representation π splits into one-dimensional irreducible representations, and another describing π as a natural scaling action on n-variate polynomials with m monomials.

The irreducible representations are given by an orthonormal basis $v_1, \ldots, v_m$ of V such that the v_j are simultaneous eigenvectors of all the matrices $\pi(g)$. That is, for all $g = \mathrm{diag}(g_1, \ldots, g_n) \in \mathrm{T}(n)$ and $j \in [m]$, we have

$$\pi(g)v_j = \lambda_j(g)v_j, \quad \text{where} \quad \lambda_j(g) = \textstyle\prod_{i=1}^n g_i^{\omega_{j,i}} \tag{2.1}$$

for fixed integer vectors $\omega_1, \ldots, \omega_m \in \mathbb{Z}^n$, which are called *weights* and encode the simultaneous eigenvalues, and completely determine the action. Below we also refer to the weights of a representation π of $\mathrm{GL}(n)$, defined as the weights of π restricted to $\mathrm{T}(n)$.

A natural way to view all these actions is as follows. The natural action of $\mathrm{T}(n)$ on $\mathbb{C}^n$ by matrix-vector multiplication induces an action of $\mathrm{T}(n)$ on n-variate polynomials $V = \mathbb{C}[x_1, x_2, \ldots, x_n]$: simply, any group element $g = \mathrm{diag}(g_1, \ldots, g_n)$ "scales" each x_i to $g_i x_i$. Note that any monomial $x^\omega = \prod_{i=1}^n x_i^{\omega(i)}$ (where ω is the integer vector of exponents) is an eigenvector of this action, with an eigenvalue $\lambda(g) = \prod_{i=1}^n g_i^{\omega(i)}$.

[1] The theory works whenever the group is connected, algebraic and reductive.
[2] We further assume that π is a morphism of algebraic groups, i.e., given by rational functions.

Now fix m integer vectors ω_j as above. Consider the linear space of n-variate Laurent polynomials (monomials may have negative exponents) with the following m monomials: $v_j = x^{\omega_j} = \prod_{i=1}^n x_i^{\omega_{j,i}}$. The action on any polynomial $v = \sum_{j=1}^m c_j v_j$ is precisely the one described above, scaling each coefficient by the eigenvalue of its monomial. The norm $\|v\|$ of a polynomial is the sum of the square moduli of its coefficients. Now let us calculate the capacity of this action. For any $v = \sum_{j=1}^m c_j v_j$,

$$\operatorname{cap}(v)^2 = \inf_{g_1,\ldots,g_n \in \mathbb{C}^*} \textstyle\sum_{j=1}^m |c_j|^2 \prod_{i=1}^n |g_i|^{2\omega_{j,i}} = \inf_{x\in\mathbb{R}^n} \sum_{j=1}^m |c_j|^2 e^{x\cdot\omega_j}, \tag{2.2}$$

where we used the change of variables $x_i = \log|g_i|^2$, which makes the problem convex (in fact, log-convex). This class of optimization problems (of optimizing norm in the orbit of a commutative group) is known as *geometric programming* and is well-studied in the optimization literature (see, e.g., Chapter 4.5 in [10]). Hence for non-commutative groups, one can view computing $\operatorname{cap}(v)$ as *non-commutative geometric programming.* Is there a similar change of variables that makes the problem convex in the non-commutative case? It does not seem so. However, the non-commutative case also satisfies a notion of convexity, known as geodesic convexity, which we will study next.

2.1 Geodesic convexity

Geodesic convexity generalizes the notion of convexity in the Euclidean space to arbitrary Riemannian manifolds. We will not go into the notion of geodesic convexity in this generality but just mention what it amounts to in our concrete setting of norm optimization for $G = \mathrm{GL}(n)$.

It turns out the appropriate way to define geodesic convexity in this case is as follows. Fix an action π of $\mathrm{GL}(n)$ and a vector v. Then $\log\|\pi(e^{tH}g)v\|$ is convex in the real parameter t for every Hermitian matrix H and $g \in \mathrm{GL}(n)$. This notion of convexity is quite similar to the notion of Euclidean convexity, where a function is convex iff it is convex along all lines. However, it is far from obvious how to import optimization techniques from the Euclidean setting to work in this non-commutative geodesic setting. An essential ingredient we describe next is the geodesic notion of a gradient, called the *moment map.*

2.2 Moment map

The moment map is by definition the gradient of the function $\log\|\pi(g)v\|$ (understood as a function of v), at the identity element of the group, $g = I$. It captures how the norm of the vector v changes when we act on it by infinitesimal perturbations of the identity.

Again, we start with the commutative case $G = \mathrm{T}(n)$ acting on the multivariate Laurent polynomials. For a direction vector $h \in \mathbb{R}^n$ and a real perturbation parameter t, let $e^{th} = \operatorname{diag}\left(e^{th_1}, \ldots, e^{th_n}\right)$. Then, for $G = \mathrm{T}(n)$, the moment map is the function $\mu\colon V \setminus \{0\} \to \mathbb{R}^n$, defined by the following property:

$$\mu(v)\cdot h = \partial_{t=0}\left[\log\left\|\pi(\operatorname{diag}(e^{th})v\right\|\right],$$

for all $h \in \mathbb{R}^n$. That is, the directional derivative in direction h is given by the dot product $\mu(v)\cdot h$. Here one can see that the moment map matches the notion of Euclidean gradient. For the action of $\mathrm{T}(n)$ in Equation (2.1), we have

$$\mu(v) = \nabla_{x=0} \log\left(\sum_{j=1}^m |c_j|^2 e^{x\cdot\omega_j}\right) = \frac{\sum_{j=1}^m |c_j|^2 \omega_j}{\sum_{j=1}^m |c_j|^2}. \tag{2.3}$$

Note that the gradient $\mu(v)$ at any point v is a convex combination of the weights. Viewing v as a polynomial, the gradient thus belongs to the so-called *Newton polytope* of v, namely the convex hull of the exponent vectors of its monomials. Conversely, every point in that polytope is a gradient of some polynomial v with these monomials. We will soon return to this curious fact!

We now proceed to the non-commutative case, focusing on $G = \mathrm{GL}(n)$. Denote by $\mathrm{Herm}(n)$ the set of $n \times n$ complex Hermitian matrices.[3] Here directions will be parametrized by $H \in \mathrm{Herm}(n)$. For the case of $G = \mathrm{GL}(n)$, the moment map is the function $\mu \colon V \setminus \{0\} \to \mathrm{Herm}(n)$ defined (in complete analogy to the commutative case above) by the following property that

$$\mathrm{tr}[\mu(v)H] = \partial_{t=0} \left[\log \left\|\pi(e^{tH})v\right\|\right]$$

for all $H \in \mathrm{Herm}(n)$. That is, the directional derivative in direction H is given by $\mathrm{tr}[\mu(v)H]$.

In the commutative case, Equation (2.3) is a convex combination of the weights ω_j. Thus, the image of μ is the convex hull of the weights – a convex polytope. This brings us to moment polytopes.

2.3 Moment polytopes

One may ask whether the above fact is true for actions of GL(n): is the set $\{\mu(v) : v \in V \backslash \{0\}\}$ convex? This turns out to be blatantly false: for instance, for the action of $\mathrm{GL}(n)$ on $\mathbb{C}^n$ by matrix-vector multiplication the moment map is $\mu(v) = vv^\dagger/\|v\|^2$, and its image is clearly not convex. However, there is still something deep and non-trivial that can be said. Given a Hermitian matrix $H \in \mathrm{Herm}(n)$, define its *spectrum* to be the vector of its eigenvalues arranged in non-increasing order. That is, $\mathrm{spec}(H) := (\lambda_1, \ldots, \lambda_n)$, where $\lambda_1 \geq \cdots \geq \lambda_n$ are the eigenvalues of H. Amazingly, the set of spectra of moment map images, that is,

$$\Delta := \left\{\mathrm{spec}\big(\mu(v)\big) : 0 \neq v \in V\right\}, \tag{2.4}$$

is a convex polytope for every representation π [70, 60, 5, 41, 55]! These polytopes are called *moment polytopes.*

Let us mention two important examples of moment polytopes. The examples are for actions of products of GL(n)'s but the above definitions generalize almost immediately.

► **Example 2.1** (Horn's problem). Let $G = \mathrm{GL}(n) \times \mathrm{GL}(n) \times \mathrm{GL}(n)$ act on $V = \mathrm{Mat}(n) \oplus \mathrm{Mat}(n)$ as follows: $\pi(g_1, g_2, g_3)(X, Y) := (g_1 X g_3^{-1}, g_2 Y g_3^{-1})$. The moment map in this case is

$$\mu(X, Y) = \frac{(XX^\dagger, YY^\dagger, -(X^\dagger X + Y^\dagger Y))}{\|X\|_F^2 + \|Y\|_F^2}.$$

Using that $XX^\dagger$ and $X^\dagger X$ are positive semidefinite and isospectral, we obtain the following moment polytopes, which characterize the eigenvalues of sums of Hermitian matrices, i.e., Horn's problem (see, e.g., [36, 9]):

$$\Delta = \left\{(\mathrm{spec}(A), \mathrm{spec}(B), \mathrm{spec}(-A - B)) \mid A, B \in \mathrm{Mat}(n),\ A \geq 0,\ B \geq 0,\ \mathrm{tr}\, A + \mathrm{tr}\, B = 1\right\}.$$

These polytopes are known as the *Horn polytopes* and correspond to Problem 3 in the introduction. They have been characterized mathematically in [58, 59, 7, 72] and algorithmically in [64, 68, 17].

[3] The reason we are restricting to directions in $\mathbb{R}^n$ in the $\mathrm{T}(n)$ case and to directions in Herm_n in the $\mathrm{GL}(n)$ case is that imaginary and skew-Hermitian directions, respectively, do not change the norm.

The preceding is one of the simplest example of a moment polytope associated with the representation of a quiver (the star quiver with two edges); see [31] for this notion. Quiver representations are relevant for the solution of Problem 5.

▶ **Example 2.2** (Tensor action)**.** $G = \mathrm{GL}(n_1)\times\mathrm{GL}(n_2)\times\mathrm{GL}(n_3)$ acts on $V = \mathbb{C}^{n_1}\otimes\mathbb{C}^{n_2}\otimes\mathbb{C}^{n_3}$ as follows: $\pi(g_1, g_2, g_3)v := (g_1 \otimes g_2 \otimes g_3)v$. We can think of vectors $v \in V$ as tripartite quantum states with local dimensions n_1, n_2, n_3. Then the moment map for this group action captures precisely the notion of *quantum marginals*. That is, $\mu(v) = (\rho_1, \rho_2, \rho_3)$, where $\rho_k = \mathrm{tr}_{k^c}(vv^\dagger)$ denotes the reduced density matrix describing the state of the k^{th} particle. This corresponds to Problem 2 in the introduction.

The moment polytopes in this case are known as *Kronecker polytopes*, since they can be equivalently described in terms of the Kronecker coefficients of the symmetric group. These polytopes have been studied in [57, 25, 22, 21, 81, 80, 19, 78, 13].

There is a more refined notion of a moment polytope. One can look at the collection of spectra of moment maps of vectors in the orbit of a particular vector $v \in V$. Surprisingly, its closure,

$$\Delta(v) := \overline{\{\mathrm{spec}(\mu(w)) : w \in \mathcal{O}_v\}}$$

is a convex polytope as well, called the *moment polytope of* v [70, 11]! It can equivalently be defined as the spectra of moment map images of the orbit's closure in projective space.

2.4 Null cone

Fix a representation π of the group G on a vector space V (again assume $G = \mathrm{T}(n)$ or $G = \mathrm{GL}(n)$ for simplicity). The *null cone* for this group action is defined as the set of vectors v such that $\mathrm{cap}(v) = 0$:

$$\mathcal{N} := \{v \in V : \mathrm{cap}(v) = 0\}.$$

In other words, v is in the null cone if and only if 0 lies in the orbit-closure of v. It is of importance in invariant theory due to the results of Hilbert and Mumford [47, 69] which state that the null cone is the algebraic variety defined by non-constant homogeneous invariant polynomials of the group action (see, e.g., the excellent textbooks [26, 76]).

Let us see what the null cone for the action of $\mathrm{T}(n)$ in Equation (2.1) is. Recall from Equation (2.2), the formulation for $\mathrm{cap}(v)$. It is easy to see that $\mathrm{cap}(v) = 0$ iff there exists $x \in \mathbb{R}^n$ such that $x \cdot \omega_j < 0$ for all $j \in \mathrm{supp}(v)$, where $\mathrm{supp}(v) = \{j \in [m] : c_j \neq 0\}$ for $v = \sum_{j=1}^m c_j v_j$. Thus the property of v being in the null cone is captured by a simple linear program defined by $\mathrm{supp}(v)$ and the weights ω_j's. Hence the null cone membership problem for non-commutative group actions can be thought of as *non-commutative linear programming.*

We know by Farkas' lemma that there exists $x \in \mathbb{R}^n$ such that $x\cdot\omega_j < 0$ for all $j \in \mathrm{supp}(v)$ iff 0 does not lie in $\mathrm{conv}\{\omega_j : j \in \mathrm{supp}(v)\}$. In other words, $\mathrm{cap}(v) = 0$ iff $0 \notin \Delta(v)$. Is this true in the non-commutative world? It is! This is the Kempf-Ness theorem [53] and it is a consequence of the geodesic convexity of the function $g \to \log\|\pi(g)v\|$. The Kempf-Ness theorem can be thought of as a *non-commutative duality theory* paralleling the linear programming duality given by Farkas' lemma (which corresponds to the commutative world). Let us now mention an example of an interesting null cone in the non-commutative case.

▶ **Example 2.3** (Operator scaling, or left-right action)**.** The action of the group $G = \mathrm{SL}(n) \times \mathrm{SL}(n)$ on $\mathrm{Mat}(n)^k$ via $\pi(g, h)(X_1, \ldots, X_k) := (gX_1h^T, \ldots, gX_kh^T)$ is called the *left-right* action. (Recall $\mathrm{SL}(n)$ denotes the group of $n \times n$ matrices with determinant 1.) The null cone for this action captures *non-commutative singularity* (see, e.g., [50, 37, 27, 49]) and Problem 1 in the introduction. The left-right action has been crucial in getting deterministic polynomial time algorithms for the non-commutative rational identity testing problem [50, 37, 27, 49]. The commutative analogue is the famous polynomial identity testing (PIT) problem, for which designing a deterministic polynomial time algorithm remains a major open question in derandomization and complexity theory. We remark that the corresponding algebraic variety $\mathrm{Sing}_{n,m}$, consisting of m-tuples in $\mathrm{Mat}(n)$ which span only singular matrices, recently has been shown to be not a null cone [65].

▶ **Example 2.4** (Generalized Kronecker quivers)**.** The action of $G = \mathrm{GL}(n) \times \mathrm{GL}(n)$ on k-tuples of matrices $(X_1, \ldots, X_k)$ via $\pi(g,h)(X_1,\ldots,X_k) := (gX_1h^{-1}, \ldots, gX_kh^{-1})$ is sometimes also referred to as the left-right action. It can be obtained from action of Example 2.3 via the isomorphism $h \mapsto (h^{-1})^T$ of $\mathrm{GL}(n)$. This action is associated to the *generalized Kronecker quiver.*

▶ **Example 2.5** (Simultaneous conjugation)**.** The action of the group $G = \mathrm{GL}(n)$ on k-tuples of matrices in $V = (\mathrm{Mat}(n))^d$ by $\pi(g)(X_1,\ldots,X_k) := (gX_1g^{-1}, \ldots, gX_kg^{-1})$ is associated to the quiver with a single vertex and k self-loops, briefly called *k-loop quiver.*

3 Computational problems and state of the art

In this section, we describe the main computational questions that are of interest for the optimization problems discussed in the previous section and then discuss what is known about them in the commutative and non-commutative worlds.

▶ **Problem 3.1** (Null cone membership)**.** *Given (π, v), determine if v is in the null cone, i.e., if $\mathrm{cap}(v) = 0$. Equivalently, test if $0 \notin \Delta(v)$.*

The null cone membership problem for $\mathrm{GL}(n)$ is interesting only when the action $\pi(g)$ is given by rational functions in the $g_{i,j}$ rather than polynomials. This is completely analogous to the commutative case (e.g., the convex hull of weights ω_j with positive entries never contains the origin). In the important case that π is homogeneous, the null cone membership problem is interesting precisely when the total degree is zero, so that scalar multiples of the identity matrix act trivially. Thus, in this case the null cone membership problem for $G = \mathrm{GL}(n)$ is equivalent to the one for $G = \mathrm{SL}(n)$.

▶ **Problem 3.2** (Scaling)**.** *Given (π, v, ε) such that $0 \in \Delta(v)$, output a group element $g \in G$ such that $\|\operatorname{spec}(\mu(g)v)\|_2 = \|\mu(\pi(g)v)\|_F \leq \varepsilon$.*

In particular, the following promise problem can be reduced to Problem 3.2: Given (π, v, ε), decide whether $0 \notin \Delta(v)$ under the promise that either $0 \in \Delta(v)$ or 0 is ε-far from $\Delta(v)$. In fact, there always exists $\varepsilon > 0$, depending only on the group action, such that this promise is satisfied! Thus, the null cone membership problem can always be reduced to the scaling problem (see Corollary 4.5 below).

One can develop a non-commutative duality theory [15, Section 3.4] showing that an efficient agorithm to minimize the norm on an orbit closure of a vector v (i.e., approximate the capacity of v) under the promise that $0 \in \Delta(v)$ results in an efficient algorithm for the scaling problem and hence for the null cone membership problem. This motivates our next computational problem.

▶ **Problem 3.3** (Norm minimization)**.** *Given (π, v, ε) such that $\mathrm{cap}(v) > 0$, output a group element $g \in G$ such that $\log\|\pi(g)v\| - \log \mathrm{cap}(v) \leq \varepsilon$.*

We also consider the moment polytope membership problem for an arbitrary point $p \in \mathbb{Q}^n$.

▶ **Problem 3.4** (Moment polytope membership). *Given (π, v, p), determine if $p \in \Delta(v)$.*

The moment polytope membership problem is more general than the null cone membership problem, but there is a reduction from the former to the latter via the "shifting trick" from [70, 11], which forms the basis of the algorithms for the moment polytope membership problem in [15]. As in the case of the null cone, we can consider a scaling version of the moment polytope membership problem.

▶ **Problem 3.5** (p-scaling). *Given (π, v, p, ε) such that $p \in \Delta(v)$, output an element $g \in G$ such that $\|\mathrm{spec}(\mu(\pi(g)v)) - p\|_2 \leq \varepsilon$.*

The above problem has been referred to as *nonuniform scaling* [13] or, for operators, matrices and tensors, as *scaling with specified or prescribed marginals* [34]. The following problem can be reduced to Problem 3.5: Given (π, v, p, ε), decide whether $p \in \Delta(v)$ under the promise that either $p \in \Delta(v)$ or p is ε-far from $\Delta(v)$. One can combine the shifting trick with the non-commutive duality theory to show that there is a value $\varepsilon > 0$ with bitsize polynomial in the input size such that this is promise is always satisfied [15]. Thus, the moment polytope membership problem can be reduced to p-scaling.

There are several interesting input models for these problems. One could explicitly describe the weights $\omega_1, \ldots, \omega_m$ for an action of $\mathrm{T}(n)$ (Equation (2.1)) and then describe v as $\sum_{j=1}^m c_j v_j$ by describing the c_j's. The analogous description in the non-commutative world would be to describe the irreducible representations occuring in V. Alternately, one could give black box access to the function $\|\pi(g)v\|$, or to the moment map $\mu(\pi(g)v)$, etc. Sometimes π can be a non-uniform input as well, such as a fixed family of representations like the simultaneous left-right action Example 2.3 as done in [37]. The inputs p and ε will be given in their binary descriptions but we will see that some of the algorithms run in time polynomial in their unary descriptions.

▶ Remark 3.6 (Running time in terms of ε). By standard considerations about the bit complexity of the facets of the moment polytope, it can be shown that polynomial time algorithms for the scaling problems (Problems 3.2 and 3.5) result in polynomial time algorithms for the exact versions (Problems 3.1 and 3.4, respectively). Polynomial time requires, in particular, $\mathrm{poly}(\log(1/\varepsilon))$ dependence on ε; a $\mathrm{poly}(1/\varepsilon)$ dependence is only known to suffice in special cases.

3.1 Commutative groups and geometric programming

In the commutative case, the preceding problems are reformulations of well-studied optimization problems and much is known about them computationally. To see this, consider the action of $\mathrm{T}(n)$ as in Equation (2.1), and a vector $v = \sum_{j=1}^m c_j v_j$. It follows from Section 2.4 that v is in the null cone iff $0 \notin \Delta(v) = \mathrm{conv}\{\omega_j : c_j \neq 0\}$. Recall from Equation (2.2), the formulation for $\mathrm{cap}(v)$. Since this formulation is convex, it follows that, given $\omega_1, \ldots, \omega_m \in \mathbb{Z}^n$ (recall this is the description of π) and $c_1, \ldots, c_m \in \mathbb{Q}[i]$ (each entry described in binary), there is a polynomial-time algorithm for the null cone membership problem via linear programming [54, 52]. The same is true for the moment polytope membership problem. The capacity optimization problem is an instance of *(unconstrained) geometric programming*. The recent paper [18] describes interior-point methods for this, which run in polynomial time. Before [18], it was hard to find an exact reference for the existence of a polynomial time algorithm for geometric programming; however, is was known that polynomial time can be achieved using the ellipsoid algorithm as done for the same problem in slightly different settings in the papers [43, 74, 75]. There has been work in the oracle setting as well, in which one has oracle access to the function $\|\pi(g)v\|$. The advantage of the oracle setting is that one can handle exponentially large representations of $\mathrm{T}(n)$ when it is not possible to describe all the weights explicitly. A very general result of this form is proved in [75]. While not explicitly mentioned in [75], their techniques can also be used to design polynomial time algorithms for *commutative* null cone and moment polytope membership in the oracle setting. Thus, in the commutative case, Problems 3.1, 3.3, and 3.4 are well-understood.

3.2 Non-commutative actions

Comparatively very little is known in the non-commutative case. In the special case, where the group is fixed, polynomial time algorithms were given by the use of quantifier elimination (which is inefficient) and, more recently, by Mulmuley in [67, Theorem 8.5] through a purely algebraic approach. For instance, this applies to the settings of $V = \operatorname{Sym}^d \mathbb{C}^n$ or $V = \Lambda^d\mathbb{C}^n$ with the natural action by $\mathrm{SL}(n)$, where n is fixed.

For nonfixed groups, the only two non-trivial group actions for which there are known polynomial-time algorithms for null cone membership (Problem 3.1) are the *simultaneous conjugation* (Example 2.5) and the *left-right* action (Example 2.4). Approximate algorithms for null cone membership have been designed for the *tensor action* of products of $\mathrm{SL}(n)$'s [16]. However the running time is exponential in the binary description of ε (i.e., polynomial in $1/\varepsilon$). This is the reason the algorithm does not lead to a polynomial time algorithm for the exact null cone membership problem for the tensor action.

Moment polytope membership is already interesting for the polytope Δ in (2.4), the moment polytope of the entire representation V (not restricted to any orbit closure). Even here, efficient algorithms are only known in very special cases, such as for the Horn polytope (Example 2.1) [64, 68, 17]. The structural results in [8, 72, 78] characterize Δ in terms of linear inequalities (it is known that in general there are exponentially many). Mathematically, this is related to the asymptotic vanishing of certain representation-theoretic multiplicities [11, 20, 6] whose non-vanishing is in general NP-hard to decide [48]. In [12] it was proved that the membership problem for Δ is in NP $\cap$ coNP. As Δ and $\Delta(v)$ coincide for generic $v \in V$, this problem captures the moment polytope membership problem (Problem 3.4) for almost all vectors (all except those in a set of measure zero).

The study of Problem 3.4 in the noncommutative case focused on *Brascamp-Lieb polytopes* (which are affine slices of moment polytopes). The paper [38] solved the moment polytope membership problem in time depending polynomially on the *unary* complexity of the target point. In [13], efficient algorithms were designed for the p-scaling problem (Problem 3.5) for tensor actions, extending the earlier work of [34] for the simultaneous left-right action. The running times of both algorithms are $\operatorname{poly}(1/\varepsilon)$; for this reason both algorithms result in moment polytope algorithms depending exponentially on the binary bitsize of p, as in [38].

Regarding the approximate computation of the capacity (Problem 3.3), efficient algorithms were previously known only for the simultaneous left-right action. The paper [37] gave an algorithm to approximate the capacity in time polynomial in all of the input description except ε, on which it had dependence $\operatorname{poly}(1/\varepsilon)$. The paper [2] gave an algorithm that depended polynomially on the input description; it has running time dependence $\operatorname{poly}(\log(1/\varepsilon))$ on the error parameter ε.

In terms of algorithmic techniques, all prior works that were based on optimization methods fall into two categories. One is that of *alternating minimization* (which can be thought of as a large-step coordinate gradient descent, i.e., roughly speaking as a first order method). However, alternating minimization is limited in applicability to "multilinear" actions of products of $\mathrm{T}(n)$'s or $\mathrm{GL}(n)$'s, where the action is linear in each component so that it is easy to optimize over one component when fixing all the others. This is true for all the actions described above and hence explains the applicability of alternating minimization (in fact, in all the above examples, one can even get a closed-form expression for the group element that has to be applied in each alternating step). The second category are geodesic analogues of *box-constrained Newton's methods* (second order). Recently, [2] designed an algorithm tailored towards the specific case of the simultaneous left-right action (Example 2.3), but no second order algorithms were known for other group actions. However, many group actions of interest – from classical problems in invariant theory about symmetric forms to the important variant of Problem 2 in the introduction for fermions – are not multilinear nor can otherwise be captured by the left-right action, and no efficient algorithms were known. All this motivates the development of new techniques.

The paper [15] shows how these limitations can be overcome. Specifically, it provides both first and second order algorithms (geodesic variants of gradient descent and box-constrained Newton's method) that apply in great generality and identify the main structural parameters that control the running time of these algorithm. We now describe these contributions in more detail.

4 Algorithmic and structural results

4.1 Essential parameters and structural results

We define here the essential parameters related to the group action which, in addition to dictating the running times of our first and second order methods, control the relationships between the null cone, the norm of the moment map, and the capacity, i.e., between Problems 3.1–3.3. For details we refer to [15]

We saw in Section 2 that for all actions of $\mathrm{T}(n)$ on a vector space V, one can find a basis of V consisting of simultaneous eigenvectors of the matrices $\pi(g)$, $g \in \mathrm{T}(n)$. While this is in general impossible for non-commutative groups, one can still decompose V into building blocks known as irreducible subspaces (or subrepresentations).

For $\mathrm{GL}(n)$, these are uniquely characterized by nonincreasing sequences $\lambda \in \mathbb{Z}^n$; such sequences λ are in bijection with irreducible representations $\pi_\lambda \colon \mathrm{GL}(n) \to \mathrm{GL}(V_\lambda)$. We say that λ *occurs in* π if one of its irreducible subspaces is of type λ. If all the λ occuring in π have nonnegative entries, then the entries of the matrix $\pi(g)$ are polynomials in the entries of g. Such representations π are called *polynomial*, and if all λ occuring in π have sum exactly (resp. at most) d, then π is said to be a *homogeneous polynomial representation of degree (resp. at most) d.*

Now we can define the complexity measure which captures the smoothness of the optimization problems of interest. One can think of the following measure as a *norm* of the Lie algebra representation Π, hence the name *weight norm.*

▶ **Definition 4.1** (Complexity measure I: weight norm). *We define the* weight norm $N(\pi)$ *of an action π of* $\mathrm{GL}(n)$ *by* $N(\pi) := \max\{\|\lambda\|_2 : \lambda$ *occurs in* $\pi\}$*, where* $\|\cdot\|_2$ *denotes the Euclidean norm.*

Another use of the weight norm is to provide a bounding ball for the moment polytope: one can show that the moment polytope is contained in a Euclidean ball of radius $N(\pi)$. The weight norm is in turn controlled by the degree of a polynomial representation. More specifically, if π is a polynomial representation of $\mathrm{GL}(n)$ of degree at most d, then $N(\pi) \leq d$.

We now describe our second measure of complexity which will govern the running time bound for our second order algorithm. This parameter also features in Theorem 4.3 concerning quantitative non-commutative duality.

▶ **Definition 4.2** (Complexity measure II: weight margin). *The* weight margin $\gamma(\pi)$ *of an action* π *of* $\mathrm{GL}(n)$ *is the minimum Euclidean distance between the origin and the convex hull of any subset of the weights of* π *that does not contain the origin.*

Our running time bound will depend inversely on the weight margin. Two interesting examples with large (inverse polynomial) weight margin are the left-right action (Example 2.3) and simultaneous conjugation. The existing second order algorithm for the left-right action relied on the large weight margin of the action [2]. It is interesting that the simultaneous conjugation action (Example 2.5), the sole other interesting example of an action of a non-commutative group for which there are efficient algorithms for the null cone membership problem [71, 33, 28] (which have nothing to do with the weight margin), also happens to have large weight margin! On the other hand, the only generally applicable lower bound on the weight margin is $N(\pi)^{1-n}n^{-1}$, and indeed this exponential behavior is seen for the somewhat intractable 3-tensor action (Example 2.2), which has weight margin at most $2^{-n/3}$ and weight norm $\sqrt{3}$ [61, 35]. We arrange in a tabular form the above information about the weight margin and weight norm for various paradigmatic group actions in Table 1 (using a definition of the weight margin and weight norm that naturally generalizes the one given above for $\mathrm{GL}(n)$).

Table 1 Weight margin and norm for various representations.

Group action	**Weight margin $\gamma(\pi)$**	**Weight norm $N(\pi)$**
Matrix scaling	$\geq n^{-3/2}$; [63]	$\sqrt{2}$
Simultan. left-right action (Example 2.3)	$\geq n^{-3/2}$; [42]	$\sqrt{2}$
Quivers	$\geq (\sum_x n(x))^{-3/2}$	$\sqrt{2}$
Simultaneous conjugation (Example 2.5)	$\geq n^{-3/2}$	$\sqrt{2}$
3-tensor action (Example 2.2)	$\leq 2^{-n/3}$; [61, 35]	$\sqrt{3}$
Polynomial $\mathrm{GL}(n)$-action of degree d	$\geq d^{-n}dn^{-1}$	$\leq d$
Polynomial $\mathrm{SL}(n)$-action of degree d	$\geq (nd)^{-n}dn^{-1}$	$\leq d$

As the moment map is the gradient of the geodesically convex function $\log\|v\|$, it stands to reason that as $\mu(v)$ tends to zero, $\|v\|$ tends to the capacity $\mathrm{cap}(v)$. However, in order to use this relationship to obtain efficient algorithms, we need this to hold in a precise quantitative sense. To this end, we show in [15] the following fundamental relation between the capacity and the norm of the moment map, which is a quantitative strengthening of the Kempf-Ness result [53].

▶ **Theorem 4.3** (Noncommutative duality). *For* $v \in V \setminus \{0\}$ *we have*

$$1 - \frac{\|\mu(v)\|_F}{\gamma(\pi)} \leq \frac{\mathrm{cap}(v)^2}{\|v\|^2} \leq 1 - \frac{\|\mu(v)\|_F^2}{4N(\pi)^2}.$$

Equipped with these inequalities, it is easy to relate Problems 3.2 and 3.3.

▶ **Corollary 4.4.** *An output* g *for the norm minimization problem on input* (π, v, ε) *is a valid output for the scaling problem on input* $(\pi, v, N(\pi)\sqrt{8\varepsilon})$. *If* $\varepsilon/\gamma(\pi) < \frac{1}{2}$ *then an output* g *for the scaling problem on input* (π, v, ε) *is a valid output for the norm minimization problem on input* $(\pi, v, \frac{2\log(2)\varepsilon}{\gamma(\pi)})$.

Because $0 \in \Delta(v)$ if and only if $\operatorname{cap}(v) > 0$, Theorem 4.3 and Corollary 4.4 immediately yield the accuracy to which we must solve the scaling problem or norm minimization problem to solve the null cone membership problem:

▶ **Corollary 4.5.** *It holds that $0 \in \Delta(v)$ if and only if $\Delta(v)$ contains a point of norm smaller than $\gamma(\pi)$. In particular, solving the scaling problem with input $(\pi, v, \gamma(\pi)/2)$ or the norm minimization problem with $(\pi, v, \frac{1}{8}(\gamma(\pi)/2N(\pi))^2)$ suffices to solve the null cone membership problem for (π, v).*

In [15] we also provide analogues of the above corollaries for the moment polytope membership problem.

4.2 First order methods: structural results and algorithms

As discussed above, in order to approximately compute the capacity in the commutative case, one can just run a Euclidean gradient descent on the convex formulation in Equation (2.2). We will see that the gradient descent method naturally generalizes to the non-commutative setting. It is worth mentioning that there are several excellent sources of the analysis of gradient descent algorithms for geodesically convex functions (in the general setting of Riemannian manifolds and not just the group setting that we are interested in); see e.g., [77, 1, 83, 82, 73, 84] and references therein. The contribution in [15] is mostly in understanding the geometric properties (such as smoothness) of the optimization problems that we are concerned with, which allow us to carry out the classical analysis of Euclidean gradient descent in our setting and to obtain quantitative convergence rates, which are not present in previous work.

The natural analogue of gradient descent for the optimization problem $\operatorname{cap}(v)$ is the following: start with $g_0 = I$ and repeat, for T iterations and a suitable step size η:

$$g_{t+1} = e^{-\eta\mu(\pi(g_t)v)} g_t. \tag{4.1}$$

Finally, return the group element g among $g_0, \ldots, g_{T-1}$, which minimizes $\|\mu(\pi(g)v)\|_F$. A natural geometric parameter which governs the complexity (number of iterations T, step size η) of gradient descent is the *smoothness* of the function to be optimized. The smoothness parameter for actions of $\mathrm{T}(n)$ in Equation (2.1) can be shown to be $O(\max_{j\in[m]}\|\omega_j\|_2^2)$ (see, e.g., [75]), which is the square of the weight norm defined in Definition 4.1 for this action. We prove in [15] that, in general, the function $\log\|\pi(g)v\|$ is geodesically smooth, with a smoothness parameter which, analogously to the commutative case, is on the order of the square of the weight norm. We now state the running time for our geodesic gradient descent algorithm for Problem 3.2.

▶ **Theorem 4.6** (First order algorithm for scaling). *Fix a representation $\pi : \mathrm{GL}(n) \to \mathrm{GL}(V)$ and a unit vector $v \in V$ such that $\operatorname{cap}(v) > 0$ (i.e., v is not in the null cone). Then the above analogue (4.1) of gradient descent, with a number of iterations at most*

$$T = O\left(\frac{N(\pi)^2}{\varepsilon^2}\left|\log \operatorname{cap}(v)\right|\right),$$

outputs a group element $g \in \mathrm{GL}(n)$ satisfying $\|\mu(\pi(g)v)\|_F \le \varepsilon$.

The analysis of Theorem 4.6 relies on the smoothness of the function $F_v(g) := \log\|\pi(g)v\|$, which implies that

$$F_v(e^H g) \le F_v(g) + \operatorname{tr}\bigl[\mu\bigl(\pi(g)v\bigr)H\bigr] + N(\pi)^2\|H\|_F^2,$$

for all $g \in \mathrm{GL}(n)$ and for all Hermitian $H \in \mathrm{Herm}(n)$.

The paper [15] also describes and analyzes a first order algorithm for the p-scaling problem via the shifting trick.

4.3 Second order methods: structural results and algorithms

As mentioned in Section 3, the paper [2] (following the algorithms developed in [3, 23] for the commutative Euclidean case) developed a second order polynomial-time algorithm for approximating the capacity for the simultaneous left-right action (Example 2.3) with running time polynomial in the bit description of the approximation parameter ε. In [15] this algorithm is generalized to arbitrary groups and actions. It repeatedly optimizes quadratic Taylor expansions of the objective in a small neighbourhood. Such algorithms also go by the name "trust-region methods" in the Euclidean optimization literature [24]. The running time of this algorithm depends inversely on the weight margin defined in Definition 4.2.

▶ **Theorem 4.7** (Second-order algorithm for norm minimization). *Fix a representation* $\pi : \mathrm{GL}(n) \to \mathrm{GL}(V)$ *and a unit vector* $v \in V$ *such that* $\operatorname{cap}(v) > 0$. *Put* $C := |\log \operatorname{cap}(v)|$, $\gamma := \gamma(\pi)$ *and* $N := N(\pi)$. *Then the second order algorithm in [15], for a suitably regularized objective function, outputs* $g \in G$ *satisfying* $\log \|\pi(g)v\| \leq \log \operatorname{cap}(v) + \varepsilon$ *with a number of iterations at most*

$$T = O\left(\frac{N\sqrt{n}}{\gamma}\left(C + \log\frac{n}{\varepsilon}\right)\log\frac{C}{\varepsilon}\right).$$

The two main structural parameters which govern the runtime of the second order algorithm are the *robustness* (controlled by the weight norm) and a *diameter bound* (controlled by the weight margin). The robustness of a function bounds third derivatives in terms of second derivatives, similarly to the well-known notion of self concordance (however, in contrast to the latter, the robustness is not scale-invariant). As a consequence of the robustness, one shows that the function $F_v(g) = \log\|\pi(g)v\|$ is sandwiched between two quadratic expansions in a small neighbourhood:

$$F(g) + \partial_{t=0}F(e^{tH}g) + \frac{1}{2e}\partial^2_{t=0}F(e^{tH}g) \leq F(e^H g) \leq F(g) + \partial_{t=0}F(e^{tH}g) + \frac{e}{2}\partial^2_{t=0}F(e^{tH}g)$$

for every $g \in \mathrm{GL}(n)$ and $H \in \mathrm{Herm}(n)$ such that $\|H\|_F \leq 1/(4N(\pi))$.

Another ingredient in the analysis of the second order algorithm is to prove the existence of "well-conditioned" approximate minimizers, i.e., $g_\star \in G$, with small condition number satisfying $\log\|\pi(g_\star)v\| \leq \log \operatorname{cap}(v) + \varepsilon$. The bound on the condition numbers of approximate minimizers helps us ensure that the algorithm's trajectory always lies in a compact region with the use of appropriate regularizers. As in [2], this "diameter bound" is obtained by designing a suitable gradient flow and bounding the (continuous) time it takes for it to converge. A crucial ingredient of this analysis is Theorem 4.3 relating capacity and norm of the moment map.

This gradient flow approach, which can be traced back to works in symplectic geometry [56], is the only one we know for proving diameter bounds in the non-commutative case. In contrast, in the commutative case several different methods are available (see, e.g., [74, 75]). It is an important open problem to develop alternative methods for diameter bounds in the non-commutative case, which will also lead to improved running time bounds for the second order algorithm.

Finally, we note that [15] also contains results bounding the running time of the obtained algorithms, beyond the number of oracle calls, in terms of the bitsize needed to describe the given action π and the given vector v.

5 Conclusion

We believe that extending this theory will be fruitful both from a mathematical and computational point of view. The paper [15] points to the following intriguing open problems and suggests further research directions.

1. Is the null cone membership problem for general group actions in P? A natural intermediate goal is to prove that they are in NP ∩ coNP. The quantitative duality theory developed in this paper makes such a result plausible. The same question may be asked about the moment polytope membership problem for general group actions [12].
2. Can we find more general classes of problems or group actions where our algorithms run in polynomial time? In view of the complexity parameters we have identified, it is of particular interest to understand when the *weight margin* is only inverse polynomially rather than exponentially small.
3. Interestingly, when restricted to the commutative case discussed in Section 3, our algorithms' guarantees do not match those of cut methods in the spirit of the ellipsoid algorithm. Can we extend non-commutative/geodesic optimization to include cut methods as well as interior point methods? The foundations we lay in extending first and second order methods to the non-commutative case makes one optimistic that similar extensions are possible of other methods in standard convex optimization. The paper [18] explicitly designs and analyses a polynomial time interior-point method in the commutative setting.
4. Can geodesic optimization lead to new efficient algorithms in combinatorial optimization? We know that it captures algorithmic problems like bipartite matching (and more generally matroid intersection). How about perfect matching in general graphs – is the Edmonds polytope a moment polytope of a natural group action?
5. Can geodesic optimization lead to new efficient algorithms in algebraic complexity and derandomization? We know that the null cone membership problem captures polynomial identity testing (PIT) in non-commuting variables. The variety corresponding to classical PIT is however *not* a null cone [65]. Can our algorithms be extended beyond null cones to membership in more general classes of varieties?

References

1 Pierre-Antoine Absil, Robert E. Mahony, and Rodolphe Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, 2008. URL: `http://press.princeton.edu/titles/8586.html`.

2 Zeyuan Allen-Zhu, Ankit Garg, Yuanzhi Li, Rafael Mendes de Oliveira, and Avi Wigderson. Operator scaling via geodesically convex optimization, invariant theory and polynomial identity testing. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 172–181. ACM, 2018. `doi:10.1145/3188745.3188942`.

3 Zeyuan Allen-Zhu, Yuanzhi Li, Rafael Mendes de Oliveira, and Avi Wigderson. Much faster algorithms for matrix scaling. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 890–901. IEEE Computer Society, 2017. `doi:10.1109/FOCS.2017.87`.

4 Carlos Améndola, Kathlén Kohn, Philipp Reichenbach, and Anna Seigal. Invariant theory and scaling algorithms for maximum likelihood estimation, 2020. `arXiv:2003.13662`.

5 Michael F Atiyah. Convexity and commuting Hamiltonians. *Bulletin of the London Mathematical Society*, 14(1):1–15, 1982. `doi:10.1112/blms/14.1.1`.

6 Velleda Baldoni, Michèle Vergne, and M. Walter. Computation of dilated Kronecker coefficients. *J. Symb. Comput.*, 84:113–146, 2018. `doi:10.1016/j.jsc.2017.03.005`.

7 Prakash Belkale and Shrawan Kumar. Eigenvalue problem and a new product in cohomology of flag varieties. *Inventiones mathematicae*, 166:185–228, 2006. doi:10.1007/s00222-006-0516-x.

8 Arkady Berenstein and Reyer Sjamaar. Coadjoint orbits, moment polytopes, and the Hilbert–Mumford criterion. *Journal of the American Mathematical Society*, 13(2):433–466, 2000. doi:10.1090/S0894-0347-00-00327-1.

9 Nicole Berline, Michèle Vergne, and Michael Walter. The Horn inequalities from a geometric point of view. *L'Enseignement Mathématique*, 63:403–470, 2017. arXiv:1611.06917.

10 Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, 2014. doi:10.1017/CBO9780511804441.

11 Michel Brion. Sur l'image de l'application moment. In *Séminaire d'algebre Paul Dubreil et Marie-Paule Malliavin*, volume 1296 of *Lecture Notes in Mathematics*, pages 177–192. Springer, 1987.

12 Peter Bürgisser, Matthias Christandl, Ketan D. Mulmuley, and Michael Walter. Membership in moment polytopes is in NP and coNP. *SIAM J. Comput.*, 46(3):972–991, 2017. doi:10.1137/15M1048859.

13 Peter Bürgisser, Cole Franks, Ankit Garg, Rafael Mendes de Oliveira, Michael Walter, and Avi Wigderson. Efficient algorithms for tensor scaling, quantum marginals and moment polytopes. *CoRR*, abs/1804.04739, 2018. arXiv:1804.04739.

14 Peter Bürgisser, Cole Franks, Ankit Garg, Rafael Mendes de Oliveira, Michael Walter, and Avi Wigderson. Towards a theory of non-commutative optimization: Geodesic 1st and 2nd order methods for moment maps and polytopes. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 845–861. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00055.

15 Peter Bürgisser, Cole Franks, Ankit Garg, Rafael Mendes de Oliveira, Michael Walter, and Avi Wigderson. Towards a theory of non-commutative optimization: geodesic first and second order methods for moment maps and polytopes. *CoRR*, abs/1910.12375, 2019. arXiv:1910.12375.

16 Peter Bürgisser, Ankit Garg, Rafael Mendes de Oliveira, Michael Walter, and Avi Wigderson. Alternating minimization, scaling algorithms, and the null-cone problem from invariant theory. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, volume 94 of *LIPIcs*, pages 24:1–24:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ITCS.2018.24.

17 Peter Bürgisser and Christian Ikenmeyer. Deciding positivity of Littlewood-Richardson coefficients. *SIAM J. Discret. Math.*, 27(4):1639–1681, 2013. doi:10.1137/120892532.

18 Peter Bürgisser, Yinan Li, Harold Nieuwboer, and Michael Walter. Interior-point methods for unconstrained geometric programming and scaling problems, 2020. arXiv:2008.12110.

19 Matthias Christandl, Brent Doran, Stavros Kousidis, and Michael Walter. Eigenvalue distributions of reduced density matrices. *Communications in Mathematical Physics*, 332(1):1–52, 2014. doi:10.1007/s00220-014-2144-4.

20 Matthias Christandl, Brent Doran, and Michael Walter. Computing multiplicities of Lie group representations. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 639–648. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.43.

21 Matthias Christandl, Aram W Harrow, and Graeme Mitchison. Nonzero Kronecker coefficients and what they tell us about spectra. *Communications in Mathematical Physics*, 270(3):575–585, 2007. doi:10.1007/s00220-006-0157-3.

22 Matthias Christandl and Graeme Mitchison. The spectra of quantum states and the Kronecker coefficients of the symmetric group. *Communications in Mathematical Physics*, 261(3):789–797, 2006. doi:10.1007/s00220-005-1435-1.

23 Michael B. Cohen, Aleksander Madry, Dimitris Tsipras, and Adrian Vladu. Matrix scaling and balancing via box constrained newton's method and interior point methods. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 902–913. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.88.

24 Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. *Trust Region Methods*. MOS-SIAM Series on Optimization. SIAM, 2000. doi:10.1137/1.9780898719857.

25 Sumit Daftuar and Patrick Hayden. Quantum state transformations and the Schubert calculus. *Annals of Physics*, 315(1):80–122, 2005. doi:10.1016/j.aop.2004.09.012.

26 Harm Derksen and Gregor Kemper. *Computational invariant theory*. Springer, 2015.

27 Harm Derksen and Visu Makam. Polynomial degree bounds for matrix semi-invariants. *Advances in Mathematics*, 310:44–63, 2017. doi:10.1016/j.aim.2017.01.018.

28 Harm Derksen and Visu Makam. Algorithms for orbit closure separation for invariants and semi-invariants of matrices, 2018. arXiv:1801.02043.

29 Harm Derksen and Visu Makam. Maximum likelihood estimation for matrix normal models via quiver representations, 2020. arXiv:2007.10206.

30 Harm Derksen, Visu Makam, and Michael Walter. Maximum likelihood estimation for tensor normal models via Castling transforms, 2020. arXiv:2011.03849.

31 Harm Derksen and Jerzy Weyman. *An introduction to quiver representations*, volume 184. American Mathematical Society, 2017.

32 Mathias Drton, Satoshi Kuriki, and Peter Hoff. Existence and uniqueness of the Kronecker covariance MLE, 2020. arXiv:2003.06024.

33 Michael A. Forbes and Amir Shpilka. Explicit noether normalization for simultaneous conjugation via polynomial identity testing. *Electron. Colloquium Comput. Complex.*, 20:33, 2013. URL: http://eccc.hpi-web.de/report/2013/033.

34 Cole Franks. Operator scaling with specified marginals. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 190–203. ACM, 2018. doi:10.1145/3188745.3188932.

35 Cole Franks and Philipp Reichenbach. Barriers for recent methods in geodesic optimization. Preprint, 2020. arXiv:2102.06652.

36 William Fulton. Eigenvalues, invariant factors, highest weights, and Schubert calculus. *Bulletin of the American Mathematical Society*, 37(3):209–249, 2000. arXiv:math/9908012.

37 Ankit Garg, Leonid Gurvits, Rafael Mendes de Oliveira, and Avi Wigderson. A deterministic polynomial time algorithm for non-commutative rational identity testing. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 109–117. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.95.

38 Ankit Garg, Leonid Gurvits, Rafael Mendes de Oliveira, and Avi Wigderson. Algorithmic and optimization aspects of brascamp-lieb inequalities, via operator scaling. 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 397–409. ACM, 2017. doi:10.1145/3055399.3055458.

39 Ankit Garg, Leonid Gurvits, Rafael Mendes de Oliveira, and Avi Wigderson. Operator scaling: Theory and applications. *Found. Comput. Math.*, 20(2):223–290, 2020. doi:10.1007/s10208-019-09417-z.

40 Ankit Garg, Christian Ikenmeyer, Visu Makam, Rafael Mendes de Oliveira, Michael Walter, and Avi Wigderson. Search problems in algebraic complexity, GCT, and hardness of generators for invariant rings. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPIcs*, pages 12:1–12:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CCC.2020.12.

41 V. Guillemin and S. Sternberg. Convexity properties of the moment mapping. *Inventiones mathematicae*, 67:491–513, 1982.

42 Leonid Gurvits. Classical complexity and quantum entanglement. *J. Comput. Syst. Sci.*, 69(3):448–484, 2004. doi:10.1016/j.jcss.2004.06.003.

43 Leonid Gurvits. Combinatorial and algorithmic aspects of hyperbolic polynomials. *Electron. Colloquium Comput. Complex.*, (070), 2004. URL: http://eccc.hpi-web.de/eccc-reports/2004/TR04-070/index.html.

44 Leonid Gurvits. Hyperbolic polynomials approach to van der waerden/schrijver-valiant like conjectures: sharper bounds, simpler proofs and algorithmic applications. In Jon M. Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 417–426. ACM, 2006. doi:10.1145/1132516.1132578.

45 Leonid Gurvits and Peter N. Yianilos. The deflation-inflation method for certain semidefinite programming and maximum determinant completion problems. *Technical Report, NECI*, 1998.

46 Linus Hamilton and Ankur Moitra. The paulsen problem made simple. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 41:1–41:6. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ITCS.2019.41.

47 David Hilbert. Über die vollen Invariantensysteme. *Math. Ann.*, 42:313–370, 1893.

48 Christian Ikenmeyer, Ketan D. Mulmuley, and Michael Walter. On vanishing of Kronecker coefficients. *Comput. Complex.*, 26(4):949–992, 2017. doi:10.1007/s00037-017-0158-y.

49 Gábor Ivanyos, Youming Qiao, and K. V. Subrahmanyam. Constructive non-commutative rank computation is in deterministic polynomial time. In Christos H. Papadimitriou, editor, *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, volume 67 of *LIPIcs*, pages 55:1–55:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ITCS.2017.55.

50 Gábor Ivanyos, Youming Qiao, and K. V. Subrahmanyam. Non-commutative Edmonds' problem and matrix semi-invariants. *Comput. Complex.*, 26(3):717–763, 2017. doi:10.1007/s00037-016-0143-x.

51 Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Comput. Complex.*, 13(1-2):1–46, 2004. doi:10.1007/s00037-004-0182-6.

52 Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Comb.*, 4(4):373–396, 1984. doi:10.1007/BF02579150.

53 George Kempf and Linda Ness. The length of vectors in representation spaces. In *Algebraic geometry*, pages 233–243. Springer, 1979.

54 Leonid G Khachiyan. A polynomial algorithm in linear programming. In *Doklady Academii Nauk SSSR*, volume 244, pages 1093–1096, 1979.

55 Frances Kirwan. Convexity properties of the moment mapping, III. *Inventiones mathematicae*, 77(3):547–552, 1984.

56 Frances Clare Kirwan. *Cohomology of quotients in symplectic and algebraic geometry*, volume 31. Princeton University Press, 1984.

57 Alexander Klyachko. Quantum marginal problem and representations of the symmetric group, 2004. arXiv:quant-ph/0409113.

58 Alexander A Klyachko. Stable bundles, representation theory and hermitian operators. *Selecta Mathematica, New Series*, 4(3):419–445, 1998. doi:10.1007/s000290050037.

59 A Knutson and T Tao. The honeycomb model of $\mathrm{GL}_n(\mathbb{C})$ tensor products I: Proof of the saturation conjecture. *Journal of the American Mathematical Society*, 12(4):1055–1090, 1999. arXiv:math/9807160.

60 B. Kostant. On convexity, the Weyl group and the Iwasawa decomposition. *Ann. scient. E.N.S*, 6:413–455, 1973.

61 V. M. Kravtsov. Combinatorial properties of noninteger vertices of a polytope in a three-index axial assignment problem. *Cybernetics and Systems Analysis*, 43(1):25–33, 2007.

62 Tsz Chiu Kwok, Lap Chi Lau, Yin Tat Lee, and Akshay Ramachandran. The Paulsen problem, continuous operator scaling, and smoothed analysis. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 182–189. ACM, 2018. `doi:10.1145/3188745.3188794`.

63 Nathan Linial, Alex Samorodnitsky, and Avi Wigderson. A deterministic strongly polynomial algorithm for matrix scaling and approximate permanents. In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 644–652. ACM, 1998. `doi:10.1145/276698.276880`.

64 Jesús A. De Loera and Tyrrell B. McAllister. On the computation of Clebsch-Gordan coefficients and the dilation effect. *Exp. Math.*, 15(1):7–19, 2006. `doi:10.1080/10586458.2006.10128948`.

65 Visu Makam and Avi Wigderson. Singular tuples of matrices is not a null cone (and, the symmetries of algebraic varieties). *CoRR*, abs/1909.00857, 2019. `arXiv:1909.00857`.

66 Ketan Mulmuley. Geometric complexity theory V: equivalence between blackbox derandomization of polynomial identity testing and derandomization of noether's normalization lemma. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 629–638. IEEE Computer Society, 2012. `doi:10.1109/FOCS.2012.15`.

67 Ketan Mulmuley. Geometric complexity theory V: Efficient algorithms for Noether normalization. *Journal of the American Mathematical Society*, 30(1):225–309, 2017. `arXiv:1209.5993`.

68 Ketan D Mulmuley, Hariharan Narayanan, and Milind Sohoni. Geometric complexity theory III: on deciding nonvanishing of a Littlewood–Richardson coefficient. *Journal of Algebraic Combinatorics*, 36(1):103–110, 2012.

69 David Mumford. *Geometric invariant theory.* Springer-Verlag, 1965.

70 Linda Ness and David Mumford. A stratification of the null cone via the moment map. *American Journal of Mathematics*, 106(6):1281–1329, 1984. `doi:10.2307/2374395`.

71 Ran Raz and Amir Shpilka. Deterministic polynomial identity testing in non-commutative models. *Comput. Complex.*, 14(1):1–19, 2005. `doi:10.1007/s00037-005-0188-8`.

72 Nicolas Ressayre. Geometric invariant theory and the generalized eigenvalue problem. *Inventiones mathematicae*, 180(2):389–441, 2010. `doi:10.1007/s00222-010-0233-3`.

73 Hiroyuki Sato, Hiroyuki Kasai, and Bamdev Mishra. Riemannian stochastic variance reduced gradient algorithm with retraction and vector transport. *SIAM J. Optim.*, 29(2):1444–1472, 2019. `doi:10.1137/17M1116787`.

74 Mohit Singh and Nisheeth K. Vishnoi. Entropy, optimization and counting. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 50–59. ACM, 2014. `doi:10.1145/2591796.2591803`.

75 Damian Straszak and Nisheeth K. Vishnoi. Maximum entropy distributions: Bit complexity and stability. In Alina Beygelzimer and Daniel Hsu, editors, *Conference on Learning Theory, COLT 2019, 25-28 June 2019, Phoenix, AZ, USA*, volume 99 of *Proceedings of Machine Learning Research*, pages 2861–2891. PMLR, 2019. URL: `http://proceedings.mlr.press/v99/straszak19a.html`.

76 Bernd Sturmfels. *Algorithms in Invariant Theory.* Texts & Monographs in Symbolic Computation. Springer, 2008. `doi:10.1007/978-3-211-77417-5`.

77 Constantin Udriste. *Convex functions and optimization methods on Riemannian manifolds*, volume 297. Springer, 1994.

78 Michele Vergne and Michael Walter. Inequalities for moment cones of finite-dimensional representations. *Journal of Symplectic Geometry*, 15(4):1209–1250, 2017. `doi:10.4310/JSG.2017.v15.n4.a8`.

79 Frank Verstraete, Jeroen Dehaene, and Bart De Moor. Normal forms and entanglement measures for multipartite quantum states. *Physical Review A*, 68(1):012103, 2003. `doi:10.1103/PhysRevA.68.012103`.

80 Michael Walter. *Multipartite Quantum States and their Marginals.* PhD thesis, ETH Zurich, 2014. doi:10.3929/ethz-a-010250985.

81 Michael Walter, Brent Doran, David Gross, and Matthias Christandl. Entanglement polytopes: multiparticle entanglement from single-particle information. *Science*, 340(6137):1205–1208, 2013. doi:10.1126/science.1232957.

82 Hongyi Zhang, Sashank J. Reddi, and Suvrit Sra. Fast stochastic optimization on riemannian manifolds. *CoRR*, abs/1605.07147, 2016. arXiv:1605.07147.

83 Hongyi Zhang and Suvrit Sra. First-order methods for geodesically convex optimization. *CoRR*, abs/1602.06053, 2016. arXiv:1602.06053.

84 Hongyi Zhang and Suvrit Sra. Towards riemannian accelerated gradient methods. *CoRR*, abs/1806.02812, 2018. arXiv:1806.02812.

First-Order Transductions of Graphs

Patrice Ossona de Mendez
Centre d'Analyse et de Mathématique Sociales CNRS UMR 8557, Paris, France
Charles University, Prague, Czech Republic

Abstract

This paper is an extended abstract of my STACS 2021 talk "First-order transductions of graphs".

2012 ACM Subject Classification Theory of computation → Finite Model Theory; Mathematics of computing → Graph theory

Keywords and phrases Finite model theory, structural graph theory

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.2

Category Invited Talk

Funding This paper is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 810115 – Dynasnet).

Acknowledgements I would like to thank Jarik Nešetřil and Sebastian Siebertz for their most valuable help in preparing this presentation.

1 Introduction

Logical methods in Computer Science have a long history, as witnessed e.g. by the relative longevity of SQL in relational database management. More recently, Courcelle's theorem, which combines second-order logic and tree decompositions of graphs, showed that a many NP-complete algorithmic problems in graph theory can be solved in polynomial time on graphs with bounded tree-width (and even on graphs with bounded clique-width). At the heart of the latter result is the notion of monadic second-order transductions, which are a way to encode a graph within a structure using coloring and monadic second-order logic formulas. In this presentation we consider first-order transductions, for which the formulas have to be first-order formulas. As a counterpart for this strong restriction, many algorithmic problems become fixed parameter tractable when restricted to nowhere dense classes, which include classes excluding a topological minor thus, in particular, classes of planar graphs and classes of graphs with bounded degrees.

In this setting, the main challenge is to extend results obtained in the sparse setting (for bounded expansion classes and nowhere dense classes) to the dense setting, in a similar way the results about monadic second-order model checking have been extended from classes with bounded tree-width to classes with bounded clique-width.

2 Sparse classes

The study of classes of sparse graphs has long been divided into two dual points of view: on the one hand, classes of graphs with bounded degrees – and particularly classes of regular graphs, enjoy strong connections with group theory and important combinatorial properties deriving from spectral properties. On the other hand, classes excluding a minor are strongly related to topological graph theory, as witnessed by Robertson and Seymour's Graph Structure Theorem [36], which is probably the most important result in structural graph theory. It was believed for a long time that at the source of this duality lied a fundamental gap between

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 2; pp. 2:1–2:7
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the notions of minor and topological minor. However, the Graph Structure Theorem has recently been extended to graphs excluding a topological minor by Grohe and Marx [17] (see also [9]), witnessing that both notions are closer than expected, even though this extension builds on two types of blocks, namely graphs almost embedded on a surface and graphs with almost bounded degrees.

Another unifying approach [21] was proposed fifteen years ago by Nešetřil and the author, based on the concepts of shallow minors [33], of low tree-depth decompositions [22], of generalized coloring numbers [19, 39], and of quasi-wideness [7]. This approach led to a taxonomy of classes of sparse graphs, with two main dividing lines, which respectively delineate classes with bounded expansion [23] and nowhere dense classes [24, 25]. These types of graph classes received numerous characterizations, and we shall use two of them as definitions.

For a graph G and a non-negative integer k, we denote by $\mathrm{TM}_k(G)$ (where TM stands for topological minor) the set of all graphs H such that some $\leq k$-subdivision of H is a subgraph of G and, by extension, for a class $\mathscr{C}$ we define $\mathrm{TM}_k(\mathscr{C}) = \bigcup_{G\in\mathscr{C}} \mathrm{TM}_k(G)$. A class of graphs $\mathscr{C}$ is *nowhere dense* if $\mathrm{TM}_k(\mathscr{C})$ has bounded clique number (or, equivalently, if $\mathrm{TM}_k(\mathscr{C})$ is not the class $\mathscr{G}$ of all graphs) for every integer k for every non-negative integer k. A class $\mathscr{C}$ has *bounded expansion* if, for each integer k, there is a uniform bound on the average degrees of the graphs in $\mathrm{TM}_k(\mathscr{C})$ (or, equivalently, if each of the class $\mathrm{TM}_k(\mathscr{C})$ is degenerate).

It is remarkable that bounded expansion and nowhere-denseness can be defined indifferently using shallow topological minors (as above), shallow minors, or shallow immersions (see [8]). Also, bounded expansion can be defined indifferently using the average degree, the degeneracy, the chromatic number [8], and even the fractional chromatic number [10]. For an in-depth study of bounded expansion and nowhere dense classes, we refer the reader to [26].

The main aspect of sparse classes is probably that vertices are easily separated. This property, which may be formalized in terms of neighborhood covers and uniform quasi-wideness, is the core of the model checking algorithm of Grohe, Kreutzer, and Siebertz, who proved the following result [16].

▶ **Theorem 2.1.** *For every nowhere dense class $\mathscr{C}$ and every $\varepsilon > 0$ there is an algorithm that checks in time $f(\theta)\, n^{1+\varepsilon}$ if a graph $G \in \mathscr{C}$ with n vertices satisfies a first-order sentence θ.*

For monotone classes of graphs, under standard assumptions from complexity theory, nowhere-denseness is actually a necessary condition for first-order model checking to be fixed-parameter tractable.

Also, one of the manifestations of these separability properties lies in the low neighborhood complexity of bounded expansion and nowhere dense classes. Precisely, for a class $\mathscr{C}$ and an integer d define the maximum number $\pi_d^{\mathscr{C}}$ of traces of the balls of radius d on subsets of vertices of size n in graphs in $\mathscr{C}$:

$$\pi_d^{\mathscr{C}}(n) = \sup_{G\in\mathscr{C}} \max_{A\subseteq V(G), |A|=n} \bigl|\{A \cap N_d(v) : v \in V(G)\}\bigr|.$$

Then we have the following characterizations.

▶ **Theorem 2.2.** *A monotone class $\mathscr{C}$ has bounded expansion if and only if $\pi_d^{\mathscr{C}}(n) = O(n)$ for every integer d and it is nowhere dense if and only if $\pi_d^{\mathscr{C}}(n) = O(n^{1+\varepsilon})$ for every integer d and every $\varepsilon > 0$.*

The above characterization of bounded expansion classes was proved by Reidl, Villaamil, and Stavropoulos [35]; the difficult direction of the characterization of nowhere dense classes was proved by Gajarskỳ et al. [13] for the case $d = 1$ and by Eickmeyer et al. [11] for the general case. These characterizations have been dramatically strengthened to bounds on the shattering functions of first-order definable families of subsets of vertices by Pilipczuk, Siebertz, and Toruńczyk [32], thus unveiling a deep connection between the notions of sparse classes and the model theoretical notion of classes with low VC-density.

Also, it follows from another separation property, namely uniform quasi-wideness, that for a monotone class of graphs, nowhere-denseness, stability and dependence are equivalent properties, where the last two refer to the model theoretical fundamental dividing lines identified by Shelah in its classification theory [37]. Precisely, based on a result of Podewski and Ziegler [34] Adler and Adler [1] proved the following collapse.

▶ **Theorem 2.3.** *For a monotone class of graphs $\mathscr{C}$ the following are equivalent:*

1. *$\mathscr{C}$ is nowhere dense;*
2. *$\mathscr{C}$ is stable;*
3. *$\mathscr{C}$ is monadically stable;*
4. *$\mathscr{C}$ is dependent;*
5. *$\mathscr{C}$ is monadically dependent.*

These two examples witness an intimate connection between graph theoretical and model theoretical dividing lines. This suggests a possible extension of the ideas and constructions introduced to deal with sparse graphs by using techniques borrowed from model theory, like interpretations and transductions. The hope, behind the search for an extension, is the possibility to define a dense analog of sparsity for hereditary classes of graphs, which would witness a relatively low complexity. In particular, we expect to cover the case of the *small* hereditary classes, which are classes with $O(n^c\, n!)$ labeled graphs with n vertices.

Another outcome of this connection between graph theory and model theory lies in the existence of totally Borel model theoretical limits for sequences of graphs in a nowhere dense graph. Here the notion of convergence consists in the convergence, for every first-order formula $\varphi(\overline{x})$ of the satisfaction probability of $\varphi(\overline{x})$ when considering a uniform and independent random assignment of the vertices to the free variables [27], which generalizes the existence of graphing limits for locally convergent sequence of graphs with bounded degrees (see [20]).

3 Transductions

A (first-order) *transduction* is a way to encode a structure within another structure by means of a coloring and a first-order formula. Precisely, a transduction T from graphs to graphs is defined by a first-order formula $\varphi(x, y)$ with two free-variable in the language of vertex-colored graphs. The atomic formulas of this language are of the form $x = y$, $E(x, y)$ (meaning x is adjacent to y) and $M_i(x)$ with $i \in \mathbb{N}$ (meaning x has color i). For a graph G, the set $\mathsf{T}(G)$ contains all the graphs H with vertex set $A \subseteq V(G)$, for which there is a vertex coloring G^+ of G such that $H \models E(u, v)$ if and only if $G^+ \models \varphi(u, v)$. It follows directly from the definition that $\mathsf{T}(G)$ is a hereditary class of graphs. A class $\mathscr{D}$ is a T-*transduction* of a class $\mathscr{C}$ if $\mathscr{D} \subseteq \mathsf{T}(\mathscr{C}) := \bigcup_{G\in\mathscr{C}} \mathsf{T}(G)$. The intuition here is that the graphs in the class $\mathscr{D}$ are non essentially more complex than the graphs in the class $\mathscr{C}$ as they can be "encoded" within them. It is easily checked that the existence of a transduction from a class to another defines a quasi-order. This quasi-order has a maximum, the class of all graphs. Classes that are not equivalent to this class – which are in some sense reasonably difficult – are exactly those graphs classes that are monadically dependent, in the model theoretical sense, as follows from [3]. It also follows from [3] that the so-called monadically stable classes of graphs are exactly those class that have no transduction to the class of all half-graphs.

Admittedly, checking if there exists a transduction from a class $\mathscr{C}$ to a class $\mathscr{D}$ may be a highly difficult task. However, the special case where $\mathscr{D}$ is the class of all graphs (that is checking if a class is not monadically dependent) is usually easier to handle. Moreover, if a class $\mathscr{C}$ is monadically dependent then checking the existence of a transduction from $\mathscr{C}$ to the class of all half-graphs (that is checking if $\mathscr{C}$ is not monadically stable) is much easier as it is surprisingly sufficient to check if arbitrarily large half-graphs are semi-induced subgraphs of graphs in $\mathscr{C}$ [28].

The structure of the transduction quasi-order seems to be difficult to establish [31]. As nowhere dense classes are monadically stable [2], every transduction of a nowhere dense class is also monadically stable. The converse statement is the object of the next conjecture.

▶ **Conjecture 3.1.** *Every monadically stable class of graphs is a transduction of a nowhere dense class.*

In general, when H is a transduction of a known graph G and that both the transduction and the vertex-coloring used by the transduction to get G are known, the problem of checking if H satisfies a first-order sentence can be easily transformed in the problem of checking if the (colored) graph G satisfies a derived formula. However, when only H and some basic information about G and the transduction are known, the problem may become much more difficult. For instance, if a graph G is a transduction of a graph with maximum degree d, computing such a pre-image and a transduction is provably hard. However, Gajarský et al. [12] proved the following (see [14] for some extension of this result).

▶ **Theorem 3.2.** *For every transduction T and every integer d there exist an integer d' (depending on T and d) and an interpretation I such that if G is a T-transduction of a graph H with maximum degree d, then there is a graph H' of maximum degree d', computable in polynomial time from G, such that $G = \mathsf{I}(H')$.*

This supports the next conjecture.

▶ **Conjecture 3.3.** *First-order model checking is fixed-parameter tractable on monadically stable classes of graphs.*

Some results have been obtained toward this conjecture, showing that first-order model checking is fixed-parameter tractable on transductions of bounded expansion classes, provided that a specific decomposition of the graphs in the graphs (a so-called depth-2 low shrub-depth cover) is given [15]. This result is proved by proving that first-order transductions transport low shrub-depth covers, which are a generalization of low tree-depth decompositions. A side consequence of this is that transductions of bounded expansion classes are linearly χ-bounded (see [15, 30, 28]). Some related results have been obtained for monadically stable classes with bounded linear rank-width [29] (showing that they are computable transductions of classes with bounded pathwidth) and for monadically stable classes with bounded rank-width (showing that they are computable transductions of classes with bounded treewidth) [28].

4 Partially ordered graphs

The use of transductions has been used to extend some properties of bounded expansion classes and nowhere dense classes within the monadically stable realm. To go further, it is necessary to introduce, at least locally, some order-like substructures. A way to it is to consider classes of *partially ordered graphs*, that is graphs with an additional partial order on the vertices, a special important case being *ordered graphs*, which are graphs with a total order on the vertices. Another example are *tree-ordered graphs*, which are graphs with a tree-order on the vertices. It appears that that the concept of ordered graphs particularly fits to the study of the recently introduced twin-width invariant [6], inspired by a width invariant defined on permutations by Guillemot and Marx [18]. Classes with bounded twin-width include several well studied classes of graphs, like classes of graphs excluding a minor, unit interval graphs, and classes with bounded clique-width. These classes are small [4] (contain at most $O(n^c n!)$ graphs with n vertices), and have fixed parameter tractable first-order model checking when a contraction sequence of the graphs is provided [6]. An essential property of twin-width is that its boundedness is preserved by transductions, as proved by Bonnet et al. [6].

▶ **Theorem 4.1.** *Every transduction of a class with bounded twin-width has bounded twin-width.*

Simon and Toruńczyk [38] recently announced the following characterization of bounded twin-width classes, which has been independently proved by Bonnet et al. [5]:

▶ **Theorem 4.2.** *A class of graphs has bounded twin-width if and only if it is the reduct of a monadically dependent class of ordered graphs.*

It is remarkable that for hereditary classes $\mathscr{C}$ of ordered graphs one can prove that, under the standard $\mathsf{FPT} \neq \mathsf{AW}[*]$ assumption from complexity theory, monadic dependence is equivalent to fixed parameter tractability of first-order model checking [5].

It might be possible that the characterization given by Theorem 4.2 could extend to the whole realm of monadically dependent classes, by considering tree-orders instead of linear orders.

▶ **Conjecture 4.3.** *Every monadically dependent class of graphs is a transduction of a monadically dependent class of tree-ordered graphs, whose reduct (obtained by forgetting the partial order) is monadically stable.*

References

1 H. Adler. An introduction to theories without the independence property. preprint, 2008. URL: `http://www.logic.univie.ac.at/~adler/docs/nip.pdf`.

2 H. Adler and I. Adler. Interpreting nowhere dense graph classes as a classical notion of model theory. *European Journal of Combinatorics*, 36:322–330, 2014. doi:10.1016/j.ejc.2013.06.048.

3 J.T. Baldwin and S. Shelah. Second-order quantifiers and the complexity of theories. *Notre Dame Journal of Formal Logic*, 26(3):229–303, 1985.

4 E. Bonnet, C. Geniet, E.J. Kim, S. Thomassé, and R. Watrigant. Twin-width II: small classes. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1977–1996, 2021. doi:10.1137/1.9781611976465.118.

5 E. Bonnet, U. Giocanti, S. Thomassé, and P. Ossona de Mendez. Twin-width IV: low complexity matrices. in preparation, 2021.

6 E. Bonnet, E.J. Kim, S. Thomassé, and R. Watrigant. Twin-width I: tractable FO model checking. *CoRR*, abs/2004.14789, 2020, To appear at FOCS 2020. arXiv:2004.14789.

7 A. Dawar. Homomorphism preservation on quasi-wide classes. *Journal of Computer and System Sciences*, 76:324–332, 2010. doi:10.1016/j.jcss.2009.10.005.

8 Z. Dvořák. On forbidden subdivision characterizations of graph classes. *European Journal of Combinatorics*, 29(5):1321–1332, 2008. doi:10.1016/j.ejc.2007.05.008.

9 Z. Dvořák. A stronger structure theorem for excluded topological minors, 2012. arXiv:1209.0129v1.

10 Z. Dvořák, P. Ossona de Mendez, and H. Wu. 1-subdivisions, fractional chromatic number and Hall ratio. *Combinatorica*, 2020. to appear. doi:10.1007/s00493-020-4223-9.

11 K. Eickmeyer, A. C Giannopoulou, S. Kreutzer, O. Kwon, M. Pilipczuk, R. Rabinovich, and S. Siebertz. Neighborhood complexity and kernelization for nowhere dense classes of graphs. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.63.

12 J. Gajarský, P. Hliněný, J. Obdržálek, D. Lokshtanov, and M.S. Ramanujan. A new perspective on FO model checking of dense graph classes. In *Proceedings of the Thirty-First Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 176–184. ACM, 2016. doi:10.1145/3383206.

13 J. Gajarskỳ, P. Hliněnỳ, J. Obdržálek, S. Ordyniak, F. Reidl, P. Rossmanith, F.S. Villaamil, and S. Sikdar. Kernelization using structural parameters on sparse graph classes. *Journal of Computer and System Sciences*, 84:219–242, 2017. doi:10.1016/j.jcss.2016.09.002.

14 J. Gajarský and D. Kráľ. Recovering sparse graphs. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.29.

15 J. Gajarský, S. Kreutzer, J. Nešetřil, P. Ossona de Mendez, M. Pilipczuk, S. Siebertz, and S. Toruńczyk. First-order interpretations of bounded expansion classes. *ACM Transactions on Computational Logic*, 21(4):Article 29, 2020. doi:10.1145/3382093.

16 M. Grohe, S. Kreutzer, and S. Siebertz. Deciding first-order properties of nowhere dense graphs. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 89–98, New York, NY, USA, 2014. ACM. doi:10.1145/2591796.2591851.

17 M. Grohe and D. Marx. Structure theorem and isomorphism test for graphs with excluded topological subgraphs. *SIAM Journal on Computing*, 44(1):114–159, 2015. doi:10.1137/120892234.

18 S. Guillemot and D. Marx. Finding small patterns in permutations in linear time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 82–101, 2014. doi:10.1137/1.9781611973402.7.

19 H.A. Kierstead and W.T. Trotter. Planar graph coloring with an uncooperative partner. *J. Graph Theory*, 18(6):569–584, 1994. doi:10.1002/jgt.3190180605.

20 L Lovász. *Large Networks and Graph Limits*, volume 60 of *Colloquium Publications*. American Mathematical Society, 2012.

21 J. Nešetřil and P. Ossona de Mendez. Linear time low tree-width partitions and algorithmic consequences. In *STOC'06. Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 391–400. ACM Press, 2006. doi:10.1145/1132516.1132575.

22 J. Nešetřil and P. Ossona de Mendez. Tree depth, subgraph coloring and homomorphism bounds. *European Journal of Combinatorics*, 27(6):1022–1041, 2006. doi:10.1016/j.ejc.2005.01.010.

23 J. Nešetřil and P. Ossona de Mendez. Grad and classes with bounded expansion I. Decompositions. *European Journal of Combinatorics*, 29(3):760–776, 2008. doi:10.1016/j.ejc.2006.07.013.

24 J. Nešetřil and P. Ossona de Mendez. First order properties on nowhere dense structures. *The Journal of Symbolic Logic*, 75(3):868–887, 2010. doi:10.2178/jsl/1278682204.

25 J. Nešetřil and P. Ossona de Mendez. On nowhere dense graphs. *European Journal of Combinatorics*, 32(4):600–617, 2011. doi:10.1016/j.ejc.2011.01.006.

26 J. Nešetřil and P. Ossona de Mendez. *Sparsity (Graphs, Structures, and Algorithms)*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012. 465 pages.

27 J. Nešetřil and P. Ossona de Mendez. Existence of modeling limits for sequences of sparse structures. *The Journal of Symbolic Logic*, 84(2):452–472, 2019. doi:10.1017/jsl.2018.32.

28 J. Nešetřil, P. Ossona de Mendez, M. Pilipczuk, R. Rabinovich, and S. Siebertz. Rankwidth meets stability. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2014–2033, 2021. doi:10.1137/1.9781611976465.120.

29 J. Nešetřil, P. Ossona de Mendez, R. Rabinovich, and S. Siebertz. Linear rankwidth meets stability. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1180–1199, 2020. doi:10.1137/1.9781611975994.72.

30 J. Nešetřil, P. Ossona de Mendez, R. Rabinovich, and S. Siebertz. Classes of graphs with low complexity: The case of classes with bounded linear rankwidth. *European Journal of Combinatorics*, 91:103223, 2021. Special issue dedicated to Xuding Zhu's 60th birthday. doi:10.1016/j.ejc.2020.103223.

31 J. Nešetřil, P. Ossona de Mendez, and S. Siebertz. Towards an arboretum of monadically stable classes of graphs, 2020. arXiv:2010.02607.

32 M. Pilipczuk, S. Siebertz, and S. Toruńczyk. On the number of types in sparse graphs. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 799–808. ACM, 2018. doi:10.1145/3209108.3209178.

33 S. Plotkin, S. Rao, and W.D. Smith. Shallow excluded minors and improved graph decompositions. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 462–470. SIAM, 1994. doi:10.5555/314464.314625.

34 K.-P. Podewski and M. Ziegler. Stable graphs. *Fund. Math.*, 100:101–107, 1978.

35 F. Reidl, F.S. Villaamil, and K. Stavropoulos. Characterising bounded expansion by neighbourhood complexity. *European Journal of Combinatorics*, 75:152–168, 2019. doi:10.1016/j.ejc.2018.08.001.

36 N. Robertson and P.D. Seymour. Graph minors I–XXIII. *J. Combin. Theory Ser. B*, 1983–2010.

37 S. Shelah. *Classification theory and the number of non-isomorphic models.* North-Holland, 1990.

38 P. Simon and S. Toruńczyk. A model-theoretic characterization of bounded twin-width. personal communication, 2020.

39 X. Zhu. Colouring graphs with bounded generalized colouring number. *Discrete Math.*, 309(18):5562–5568, 2009. doi:10.1016/j.disc.2008.03.024.

On the Fluted Fragment

Lidia Tendera
Institute of Computer Science, University of Opole, Poland

Abstract

The *fluted fragment* is a recently rediscovered decidable fragment of first-order logic whose history is dating back to Quine and the sixties of the 20th century. The fragment is defined by fixing simultaneously the order in which variables occur in atomic formulas and the order of quantification of variables; no further restrictions concerning e.g. the number of variables, guardedness or usage of negation apply. In the talk we review some motivation and the history of the fragment, discuss the differences between the fluted fragment and other decidable fragments of first-order logic, present its basic model theoretic and algorithmic properties, and discuss recent work concerning limits of decidability of its extensions.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic; Theory of computation → Finite Model Theory

Keywords and phrases decidability, fluted fragment, first-order logic, complexity, satisfiability, non-elementary

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.3

Category Invited Talk

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 3; pp. 3:1–3:1
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Improved (Provable) Algorithms for the Shortest Vector Problem via Bounded Distance Decoding

Divesh Aggarwal
Centre for Quantum Technologies, Singapore
National University of Singapore, Singapore

Yanlin Chen
Institute of Information Science, Academia Sinica, Taipei, Taiwan

Rajendra Kumar
IIT Kanpur, India
National University of Singapore, Singapore

Yixin Shen
Université de Paris, IRIF, CNRS, F-75006, France

Abstract

The most important computational problem on lattices is the Shortest Vector Problem (SVP). In this paper, we present new algorithms that improve the state-of-the-art for provable classical/quantum algorithms for SVP. We present the following results.

1. A new algorithm for SVP that provides a smooth tradeoff between time complexity and memory requirement. For any positive integer $4 \leq q \leq \sqrt{n}$, our algorithm takes $q^{13n+o(n)}$ time and requires $poly(n) \cdot q^{16n/q^2}$ memory. This tradeoff which ranges from enumeration ($q = \sqrt{n}$) to sieving (q constant), is a consequence of a new time-memory tradeoff for Discrete Gaussian sampling above the smoothing parameter.
2. A quantum algorithm that runs in time $2^{0.9533n+o(n)}$ and requires $2^{0.5n+o(n)}$ classical memory and poly(n) qubits. This improves over the previously fastest classical (which is also the fastest quantum) algorithm due to [2] that has a time and space complexity $2^{n+o(n)}$.
3. A classical algorithm for SVP that runs in time $2^{1.741n+o(n)}$ time and $2^{0.5n+o(n)}$ space. This improves over an algorithm of [15] that has the same space complexity.

The time complexity of our classical and quantum algorithms are expressed using a quantity related to the kissing number of a lattice. A known upper bound of this quantity is $2^{0.402n}$, but in practice for most lattices, it can be much smaller and even $2^{o(n)}$. In that case, our classical algorithm runs in time $2^{1.292n}$ and our quantum algorithm runs in time $2^{0.750n}$.

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

Keywords and phrases Lattices, Shortest Vector Problem, Discrete Gaussian Sampling, Time-Space Tradeoff, Quantum computation, Bounded distance decoding

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.4

Related Version *Full Version*: `https://arxiv.org/abs/2002.07955`

Funding *Divesh Aggarwal*: This work was partially supported by the Singapore Ministry of Education under grant MOE2019-T2-1-145, National Research Foundation under grant R-710-000-012-135.
Rajendra Kumar: Supported in part by National Research Foundation Singapore under its AI Singapore Programme [Award Number: AISG-RP-2018-005].

Acknowledgements We would like to thank Pierre-Alain Fouque, Paul Kirchner, Amaury Pouly and Noah Stephens-Davidowitz for useful comments and suggestions.

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 4; pp. 4:1–4:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

A lattice $\mathcal{L} = \mathcal{L}(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) := \{\sum_{i=1}^{n} z_i \boldsymbol{b}_i : z_i \in \mathbb{Z}\}$ is the set of all integer combinations of linearly independent vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n \in \mathbb{R}^n$. We call n the rank of the lattice and $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ a basis of the lattice.

The most important computational problem on lattices is the Shortest Vector Problem (SVP). Given a basis for a lattice $\mathcal{L} \subseteq \mathbb{R}^n$, SVP asks us to compute a non-zero vector in $\mathcal{L}$ with the smallest Euclidean norm. Starting from the '80s, the use of approximate and exact solvers for SVP (and other lattice problems) gained prominence for their applications in algorithmic number theory [41], convex optimization [32, 34, 20], coding theory [17], and cryptanalysis tool [56, 14, 40]. The security of many cryptographic primitives is based on the worst-case hardness of (a decision variant of) approximate SVP to within polynomial factors [6, 44, 53, 52, 45, 23, 13] in the sense that any cryptanalytic attack on these cryptosystems that runs in time polynomial in the security parameter implies a polynomial time algorithm to solve approximate SVP to within polynomial factors. Such cryptosystems have attracted a lot of research interest due to their conjectured resistance to quantum attacks.

The SVP is a well studied computational problem in both its exact and approximate (decision) versions. By a randomized reduction, it is known to be NP-hard to approximate within any constant factor, and hard to approximate within a factor $n^{c/\log\log n}$ for some $c > 0$ under reasonable complexity-theoretic assumptions [42, 35, 27]. For an approximation factor $2^{\mathcal{O}(n)}$, one can solve SVP in time polynomial in n using the celebrated LLL lattice basis reduction algorithm [41]. In general, the fastest known algorithm(s) for approximating SVP within factors polynomial in n rely on (a variant of) the BKZ lattice basis reduction algorithm [54, 55, 7, 21, 25, 3], which can be seen as a generalization of the LLL algorithm and gives an $r^{n/r}$ approximation in $2^{\mathcal{O}(r)} \operatorname{poly}(n)$ time. All these algorithms internally use an algorithm for solving (near) exact SVP in lower-dimensional lattices. Therefore, finding faster algorithms to solve SVP is critical to choosing security parameters of cryptographic primitives.

As one would expect from the hardness results above, all known algorithms for solving exact SVP, including the ones we present here, require at least exponential time. In fact, the fastest known algorithms also require exponential space. There has been some recent evidence [4] showing that one cannot hope to get a $2^{o(n)}$ time algorithm for SVP if one believes in complexity theoretic conjectures such as the (Gap) Exponential Time Hypothesis. Most of the known algorithms for SVP can be broadly classified into two classes: (i) the algorithms that require memory polynomial in n but run in time $n^{\mathcal{O}(n)}$ and (ii) the algorithms that require memory $2^{\mathcal{O}(n)}$ and run in time $2^{\mathcal{O}(n)}$.

The first class, initiated by Kannan [34, 28, 26, 22, 48], combines basis reduction with exhaustive enumeration inside Euclidean balls. While enumerating vectors requires $2^{\mathcal{O}(n \log n)}$ time, it is much more space-efficient than other kinds of algorithms for exact SVP.

Another class of algorithms, and currently the fastest, is based on sieving. First developed by Ajtai, Kumar, and Sivakumar [7], they generate many lattices vectors and then divide-and-sieve to create shorter and shorter vectors iteratively. A sequence of improvements [51, 49, 46, 50, 2, 5], has led to a $2^{n+o(n)}$ time and space algorithm by sieving the lattice vectors and carefully controlling the distribution of output, thereby outputting a set of lattice vectors that contains the shortest vector with overwhelming probability.

An alternative approach using the Voronoi cell of the lattice was proposed by Micciancio and Voulgaris [47] and gives a deterministic $2^{2n+o(n)}$-time and $2^{n+o(n)}$-space algorithm for SVP (and many other lattice problems).

There are variants [49, 46, 39, 11] of the above mentioned sieving algorithms that, under some heuristic assumptions, have an asymptotically smaller (but still $2^{\Theta(n)}$) time and space complexity than their provable counterparts.

Algorithms giving a time/space tradeoff

Even though sieving algorithms are asymptotically the fastest known algorithms for SVP, the memory requirement, in high dimension, has historically been a limiting factor to run these algorithms. Some recent works [18, 8] have shown how to use new tricks to make it possible to use sieving on high-dimensional lattices in practice and benefit from their efficient running time [57].

Nevertheless, it would be ideal and has been a long standing open question to obtain an algorithm that achieves the "best of both worlds", i.e. an algorithm that runs in time $2^{\mathcal{O}(n)}$ and requires memory polynomial in n. In the absence of such an algorithm, it is desirable to have a smooth tradeoff between time and memory requirement that interpolates between the current best sieving algorithms and the current best enumeration algorithms.

To this end, Bai, Laarhoven, and Stehlé [10] proposed the tuple sieving algorithm, providing such a tradeoff based on heuristic assumptions similar in nature to prior sieving algorithms. They conjectured a running time $k^{n+o(n)}$ and space complexity $k^{n/k+o(n)}$. One can vary the parameter k to obtain a smooth time/space tradeoff. Nevertheless, it is still desirable to obtain a provable variant of this algorithm that does not rely on any heuristics. The complexity of this algorithm was later proven, under the same heuristic assumptions [29], but only for constant k, therefore leaving the subexponential memory regime open.

Kirchner and Fouque [36] attempted to do this. They claim an algorithm for solving SVP in time $q^{\Theta(n)}$ and in space $q^{\Theta(n/q)}$ for any positive integer $q > 1$. Unfortunately, their analysis falls short of supporting their claimed result, and the correctness of the algorithm is not clear. We refer the reader to the full version of the paper for more details.

In addition to the above, Chen, Chung, and Lai [15] propose a variant of the algorithm based on Discrete Gaussian sampling in [2]. Their algorithm runs in time $2^{2.05n+o(n)}$ and the memory requirement is $2^{0.5n+o(n)}$. The quantum variant of their algorithm runs in time $2^{1.2553n+o(n)}$ time and has the same space complexity. Their algorithm has the best space complexity among known provably correct algorithms that run in time $2^{O(n)}$.

A number of works have also investigated the potential quantum speedups for lattice algorithms, and SVP in particular. A similar landscape to the classical one exists, although the quantum memory model has its importance. While quantum enumeration algorithms only require qubits [9], sieving algorithms require more powerful QRAMs [39, 37].

1.1 Our results

We first present a new algorithm for SVP that provides a smooth tradeoff between the time complexity and memory requirement of SVP without any heuristic assumptions. This algorithm is obtained by giving a new algorithm for sampling lattice vectors from the Discrete Gaussian distribution that runs in time $q^{\mathcal{O}(n)}$ and requires $q^{\mathcal{O}(n/q^2)}$ space.

▶ **Theorem 1** (Time-space tradeoff for smooth discrete Gaussian, informal). *There is an algorithm that takes as input a lattice $\mathcal{L} \subset \mathbb{R}^n$, a positive integer q, and a parameter s above the smoothing parameter of $\mathcal{L}$, and outputs q^{16n/q^2} samples from $D_{\mathcal{L},s}$ using $q^{13n+o(n)}$ time and poly$(q) \cdot q^{16n/q^2}$ space.*

Using the standard reduction from Bounded Distance Decoding (BDD) with preprocessing (where an algorithm solving the problem is allowed unlimited preprocessing time on the lattice before the algorithm receives the target vector) to Discrete Gaussian Sampling (DGS) from [16] and a reduction from SVP to BDD given in [15], we obtain the following.

▶ **Theorem 2** (Time-space tradeoff for SVP). *Let $n \in \mathbb{N}, q \in [4, \sqrt{n}]$ be a positive integer. Let $\mathcal{L}$ be the lattice of rank n. There is a randomized algorithm that solves SVP in time $q^{13n+o(n)}$ and in space poly$(n) \cdot q^{\frac{16n}{q^2}}$.*

If we take $k = q^2$, then the time complexity of the previous SVP algorithm becomes $k^{6.5n+o(n)}$ and the space complexity $\mathrm{poly}(n) \cdot k^{(8n/k)}$. Our tradeoff is thus the same (up to a constant in the exponents) as what was claimed by Kirchner and Fouque [36] and proven in [29] *under heuristic assumptions.*

Our second result is a quantum algorithm for SVP that improves over the current *fastest quantum algorithm* for SVP [2] (Notice that the algorithm in [2] is still the fastest classical algorithm for SVP).

▶ **Theorem 3** (Quantum Algorithm for SVP). *There is a quantum algorithm that solves SVP in $2^{0.9533n+o(n)}$ time and classical $2^{0.5n+o(n)}$ space with an additional number of qubits polynomial in n.*

Our third result is a classical algorithm for SVP that improves over the algorithm from [15] and results in the fastest classical algorithm that has a space complexity $2^{0.5n+o(n)}$.

▶ **Theorem 4** (Algorithm for SVP with $2^{0.5n+o(n)}$ space). *There is a classical algorithm that solves SVP in $2^{1.740n+o(n)}$ time and $2^{0.5n+o(n)}$ space.*

The time complexity of our second and third results are obtained using a quantity related to the kissing number of a lattice. A known upper bound of this quantity is $2^{0.402n}$, but in practice for most lattices, it can be much smaller and even $2^{o(n)}$. In that case, our classical algorithm runs in time $2^{1.292n}$ and our quantum algorithm runs in time $2^{0.750n}$. See Section 5 of the full version of the paper for more details [1].

We summarize known provable Classical and Quantum algorithms in Table 1. Note that all the classical algorithms are also quantum algorithms but they don't use any quantum power.

Table 1 Comparison of algorithms for the Shortest vector problem. [39] uses the quantum RAM model. [15] and our quantum algorithm need only polynomial qubits and $2^{0.5n+o(n)}$ classical space.

Classical Algorithms		
Time	**Space**	**Reference**
$n^{n+o(n)}$	poly(n)	[34]
$2^{n+o(n)}$	$2^{n+o(n)}$	[2]
$2^{2.05n+o(n)}$	$2^{0.5n+o(n)}$	[15]
$2^{1.741n+o(n)}$	$2^{0.5n+o(n)}$	This paper

Quantum Algorithms		
Time	**Space**	**Reference**
$2^{1.799n+o(n)}$	$2^{1.286n+o(n)}$, QRAM	[39]
$2^{1.2553n+o(n)}$	$2^{0.5n+o(n)}$	[15]
$2^{0.9533n+o(n)}$	$2^{0.5n+o(n)}$	This paper

▶ Remark 5 (Magic constants). Most of the constants that appear in this paper were calculated by optimising the complexity with respect to a quantity related to the kissing number and then instantiating with $b = 0.402$, the best known upper-bound on this quantity. The details of these calculations are available in the full version, section 5.

Roadmap

In the following, we give a high-level overview of our proofs in Section 1.2. Section 2 contain some preliminaries on lattices. The proofs of the time-space tradeoff for Discrete Gaussian sampling above the smoothing parameter and the time-space tradeoff for SVP are given in Section 3. Our classical and quantum algorithms for solving SVP with space complexity $2^{0.5n+o(n)}$ are presented in Section 4. We also shows how the time complexity of our algorithms varies with a quantity related to the kissing number in Section 5 of the full version of the paper [1].

1.2 Proof overview

We now include a high-level description of our proofs. Before describing our proof ideas, we emphasize that it was shown in [16, 2] that given an algorithm for DGS a constant factor c above the smoothing parameter, we can solve the problem of BDD where the target vector is within distance $\alpha\lambda_1(\mathcal{L})$ of the lattice, where the constant $\alpha < 0.5$ depends on the constant c. Additionally, using [15], one can enumerate all lattice points within distance $p\delta$ to a target $\boldsymbol{t}$ by querying p^n times a BDD oracle with decoding distance δ (or $p^{n/2}$ times if we are given a quantum BDD oracle). Thus, by choosing $p = \lceil \lambda_1(\mathcal{L})/\delta \rceil$ and $\boldsymbol{t} = \boldsymbol{0}$, an algorithm for BDD immediately gives us an algorithm for SVP. Therefore, it suffices to give an algorithm for DGS above the smoothing parameter.

1.2.1 Time-space tradeoff for DGS above smoothing

Recall that efficient algorithms are known for sampling from a discrete Gaussian with a large enough parameter (width) [38, 24, 12]. In [2], the authors begin by sampling $N = 2^{n+o(n)}$ vectors from the Discrete Gaussian distribution with (large) parameter s and then look for pairs of vectors whose sum is in $2\mathcal{L}$, or equivalently pairs of vectors that lie in the same coset $\boldsymbol{c} \in \mathcal{L}/2\mathcal{L}$. Since there are 2^n cosets, if we take $\Omega(2^n)$ samples from $D_{\mathcal{L},s}$, almost all of the resulting vectors (except at most 2^n vectors) will be paired and are statistically close to independent samples from the distribution $D_{\mathcal{L},s/\sqrt{2}}$, provided that the parameter s is sufficiently above the smoothing parameter.

To reduce the space complexity, we modify the algorithm by generating random samples and checking if the sum of d of those samples is in $q\mathcal{L}$ for some integer q. Intuitively, if we start with two lists of vectors (L_1 and L_2) of size $q^{\mathcal{O}(n/d)}$ from $D_{\mathcal{L},s}$, where s is sufficiently above the smoothing parameter, each of these vectors is contained in any coset $q\mathcal{L}+\boldsymbol{c}$ for any $\boldsymbol{c} \in \mathcal{L}/q\mathcal{L}$ with probability roughly $1/q^n$. We therefore expect that the coset of a uniformly random d-combination of vectors from L_2 is uniformly distributed in $\mathcal{L}/q\mathcal{L}$. The proof of this statement follows from the Leftover Hash Lemma [31]. We therefore expect that for any vector $\boldsymbol{v} \in L_1$, with high probability, there is a set of d vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d$ in L_2 that sum to a vector in $q\mathcal{L}+\boldsymbol{v}$, and hence $\frac{1}{q}\left(\sum_{i=1}^{d} \boldsymbol{x}_i - \boldsymbol{v}\right) \in \mathcal{L}$. A lemma by Micciancio and Peikert ([43]) shows that this vector is statistically close to a sample from the distribution $D_{\mathcal{L},s\sqrt{d+1}/q}$. We can find such a combination by trying all subsets of d vectors.

We would like to repeat this and find $q^{\mathcal{O}(n/d)}$ (nearly) independent vectors in $q\mathcal{L}$. It is not immediately clear how to continue since, in order to guarantee independence, one would not want to reuse the already used vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d$ and conditioned on the choice of these vectors, the distribution of the cosets containing the remaining vectors is disturbed and is no longer nearly uniform. By using a simple combinatorial argument, we show that even after removing any $1/\operatorname{poly}(d)$ fraction of vectors from the list L_2, the d-combination of vectors in L_2 has at least cq^n different cosets. This is sufficient to output $q^{\mathcal{O}(n/d)}$ independent vectors in $q\mathcal{L}$ with overwhelming probability.

1.2.2 A new algorithm for BDD with preprocessing leads to a faster quantum algorithm for SVP

This result improves the quantum algorithm from [15]. As mentioned above, a BDD oracle from discrete Gaussian sampling can have a decoding distance $\alpha\lambda_1(\mathcal{L})$ with $\alpha < 0.5$, and, using [15], one needs to enumerate all lattice points within distance $p\alpha\lambda_1(\mathcal{L})$ to a target $\boldsymbol{t}$ by querying p^n times a BDD oracle with decoding distance $\alpha\lambda_1(\mathcal{L})$ (or $p^{n/2}$ times if we are given a quantum BDD oracle). Hence, we need to take $p = 3$ so that $p\alpha\lambda_1(\mathcal{L}) \geqslant \lambda_1(\mathcal{L})$, and the search space is at least 3^n, or $3^{n/2}$ quantum queries. Thus, towards optimizing the algorithm for SVP, one should aim to solve α-BDD for α slightly larger than 1/3 since a larger value of α will still lead to the same running time for SVP. Using known bounds, it can be shown that such an algorithm requires $2^{0.1605n+o(n)}$ independent (preprocessed) samples from $D_{\mathcal{L},\eta_\varepsilon(\mathcal{L})}$[1] for $\varepsilon = 2^{-cn}$ for some constant c.

In [2], the authors gave an algorithm that runs in time $2^{n/2+o(n)}$ and outputs $2^{n/2+o(n)}$ samples from $D_{\mathcal{L},s}$ for any $s \geq \sqrt{2}\eta_{0.5}(\mathcal{L})$, i.e. a factor $\sqrt{2}$ above the smoothing parameter). In order to obtain samples at the smoothing parameter, we construct a dense lattice $\mathcal{L}'$ of smaller smoothing parameter than $\mathcal{L}$. We then sample $2^{0.5n+o(n)}$ vectors from $D_{\mathcal{L}',s}$ and reject those that are not in $\mathcal{L}$. Using the reduction from BDD to DGS, and by repeating this algorithm, we obtain a $2^{0.661n+o(n)}$ time and $2^{0.5n+o(n)}$-space algorithm to solve 1/3-BDD with preprocessing, where each call to BDD requires $2^{0.161n+o(n)}$ time. Thus, the total time complexity of the classical algorithm is $3^n \cdot 2^{0.161n+o(n)}$, and that of the corresponding quantum algorithm is $3^{n/2} \cdot 2^{0.161n+o(n)}$.

1.2.3 Covering surface of a ball by spherical caps

As we mentioned above, one can enumerate all lattice points within a $p\delta$ distance to a target $\boldsymbol{t}$ by querying p^n times a BDD oracle with decoding distance δ. Our algorithm for BDD is obtained by preparing samples from the discrete Gaussian distribution. However, note that the decoding distance of BDD oracle built by discrete Gaussian samples as shown in [16] is successful if the target vector is within a radius $\alpha\lambda_1(\mathcal{L})$ for $\alpha < 1/2$ (there is a tradeoff between α and the number of DGS samples needed), and therefore, if we choose $\boldsymbol{t}$ to be $\mathbf{0}$, as we do in the other algorithms mentioned above, then p has to be at least 3 to ensure that the shortest vector is one of the vectors output by the enumeration algorithm. We observe here that if we choose a target $\boldsymbol{t}$ to be a random vector "close to" but not at the origin, then the shortest vector will be within a radius 2δ from the target $\boldsymbol{t}$ with some probability P, and thus we can find the shortest vector by making $2^n/P$ calls to the BDD oracle. An appropriate choice of the target $\boldsymbol{t}$ and the factor α gives an algorithm that runs in time $2^n \cdot 2^{0.74n+o(n)}$, which is faster than the algorithm (running in time $3^n 2^{0.161n+o(n)}$) mentioned above.

We note that the corresponding quantum algorithm runs in time $2^{n/2} \cdot 2^{0.74n+o(n)}$, which is significantly slower than the quantum algorithm mentioned above.

We also note that the running time of this algorithm crucially depends on a quantity related to the kissing number of a lattice. Since a tight bound on this quantity is not known, the actual running time of this algorithm might be smaller than that promised above. For a more elaborate discussion on this, see Section 5 of the full version [1].

[1] The number of samples depends on a quantity related to the kissing number of a lattice, we used the best known upper bound on this quantity due to [33].

2 Preliminaries

Let $\mathbb{N} = \{1, 2, \ldots, \}$. We use bold letters $\boldsymbol{x}$ for vectors and denote a vector's coordinates with indices x_i. We use log to represent the logarithm base 2 and ln to represent the natural logarithm. Throughout the paper, n will always be the dimension of the ambient space $\mathbb{R}^n$.

Lattices

A *lattice* $\mathcal{L}$ is a discrete subgroup of $\mathbb{R}^n$, *i.e.* the set $\mathcal{L}(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) = \{\sum_{i=1}^m x_i \boldsymbol{b}_i \ : \ x_i \in \mathbb{Z}\}$ of all integer combinations of m linearly independent vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n \in \mathbb{R}^n$. Such $\boldsymbol{b}_i$'s form a *basis* of $\mathcal{L}$. The lattice $\mathcal{L}$ is said to be *full-rank* if $n = m$. We denote by $\lambda_1(\mathcal{L})$ the first minimum of $\mathcal{L}$, defined as the length of a shortest non-zero vector of $\mathcal{L}$.

For a rank n lattice $\mathcal{L} \subset \mathbb{R}^n$, the *dual lattice*, denoted $\mathcal{L}^*$, is defined as the set of all points in $\mathsf{span}(\mathcal{L})$ that have integer inner products with all lattice points, $\mathcal{L}^* = \{\vec{w} \in \mathsf{span}(\mathcal{L}) : \forall \vec{y} \in \mathcal{L}, \langle \vec{w}, \vec{y} \rangle \in \mathbb{Z}\}$. Similarly, for a lattice basis $\mathbf{B} = (\vec{b}_1, \ldots, \vec{b}_n)$, we define the dual basis $\mathbf{B}^* = (\vec{b}_1^*, \ldots, \vec{b}_n^*)$ to be the unique set of vectors in $\mathsf{span}(\mathcal{L})$ satisfying $\langle \vec{b}_i^*, \vec{b}_j \rangle = 1$ if $i = j$, and 0, otherwise. It is easy to show that $\mathcal{L}^*$ is itself a rank n lattice and $\mathbf{B}^*$ is a basis of $\mathcal{L}^*$. Given a lattice $\mathbf{B} = (\vec{b}_1, \ldots, \vec{b}_n)$, we denote $\| \mathbf{B} \|_2 = \max_i \|b_i\|$.

Probability distributions

Given two random variables X and Y on a set E, we denote by d_{SD} the *statistical distance* between X and Y, which is defined by

$$\mathrm{d}_{\mathrm{SD}}(X, Y) = \tfrac{1}{2} \sum_{z \in E} \Big| \Pr_X[X = z] - \Pr_Y[Y = z] \Big| = \sum_{\substack{z \in E : \\ \Pr_X[X=z] > \Pr_Y[Y=z]}} \Big(\Pr_X[X = z] - \Pr_Y[Y = z] \Big) .$$

We write X is ε-close to Y to denote that the statistical distance between X and Y is at most ε. Given a finite set E, we denote by U_E a uniform random variable on E, i.e., for all $x \in E$, $\Pr_{U_E}[U_E = x] = \frac{1}{|E|}$.

Discrete Gaussian Distribution

For any $s > 0$, define $\rho_s(\boldsymbol{x}) = \exp(-\pi \|\boldsymbol{x}\|^2 / s^2)$ for all $\boldsymbol{x} \in \mathbb{R}^n$. We write ρ for ρ_1. For a discrete set S, we extend ρ to sets by $\rho_s(S) = \sum_{\boldsymbol{x} \in S} \rho_s(\boldsymbol{x})$. Given a lattice $\mathcal{L}$, the *discrete Gaussian* $D_{\mathcal{L},s}$ is the distribution over $\mathcal{L}$ such that the probability of a vector $\boldsymbol{y} \in \mathcal{L}$ is proportional to $\rho_s(\boldsymbol{y})$: $\Pr_{X \sim D_{\mathcal{L},s}}[X = \boldsymbol{y}] = \frac{\rho_s(\boldsymbol{y})}{\rho_s(\mathcal{L})}$.

2.1 Lattice problems

The following problem plays a central role in this paper.

▶ **Definition 6.** *For $\delta = \delta(n) \geq 0$, σ a function that maps lattices to non-negative real numbers, and $m = m(n) \in \mathbb{N}$, δ-DGS$_\sigma^m$ (the Discrete Gaussian Sampling problem) is defined as follows: The input is a basis $\mathbf{B}$ for a lattice $\mathcal{L} \subset \mathbb{R}^n$ and a parameter $s > \sigma(\mathcal{L})$. The goal is to output a sequence of m vectors whose joint distribution is δ-close to m independent samples from $D_{\mathcal{L},s}$.*

We omit the parameter δ if $\delta = 0$, and the parameter m if $m = 1$. We stress that δ bounds the statistical distance between the *joint* distribution of the output vectors and m independent samples from $D_{\mathcal{L},s}$. We consider the following lattice problems.

▶ **Definition 7.** *The search problem SVP (Shortest Vector Problem) is defined as follows: The input is a basis* $\mathbf{B}$ *for a lattice* $\mathcal{L} \subset \mathbb{R}^n$*. The goal is to output a vector* $\boldsymbol{y} \in \mathcal{L}$ *with* $\|\vec{y}\| = \lambda_1(\mathcal{L})$*.*

▶ **Definition 8.** *The search problem CVP (Closest Vector Problem) is defined as follows: The input is a basis* $\mathbf{B}$ *for a lattice* $\mathcal{L} \subset \mathbb{R}^n$ *and a target vector* $\vec{t} \in \mathbb{R}^n$*. The goal is to output a vector* $\vec{y} \in \mathcal{L}$ *with* $\|\vec{y} - \vec{t}\| = \mathsf{dist}(\vec{t}, \mathcal{L})$*.*

▶ **Definition 9.** *For* $\alpha = \alpha(n) < 1/2$*, the search problem* α*-BDD (Bounded Distance Decoding) is defined as follows: The input is a basis* $\mathbf{B}$ *for a lattice* $\mathcal{L} \subset \mathbb{R}^n$ *and a target vector* $\vec{t} \in \mathbb{R}^n$ *with* $\mathsf{dist}(\boldsymbol{t}, \mathcal{L}) \leq \alpha \cdot \lambda_1(\mathcal{L})$*. The goal is to output a vector* $\vec{y} \in \mathcal{L}$ *with* $\|\vec{y} - \vec{t}\| = \mathsf{dist}(\vec{t}, \mathcal{L})$*.*

Note that while our other problems become more difficult as the approximation factor γ becomes smaller, α-BDD becomes more difficult as α gets larger.

For convenience, when we discuss the running time of algorithms solving the above problems, we ignore polynomial factors in the bit-length of the individual input basis vectors (i.e. we consider only the dependence on the ambient dimension n).

For a lattice $\mathcal{L}$ and $\varepsilon > 0$, the *smoothing parameter* $\eta_\varepsilon(\mathcal{L})$ is the smallest s such that $\rho_{1/s}(\mathcal{L}^*) = 1 + \varepsilon$. Recall that if $\mathcal{L}$ is a lattice and $\boldsymbol{v} \in \mathcal{L}$ then $\rho_s(\mathcal{L} + \boldsymbol{v}) = \rho_s(\mathcal{L})$ for all s. The *smoothing parameter* has the following well-known property.

▶ **Lemma 10** ([53, Claim 3.8]). *For any lattice* $\mathcal{L} \subset \mathbb{R}^n$*,* $\boldsymbol{c} \in \mathbb{R}^n$*,* $\varepsilon > 0$*, and* $s \geq \eta_\varepsilon(\mathcal{L})$*,*

$$\frac{1-\varepsilon}{1+\varepsilon} \leq \frac{\rho_s(\mathcal{L} + \boldsymbol{c})}{\rho_s(\mathcal{L})} \leq 1 \ .$$

▶ **Corollary 11** ([1, Corollary 10]). *Let* $\mathcal{L} \subset \mathbb{R}^n$ *be a lattice,* q *be a positive integer, and let* $s \geq \eta_\varepsilon(q\,\mathcal{L})$*. Let* C *be a random coset in* $\mathcal{L}/q\,\mathcal{L}$ *sampled such that* $\Pr[C = q\,\mathcal{L} + \boldsymbol{c}] = \frac{\rho_s(q\,\mathcal{L} + \boldsymbol{c})}{\rho_s(\mathcal{L})}$*. Also, let* U *be a coset in* $\mathcal{L}/q\,\mathcal{L}$ *sampled uniformly at random. Then* $\mathrm{d}_{SD}(C, U) \leq 2\varepsilon$ *.*

The following lemma gives a bound on the smoothing parameter.

▶ **Lemma 12** ([2, Lemma 2.7]). *For any lattice* $\mathcal{L} \subset \mathbb{R}^n, \varepsilon \in (0,1)$ *and* $k > 1$*, we have* $k\eta_\varepsilon(\mathcal{L}) > \eta_{\varepsilon^{k^2}}(\mathcal{L})$

Micciancio and Peikert [43] showed the following result about resulting distribution from the sum of many Gaussian samples.

▶ **Theorem 13** ([43, Theorem 3.3]). *Let* $\mathcal{L}$ *be an* n *dimensional lattice,* $\boldsymbol{z} \in \mathbb{Z}^m$ *a nonzero integer vector,* $s_i \geq \sqrt{2}\|\boldsymbol{z}\|_\infty \cdot \eta_\varepsilon(\mathcal{L})$*, and* $\mathcal{L} + \boldsymbol{c_i}$ *arbitrary cosets of* $\mathcal{L}$ *for* $i = 1 \cdots, m$*. Let* $\boldsymbol{y_i}$ *be independent vectors with distributions* $D_{\mathcal{L} + \boldsymbol{c_i}, s_i}$*, respectively. Then the distribution of* $\boldsymbol{y} = \sum\limits_i z_i \boldsymbol{y_i}$ *is* $m\varepsilon$ *close to* $D_{Y,s}$*, where* $Y = gcd(\boldsymbol{z})\,\mathcal{L} + \sum\limits_i z_i \boldsymbol{c_i}$*, and* $s = \sqrt{\sum (z_i s_i)^2}$*.*

We will need the following reduction from α-BDD to DGS that was shown in [16].

▶ **Theorem 14** ([16, Theorem 3.1], [2, Theorem 7.3]). *For any* $\varepsilon \in (0, 1/200)$*, let* $\phi(\mathcal{L}) \equiv \frac{\sqrt{\ln(1/\varepsilon)/\pi - o(1)}}{2\eta_\varepsilon(\mathcal{L}^*)}$*. Then, there exists a randomized reduction from* CVP^ϕ *to* 0.5*-*$\mathsf{DGS}^m_{\eta_\varepsilon}$*, where* $m = \mathcal{O}(\frac{n \log(1/\varepsilon)}{\sqrt{\varepsilon}})$ *and* CVP^ϕ *is the problem of solving* CVP *for target vectors that are guaranteed to be within a distance* $\phi(\mathcal{L})$ *of the lattice. The reduction preserves the dimension, makes a single call to the* DGS *oracle, and runs in time* $m \cdot poly(n)$*. Furthermore, the reduction always reduces an instance of* CVP^ϕ *on a lattice* $\mathcal{L}$ *to an instance of* DGS *on the dual lattice* $\mathcal{L}^*$*.*

We need the following relation between the first minimum of lattice and the smoothing parameter of dual lattice. We will use this to compute the decoding distance of BDD oracle.

▶ **Lemma 15** ([2, Lemma 6.1]). *For any lattice $\mathcal{L} \subset \mathbb{R}^n$, $\varepsilon \in (0,1)$, we have*

$$\sqrt{\frac{\ln(1/\varepsilon)}{\pi}} < \lambda_1(\mathcal{L})\eta_\varepsilon(\mathcal{L}^*) < \sqrt{\frac{\beta^2 n}{2\pi e}} \cdot \varepsilon^{-1/n} \cdot (1 + o(1)), \tag{1}$$

and if $\varepsilon \leq (e/\beta^2 + o(1))^{-\frac{n}{2}}$, we have

$$\sqrt{\frac{\ln(1/\varepsilon)}{\pi}} < \lambda_1(\mathcal{L})\eta_\varepsilon(\mathcal{L}^*) < \sqrt{\frac{\ln(1/\varepsilon) + n\ln\beta + o(n)}{\pi}}. \tag{2}$$

where $\beta(\mathcal{L}) = \inf\{\gamma : \forall r \geq 1,\ N(\mathcal{L}, r\lambda_1(\mathcal{L})) \leq (\gamma r)^n\}$. In particular $\beta(\mathcal{L}) \in [1, 2^{0.402}]$.

The following theorem proved in [15], is required to solve SVP by exponential number of calls to α-BDD oracle.

▶ **Theorem 16** ([15, Theorem 8]). *Given a basis matrix $\mathbf{B} \subset \mathbb{R}^{n\times n}$ for lattice $\mathcal{L}(\mathbf{B}) \subset \mathbb{R}^n$, a target vector $\boldsymbol{t} \in \mathbb{R}^n$, an α-BDD oracle BDD_α with $\alpha < 0.5$, and an integer scalar $p > 0$. Let $f_p^\alpha : \mathbb{Z}_p^n \to \mathbb{R}^n$ be $f_p^\alpha(\boldsymbol{s}) = -p \cdot BDD_\alpha(\mathcal{L}, (\mathbf{B}\,\boldsymbol{s} - \boldsymbol{t})/p) + \mathbf{B}\,\boldsymbol{s}$. If $\mathsf{dist}(\mathcal{L}, \boldsymbol{t}) \leq \alpha\lambda_1(\mathcal{L})$, then the list $m = \{f_p^\alpha(\boldsymbol{s}) \mid \boldsymbol{s} \in \mathbb{Z}_p^n\}$ contains all lattice points within distance $p\alpha\lambda_1(\mathcal{L})$ to $\boldsymbol{t}$.*

We will need the following theorems to sample the DGS vectors with a large width.

▶ **Theorem 17** ([2],Proposition 2.17). *For any $\varepsilon \leq 0.99$, there is an algorithm that takes as input a lattice $\mathcal{L} \in \mathbb{R}^n$, $M \in \mathbb{Z}_{>0}$ (the desired number of output vectors), and $s > 2^{n\log\log n/\log n} \cdot \eta_\varepsilon(\mathcal{L})$, and outputs M independent samples from $D_{\mathcal{L},s}$ in time $M \cdot poly(n)$.*

▶ **Theorem 18** ([2, Theorem 5.11]). *For a lattice $\mathcal{L} \subset \mathbb{R}^n$, let $\sigma(\mathcal{L}) = \sqrt{2}\eta_{1/2}(\mathcal{L})$. Then there exists an algorithm that solves $\exp(-\Omega(\kappa))$-$DGS_\sigma^{2^{n/2}}$ in time $2^{n/2+\mathrm{polylog}(\kappa)+o(n)}$ with space $\mathcal{O}(2^{n/2})$ for any $\kappa \geq \Omega(n)$. Moreover, if the input does not satisfy the promise, and the input parameter $s < \sigma(\mathcal{L}) = \sqrt{2}\eta_{1/2}(L)$, then the algorithm may output M vectors for some $M \leq 2^{n/2}$ that are $\exp(-\Omega(\kappa))$-close to M independent samples from $D_{\mathcal{L},s}$.*

▶ **Lemma 19** ([2, Lemma 5.12]). *There is a probabilistic polynomial-time algorithm that takes as input a lattice $\mathcal{L} \subset \mathbb{R}^n$ of rank n and an integer a with $n/2 \leq a < n$ and returns a super lattice $\mathcal{L}' \supset \mathcal{L}$ of index 2^a with $\mathcal{L}' \subseteq \mathcal{L}/2$ such that for any $\varepsilon \in (0,1)$, we have $\eta_{\varepsilon'}(\mathcal{L}') \leq \eta_\varepsilon(\mathcal{L})/\sqrt{2}$ with probability at least $1/2$ where $\varepsilon' := 2\varepsilon^2 + 2^{(n/2)+1-a}(1+\varepsilon)$.*

2.2 Probability

We need the following lemma on distribution of vector inner product which directly follows from the Leftover Hash Lemma [31].

▶ **Lemma 20.** *Let $\mathbb{G}$ be a finite abelian group, and let f be a positive integer. Let $\mathcal{Y} \subseteq \{0,1\}^f$. Define the inner product $\langle\cdot,\cdot\rangle : \mathbb{G}^f \times \mathcal{Y} \to \mathbb{G}$ by $\langle x, y\rangle = \sum_i x_i y_i$ for all $x \in \mathbb{G}^f, y \in \mathcal{Y}$. Let X, Y be independent and uniformly random variables on $\mathbb{G}^f, \mathcal{Y}$, respectively. Then $\mathrm{d}_{SD}((\langle X, Y\rangle, X), (U_\mathbb{G}, X)) \leq \frac{1}{2} \cdot \sqrt{\frac{|\mathbb{G}|}{|\mathcal{Y}|}}$, where $U_\mathbb{G}$ is uniform in $\mathbb{G}$ and independent of X.*

We will also need the Chernoff-Hoeffding bound [30].

▶ **Lemma 21.** *Let $X_1, \ldots, X_M$ be the independent and identically distributed random boolean variables of expectation p. Then for $\varepsilon > 0$, $\Pr\left[\frac{1}{M}\sum\limits_{i=1}^{M} X_i \leq p(1-\delta)\right] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{pM}$.*

For preliminaries on quantum computing, see [1, Section 2.2]

3 Algorithms with a time-memory tradeoff for lattice problems

In this section, we present a new algorithm for Discrete Gaussian sampling above the smoothing parameter.

3.1 Algorithm for Discrete Gaussian Sampling

We now present the main result of this section.

▶ **Theorem 22.** *Let $n \in \mathbb{N}, q \geq 2, d \in [1,n]$ be positive integers, and let $\varepsilon > 0$. Let C be any positive integer. Let $\mathcal{L}$ be a lattice of rank n, and let $s \geq 2\sqrt{d}\eta_\varepsilon(q\,\mathcal{L}) = 2\sqrt{d}q\eta_\varepsilon(\mathcal{L})$. There is an algorithm that, given $N = 160d^2 \cdot C \cdot q^{n/d}$ independent samples from $D_{\mathcal{L},s}$, outputs a list of vectors that is $(10d\varepsilon^{2d}N + 11Cq^{-5n/2})$-close to $Cq^{n/d}$ independent vectors from $D_{\mathcal{L},\frac{\sqrt{8d+1}}{q}s}$. The algorithm runs in time $C \cdot (10e \cdot d)^{8d} \cdot q^{8n+n/d+o(n)}$ and requires memory $\mathrm{poly}(d) \cdot q^{n/d}$ excluding the input and output memory.*

Proof. We prove the result for $C = 1$, and the general result follows by repeating the algorithm. Let $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ be the N input vectors and let $\{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_N\}$ be the corresponding cosets in $\mathcal{L}/q\,\mathcal{L}$. The algorithm does the following:

1. Initialize two lists $L_1 = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\frac{N}{2}}\}$ and $L_2 = \{\boldsymbol{x}_{\frac{N}{2}+1}, \ldots, \boldsymbol{x}_N\}$ each with $\frac{N}{2}$ input vectors, and let $Q = 0$.
2. Let $\boldsymbol{v}$ be the first vector in L_1.
3. Find $8d$ vectors (by trying all $8d$-tuples) $\boldsymbol{x}_{i_1}, \ldots, \boldsymbol{x}_{i_{8d}}$ from L_2 such that $\boldsymbol{c}_{i_1} + \cdots + \boldsymbol{c}_{i_{8d}} - \boldsymbol{v} \in q\,\mathcal{L}$. If no such vectors exist go to step(6).
4. Output the vector $\frac{\boldsymbol{x}_{i_1}+\cdots+\boldsymbol{x}_{i_{8d}}-\boldsymbol{v}}{q} \in \mathcal{L}$, and let $Q = Q + 1$. If $Q = q^{n/d}$, then END.
5. Remove vectors $\boldsymbol{x}_{i_1}, \cdots, \boldsymbol{x}_{i_{8d}}$ from L_2
6. Remove vector $\vec{v}$ from L_1 and repeat Steps (2) to (5).

The time complexity of the algorithm is $\frac{N}{2} \cdot \binom{N/2}{8d} \leq \frac{N}{2}\left(\frac{eN}{16d}\right)^{8d} \leq (10e \cdot d)^{8d} \cdot q^{8n+n/d+o(n)}$, and memory requirement of the algorithm is immediate. We now show correctness. Let $\varepsilon' = \varepsilon^{2d}$ so that $s \geq \sqrt{2}\eta_{\varepsilon'}(q\,\mathcal{L})$ by Lemma 12. Without loss of generality, we can assume that the vectors $\boldsymbol{x}_i$ for $i \in [N]$ are sampled by first sampling $\boldsymbol{c}_i \in \mathcal{L}/q\,\mathcal{L}$ such that $\Pr[\boldsymbol{c}_i = \boldsymbol{c}] = \Pr[D_{\mathcal{L},s} \in q\,\mathcal{L} + \boldsymbol{c}]$ and then sampling the vector $\boldsymbol{x}_i$ according to $D_{q\,\mathcal{L}+\boldsymbol{c}_i,s}$. Moreover, by Corollary 11, this distribution is $2\varepsilon' N$-close to sampling $\boldsymbol{c}_i$ for $i \in [N]$, independently and uniformly from $\mathcal{L}/q\,\mathcal{L}$, and then sampling the vectors $\boldsymbol{x}_i$ according to $D_{q\,\mathcal{L}+\boldsymbol{c}_i,s}$. We now assume that the input is sampled from this distribution.

Without loss of generality, we can assume that the algorithm initially gets only the corresponding cosets as input, and the vectors $\boldsymbol{x}_{i_j} \in q\,\mathcal{L} + \boldsymbol{c}_{i_j}$ for $j \in [8d]$, and $\boldsymbol{v} \in q\,\mathcal{L} + \boldsymbol{c}$ are sampled from $D_{q\,\mathcal{L}+\boldsymbol{c}_{i_j},s}$ and $D_{q\,\mathcal{L}+\boldsymbol{c},s}$ only before such a tuple is needed in Step 4 of the algorithm. Since any input vector is used only once in Step 4, these samples are independent of all prior steps. This implies, by Theorem 13, that the vector obtained in Step 4 of the algorithm is $\varepsilon'(8d+1)$-close to being distributed as $D_{\mathcal{L},s\frac{\sqrt{8d+1}}{q}}$.

It remains to show that our algorithm finds $q^{n/d}$ vectors (with high probability). Let $N' = \frac{N}{2}$ be an integer, X be a random variable uniform over $(\mathcal{L}/q\,\mathcal{L})^{N'}$, and let Y be a random variable independent of X and uniform over vectors in $\{0,1\}^{N'}$ with Hamming weight $8d$. The number of such vectors is

$$\binom{N'}{8d} \geq \left(\frac{N'}{8d}\right)^{8d} \geq q^{8n} \,. \tag{3}$$

Let U be a uniformly random coset of $\mathcal{L}/q\mathcal{L}$. By Lemma 20 and (3), we have

$$\mathrm{d_{SD}}((\langle X, Y\rangle, X), (U, X)) \leq \frac{1}{2} \cdot \sqrt{\frac{q^n}{q^{8n}}} < q^{-7n/2} ,$$

for a large enough value of n. By Markov inequality, with probability greater than $1 - (10 \cdot q^{-5n/2})$ over the choice of $x \leftarrow X$, we have that the statistical distance between $\langle x, Y\rangle$ and U is less than $\frac{q^{-n}}{10}$, which implies for any $\boldsymbol{v} \in \mathcal{L}/q\mathcal{L}$,

$$q^{-n} + \frac{q^{-n}}{10} > \Pr[\langle x, Y\rangle = \boldsymbol{v} \mod q\mathcal{L}] > q^{-n} - \frac{q^{-n}}{10}. \tag{4}$$

We assume that the input vectors in list L_2 satisfy (4), introducing a statistical distance of at most $10 \cdot q^{-5n/2}$. Notice that after the algorithm found i vectors for any $i < q^{n/d}$, it has removed $8id$ vectors from L_2. We will show that for each vector from L_1 (which is uniformly sampled from $\mathcal{L}/q\mathcal{L}$) with constant probability we will find $8d$-vectors in Step (3).

After $i < q^{n/d}$ output vectors have been found, there are $M = N' - 8id$ vectors remaining in the list L_2. There are $\binom{M}{8d}$ different $8d$-combinations possible with vectors remaining in L_2.

$$\binom{N'}{8d}/\binom{M}{8d} = \frac{N' \cdots (N' - 8d + 1)}{M \cdots (M - 8d + 1)} < \left(\frac{N'}{N' - 8d(i+1)}\right)^{8d} \leqslant \left(1 + \frac{8dq^{n/d}}{N' - 8dq^{n/d}}\right)^{8d}$$
$$= \left(1 + \frac{1}{10d - 1}\right)^{8d} < \frac{5}{2} \quad \text{since } N' = 80d^2q^{n/d} \text{ for } C = 1 \tag{5}$$

At the beginning of the algorithm, there are $\binom{N'}{8d}$ combinations, and hence by (4), each of the q^n cosets appears at least $0.9q^{-n}\binom{N'}{8d}$ times. After $i < q^{n/d}$ output vectors have been found, there are only $\binom{M}{8d}$ combinations left, and $\binom{N'}{8d} - \binom{M}{8d}$ possible combinations have been removed. We say that a coset $\boldsymbol{c}$ *disappears* if there is no set of $8d$ vectors in L_2 that add to $\boldsymbol{c}$. In order for a coset to disappear, all of the at least $0.9q^{-n}\binom{N'}{8d}$ combinations from the initial list must be removed. Hence, the number of cosets that disappear is at most $\frac{\binom{N'}{8d} - \binom{M}{8d}}{0.9q^{-n}\binom{N'}{8d}} < \frac{3/5}{0.9}q^n = \frac{2}{3}q^n$ distinct cosets by (5). Hence with probability at least $1/3$, we find $8d$ vectors $\boldsymbol{x}_{i_1}, \ldots, \boldsymbol{x}_{i_{8d}}$ from L_2 such that $\boldsymbol{x}_{i_1} + \cdots + \boldsymbol{x}_{i_{8d}} - \boldsymbol{v} \in q\mathcal{L}$. By Chernoff-Hoeffding bound with probability greater than $1 - e^{-d^2q^{n/d}}$, the algorithm finds at least $q^{n/d}$ vectors. In total, the statistical distance from the desired distribution is

$$(8d + 1)\varepsilon' \cdot N + 2\varepsilon' q^{n/d} + 10 \cdot q^{-5n/2} + e^{-d^2q^{n/d}} \leq 10d\varepsilon' \cdot N + 11 \cdot q^{-5n/2}. \qquad ◀$$

▶ **Corollary 23.** *Let $n \in \mathbb{N}$, $q \in [4, \sqrt{n}]$ be an integer, and let $\varepsilon = q^{-32n/q^2}$. Let $\mathcal{L}$ be a lattice of rank n, and let $s \geq \eta_\varepsilon(\mathcal{L})$. There is an algorithm that outputs a list of vectors that is $q^{-\Omega(n)}$-close to q^{16n/q^2} independent vectors from $D_{\mathcal{L},s}$. The algorithm runs in time $q^{13n+o(n)}$ and requires memory* $\mathrm{poly}(n) \cdot q^{16n/q^2}$.

Proof. Choose d so that $16d - 16 < q^2 \leqslant 16d$, which is possible when $q \geqslant 4$, and let $\alpha = q/\sqrt{8d+1}$ – this is the ratio by which we decrease the Gaussian width in Theorem 22 – and note that $\alpha \geq 1.2$.

Let $p = \lceil 2\sqrt{d}q \rceil < q^2$ and k be the smallest integer such that $\alpha^k \cdot p \geq 2^{n \log\log n/\log n}$. Thus $k = O(n \log\log n/\log n)$. Let $g = \alpha^k ps \geq 2^{n \log\log n/\log n} \cdot \eta_\varepsilon(\mathcal{L})$. By Theorem 17, in time $N_0 \cdot \mathrm{poly}(n)$, we get $N_0 = (160d^2)^k q^{n/d}$ samples from $D_{\mathcal{L},g}$.

We now iterate k times the algorithm from Theorem 22. Initially we have N_0 vectors. At the beginning of the i-th iteration for $i \leq k-1$, we have $N_i := N_0 \cdot (160d^2)^{-i}$ vectors that are Δ_i-close to being independently distributed from $D_{\mathcal{L},\alpha^{-i}g}$, where $\alpha^{-i}g \geqslant \alpha p \cdot \eta_\varepsilon(\mathcal{L})$. Hence, we can apply Theorem 22 and get $N_{i+1} = N_i/160d^2$ vectors that are Δ_{i+1}-close to being independently distributed from $D_{\mathcal{L},\alpha^{-(i+1)}g}$, where $\Delta_{i+1} \leqslant \Delta_i + 4\varepsilon^{2d}N_i + 11(160d^2)^{k-i}q^{-5n/2}$. At each iteration we had $N_i \geq 160d^2q^{n/d}$ vectors, a necessary condition to apply Theorem 22. Therefore after k iterations, we have at least $N_k = N_0/(160d^2)^k = q^{n/d}$ samples that are Δ_k-close to being independently distributed from $D_{\mathcal{L},\alpha^{-k}g}$, where

$$\begin{aligned}\Delta_k &\leqslant 11q^{-5n/2}\sum_{i=1}^{k}(160d^2)^{k-i} + \sum_{i=0}^{k-1} 10d\varepsilon^{2d}N_i \\ &\leq 11(160d^2)^k q^{-5n/2} + 10dq^{-4n}q^{n/d}\sum_{i=0}^{k-1}(160d^2)^{k-i} && \text{since } 16d \geqslant q^2 \\ &\leq \left(11q^{-5n/2} + 10dq^{-4n+n/d}\right)(160d^2)^{k+1} = q^{-5n/2+o(n)} && \text{since } (160d^2)^{k+1} = q^{o(n)}.\end{aligned}$$

Any vector distributed as $D_{\mathcal{L},ps}$ is in $p\mathcal{L}$ with probability at least p^{-n}. We repeat the algorithm $2p^n = O(q^{2n})$ times to obtain $p^n \cdot 2 \cdot q^{n/d}$ vectors that are $2p^nq^{-5n/2+o(n)} = q^{-n/2+o(n)}$ close to $2p^n \cdot q^{n/d}$ independent samples from $D_{\mathcal{L},ps}$. Of these samples obtained, we only keep vectors that fall in $p\mathcal{L}$ and divide them by p. Let $M = p^n \cdot 2 \cdot q^{n/d}$. By Chernoff-Hoeffding (Lemma 21) with $P = p^{-n}$, and $\delta = \frac{1}{2}$, the probability to obtain less than $(1-\delta)PM = q^{n/d}$ samples is at most $\left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{PM} \leqslant e^{-\frac{1}{10}q^{n/d}}$. Furthermore, $d \leqslant \frac{q^2+16}{16}$ and $q \mapsto \frac{\ln q}{16+q^2}$ is decreasing for $q \geqslant 4$, hence for $q \leqslant \sqrt{n}$,

$$q^{n/d} \geqslant e^{16n\frac{\ln q}{16+q^2}} \geqslant e^{16n\frac{\ln\sqrt{n}}{16+n}} \geqslant e^{16\ln\sqrt{n}-o(1)} = \Omega(n^8).$$

Hence with probability greater than $1 - e^{-\frac{1}{10}q^{n/d}} = 1 - q^{-\Omega(n^8)}$, we get $q^{n/d}$ vectors from the distribution $D_{\mathcal{L},s}$. The statistical distance from the desired distribution is $q^{-\Omega(n^8)} + q^{-n/2+o(n)} \leq q^{-n/2+o(n)}$. We repeat this for $\frac{q^{16n/q^2}}{q^{n/d}}$ times, to get q^{16n/q^2} vectors. The total statistical distance from the desired distribution is $\frac{q^{16n/q^2}}{q^{n/d}} \cdot q^{-n/2+o(n)} \leq q^{-\Omega(n)}$. The total running time is bounded by

$$q^{2n}\left(\frac{q^{16n/q^2}}{q^{n/d}}\right)\left(\operatorname{poly}(n)\cdot N_0 + \sum_{i=0}^{k-1}(10ed)^{8d}\cdot(160d^2)^{k-i}q^{8n+n/d+o(n)}\right) \leqslant q^{13n+o(n)}.$$

The memory usage is slightly more involved: we can think of the k iterations as a pipeline with k intermediate lists and we observe that as soon as a list (at any level) has more than $160d^2q^{16n/q^2}$ elements, we can apply Theorem 22 to produce q^{16n/q^2} vectors at the next level. Hence, we can ensure that at any time, each level contains at most $160d^2q^{16n/q^2}$ vectors, so in total we only need to store at most $k \cdot 160d^2q^{16n/q^2} = \operatorname{poly}(n)q^{16n/q^2}$ vectors, to which we add the memory usage of the algorithm of Theorem 22 which is bounded by $\operatorname{poly}(n)\cdot q^{n/d} \leqslant \operatorname{poly}(n)\cdot q^{16n/q^2}$. Finally, we run the filter ($p\mathcal{L}$) on the fly at the end of the k iterations to avoid storing useless samples. ◀

This tradeoff works for any $q \geq 4$, and the running time can be bounded by $c_1^{n+o(n)} \cdot q^{c_2 n}$ for some constants c_1 and c_2 that we have not tried to optimize.

3.2 Algorithms for BDD and SVP

▶ **Theorem 24.** *Let $n \in \mathbb{N}, q \in [4, \sqrt{n}]$ be a positive integer. Let $\mathcal{L}$ be a lattice of rank n, there exists an algorithm that creates a $0.1/q$-BDD oracle in time $q^{13n+o(n)}$ and space $\operatorname{poly}(n)\cdot q^{16n/q^2}$. Every call to this oracle takes time $\operatorname{poly}(n)q^{16n/q^2}$ time.*

Proof. Let $\varepsilon = q^{\frac{-32n}{q^2}}$ and $s = \eta_\varepsilon(\mathcal{L}^*)$. From corollary 23, there exists an algorithm that outputs q^{16n/q^2} vectors whose distribution is statistically close to time $D_{\mathcal{L}^*,s}$ in $q^{13n+o(n)}$ and space $\mathrm{poly}(n) \cdot q^{16n/q^2}$.

By Theorem 14, there is a reduction from α-BDD to $\frac{1}{2}$-DGS$^m_{\eta_\varepsilon}$ with $m = \mathcal{O}(\frac{n\log(1/\varepsilon)}{\sqrt{\varepsilon}}) = \mathcal{O}(\frac{n^2}{q^2}q^{16n/q^2})$, where the decoding coefficient is $\alpha = \frac{\sqrt{\log(1/\varepsilon)/\pi - o(1)}}{2\eta_\varepsilon(\mathcal{L}^*)\lambda_1(\mathcal{L})}$. By repeating $\mathrm{poly}(n)$ times the algorithm from Corollary 23, we get m vectors from $D_{\mathcal{L}^*,\eta_\varepsilon(\mathcal{L}^*)}$. By Lemma 15, we get

$$\alpha(\mathcal{L}) = \frac{\sqrt{\log(1/\varepsilon)/\pi - o(1)}}{2\eta_\varepsilon(\mathcal{L}^*)\lambda_1(\mathcal{L})} \geq \sqrt{\frac{\log(1/\varepsilon)}{2n(\beta^2/e)\varepsilon^{-1/n}}} \cdot (1-o(1)) \geq \frac{1}{q}\sqrt{\frac{32 \cdot e \cdot \log q}{2\beta^2 q^{32/q^2}}} \geq (10q)^{-1}.$$

Note that here we are using the fact that the reduction in Theorem 14 always reduces an instance on a lattice $\mathcal{L}$ to an instance on the dual lattice $\mathcal{L}^*$: this is why we generate samples from $D_{\mathcal{L}^*,\eta_\varepsilon(\mathcal{L}^*)}$ in the preprocessing phase, even before any call to the oracle is made. Finally, by Theorem 14, each call to the oracle takes time $m \cdot \mathrm{poly}(n) = \mathcal{O}(q^{16n/q^2}\,\mathrm{poly}(n))$. ◀

▶ **Theorem 25.** *Let $n \in \mathbb{N}, q \in [4, \sqrt{n}]$ be a positive integer. Let $\mathcal{L}$ be a lattice of rank n. There is a randomized algorithm that solves SVP in time $q^{13n+o(n)}$ and in space* $\mathrm{poly}(n) \cdot q^{\frac{16n}{q^2}}$.

Proof. By Theorem 24, we can construct a $\frac{0.1}{q}$-BDD oracle in time $q^{13n+o(n)}$ and in space $\mathrm{poly}(n) \cdot q^{\frac{16n}{q^2}}$. Each execution of the BBD oracle now takes $\mathcal{O}(\mathrm{poly}(n)q^{16n/q^2})$ time. By Theorem 16, with $(10q)^n$ queries to $\frac{0.1}{q}$-BDD oracle, we can find the shortest vector. The total time complexity is $q^{13n+o(n)} + \mathrm{poly}(n)q^{16n/q^2} \cdot (10q)^n = q^{13n+o(n)}$. ◀

▶ Remark 26. If we take $q = \sqrt{n}$, Theorem 25 gives a SVP algorithm that takes $n^{\mathcal{O}(n)}$ time and $\mathrm{poly}(n)$ space. The constant in the exponent of time complexity is worse than the best enumeration algorithms. When q is a large enough constant, for any constant $\varepsilon > 0$, there exists a constant $C = C(\varepsilon) > 2$, such that there is a 2^{Cn} time and $2^{\varepsilon n}$ space algorithm for DGS, and SVP. In particular, the time complexity of the algorithm in this regime is worse than the best sieving algorithms.

4 New space efficient algorithms for SVP

In this section, we present relatively space-efficient classical and quantum algorithms to find a shortest nonzero lattice vector. Our quantum algorithm is the first provable algorithm for exact-SVP that takes less than $\mathcal{O}(2^n)$ time. Recall that there exists an algorithm [15] that, given a lattice $\mathcal{L}$ and a target vector $\boldsymbol{t}$, outputs all lattice vectors within distance $p\alpha\lambda_1(\mathcal{L})$ to $\boldsymbol{t}$, by making p^n calls to an α-BDD oracle. We present a quantum algorithm for SVP that takes $2^{0.9532n+o(n)}$ time and $2^{0.5n+o(n)}$ space with $\mathrm{poly}(n)$ qubits. We also present a classical algorithm for SVP that takes $2^{1.741n+o(n)}$ time and $2^{0.5n+o(n)}$ space.

The strategy followed by [15] is to choose $p = \lceil 1/\alpha \rceil$, the target vector $\boldsymbol{t}$ to be the origin, and sequentially compute the candidate vectors for SVP. There are two ways to reduce the time complexity: one can improve the BDD oracle or reduce the number of queries. We will show how to improve both aspects.

4.1 Quantum algorithm for SVP

In order to solve SVP by the method in [15], it is sufficient to use a BDD oracle with decoding coefficient α slightly greater than $1/3$. In [15], the authors use a reduction from BDD to DGS by [16] and use the Gaussian sampler of [2] to obtain many samples with standard deviation equal to $\sqrt{2}\eta_{1/2}$. This allows them to construct a 0.391-BDD but each call to the BDD oracle uses many DGS samples. This is wasteful since we really only need a 1/3-BDD. The reason why it is so expensive is that in the analysis they need to find ε such that $\eta_\varepsilon > \sqrt{2}\eta_{1/2}$ to apply the reduction, and it requires them to take ε much smaller than would be strictly necessary to construct a 1/3-BDD oracle; this smaller ε explains the bigger decoding radius.

We obtain a BDD oracle with decoding distance 1/3 by using the same reduction but making each call cheaper. This is achieved by building a sampler that directly samples at the smoothing parameter, hence avoiding the $\sqrt{2}$ factor, allowing us to take a bigger ε. In [2], it was shown how to construct a dense lattice $\mathcal{L}'$ whose smoothing parameter $\eta(\mathcal{L}')$ is $\sqrt{2}$ times smaller than the original lattice, and that contains all lattice points of the original lattice. Suppose that we first use such a dense lattice to construct a corresponding discrete Gaussian sampler with standard deviation equal to $s = \sqrt{2}\eta(\mathcal{L}')$. We then do the rejection sampling on condition that the output is in the original lattice $\mathcal{L}$. We thus have constructed a discrete Gaussian sampler of $\mathcal{L}$ whose standard deviation is $\sqrt{2}\eta(\mathcal{L}') = \eta(\mathcal{L})$. Nevertheless, $|\,\mathcal{L}'\,/\,\mathcal{L}\,|$ will be at least $2^{0.5n}$, which implies that this procedure needs at least $2^{0.5n}$ input vectors to produce an output vector. We use this idea to obtain the following lemma.

► **Lemma 27.** *There is an probabilistic algorithm that, given a lattice $\mathcal{L} \subset \mathbb{R}^n$, $m \in \mathbb{Z}_+$ and $s \geq \eta_{1/3}(\mathcal{L})$ as input, outputs m samples from a distribution $(m \cdot 2^{-\Omega(n^2)})$-close to $D_{\mathcal{L},s}$ in expected time $m \cdot 2^{(n/2)+o(n)}$ and $(m + 2^{n/2}) \cdot 2^{o(n)}$ space.*

Proof. Let $a = \frac{n}{2} + 4$. We repeat the following until we output m vectors. We use the algorithm in Lemma 19 to obtain a lattice $\mathcal{L}' \supset \mathcal{L}$ of index 2^a. We then run the algorithm from Theorem 18 with input $(\mathcal{L}', s)$ to obtain a list of vectors from $\mathcal{L}'$. We output the vectors in this list that belong to $\mathcal{L}$.

By Theorem 18, we obtain, in time and space $2^{(n/2)+o(n)}$, $M = 2^{n/2}$ vectors that are $2^{-\Omega(n^2)}$-close to M vectors independently sampled from $D_{\mathcal{L}',s}$. Also, by Lemma 19, with probability at least 1/2, we have $s \geq \eta_{1/3}(\mathcal{L}) \geq \sqrt{2}\eta_{1/2}(\mathcal{L}')$.

From these M vectors, we will reject the vectors which are not in lattice $\mathcal{L}$. It is easy to see that the probability that a vector sampled from the distribution $D_{\mathcal{L}',s}$ is in $\mathcal{L}$ is at least $\rho_s(\mathcal{L})/\rho_s(\mathcal{L}') \geq \frac{1}{2^a}$ using Lemma 10. Thus, the probability that we obtain at least one vector from $\mathcal{L}$ (which is distributed as $D_{\mathcal{L},s}$) is at least

$$\frac{1}{2}\left(1 - (1 - 1/2^a)^{2^{n/2}}\right) \geq \frac{1}{2} \cdot \left(1 - e^{-2^{n/2}/2^{n/2+4}}\right) = \frac{1}{2}(1 - e^{-1/16}).$$

It implies that after rejection of vectors, with constant probability we will get at least one vector from $D_{\mathcal{L},s}$. Thus, the expected number of times we need to repeat the algorithm is $O(m)$ until we obtain vectors $\boldsymbol{y_1}, \ldots, \boldsymbol{y_m}$ whose distribution is statistically close to being independently distributed from $D_{\mathcal{L},s}$. The time and space complexity is clear from the algorithm. ◄

► **Theorem 28.** *For any sufficiently large integer n, any integer $m > 0$, and a lattice $\mathcal{L} \subset \mathbb{R}^n$, there exists an algorithm that creates a 1/3-BDD oracle in $2^{0.6608n+o(n)}$ time and $2^{0.5n+o(n)}$ space. Every call to this oracle takes $2^{0.1608n+o(n)}$ time and space.*

Proof. See full version [1, Theorem 30]; it is similar to Theorem 24 but using Lemma 27. ◄

From [15], we can enumerate all vectors of length $p \cdot \frac{1}{3}\lambda_1(\mathcal{L})$ by making p^n calls to 1/3-BDD oracle. Although naively searching for the minimum in the set of vectors of length less than or equal to $p \cdot \frac{1}{3}\lambda_1(\mathcal{L})$, will find the origin with high probability, one can work around this issue by shifting the zero vector. Choosing an arbitrary nonzero lattice vector as the shift, we are guaranteed to obtain a vector of length at least λ_1 for $p \geq 3$. Hence by combining the 1/3-BDD oracle from Theorem 28 and the quantum minimum finding algorithm from [19, Theorem 1], we can find the shortest vector. Note that, we can directly use the quantum speedup construction from [15]. The following theorem is a simplified construction for the quantum algorithm.

▶ **Theorem 29.** *For any $n \geq 5$, there is a quantum algorithm that solves SVP in time $2^{0.9533n+o(n)}$ and classical-space $2^{0.5n+o(n)}$ with polynomial number of qubits.*

Proof. Let $\mathbf{B}$ be a basis of the lattice, $\mathsf{BDD}_{1/3}$ be a 1/3-BDD oracle and let $f : \mathbb{Z}_3^n \to \mathcal{L}$ be $f(\boldsymbol{s}) = -3 \cdot \mathsf{BDD}_{1/3}(\mathcal{L}, (\boldsymbol{Bs})/3) + \boldsymbol{Bs}$. The algorithm works on three quantum registers and our goal is to build a superposition of states of the form $|\boldsymbol{s}\rangle|f(\boldsymbol{s})\rangle|x\rangle$ where $x = \|f(\boldsymbol{s})\|$ most the time (see the definition of U below). The algorithm goes like this, we first use Theorem 28 to construct a quantum oracle O_{BDD} on the first two registers that satisfies $O_{\mathsf{BDD}}|\boldsymbol{s}\rangle|\mathbf{0}\rangle = |\boldsymbol{s}\rangle|f(\boldsymbol{s})\rangle$ for all $\boldsymbol{s} \in \mathbb{Z}_3^n$. We then construct another quantum circuit U satisfying

$$U(|\boldsymbol{\omega}\rangle|0\rangle) = \begin{cases} |\boldsymbol{\omega}\rangle|\,\|\boldsymbol{\omega}\|\rangle & \text{if } \boldsymbol{\omega} \neq \mathbf{0} \\ |\boldsymbol{\omega}\rangle|\,\|\boldsymbol{Be_1}\| + 1\rangle & \text{if } \boldsymbol{\omega} = \mathbf{0}, \end{cases}$$

and apply it on the second and third registers. Here $\boldsymbol{e_1} \in \mathbb{Z}^n$ is a vector whose first coordinate is one and rest are zero. After that, we apply the quantum minimum finding algorithm on the first and third registers and get an index $\boldsymbol{s'}$. The output of the algorithm will be $f_3^{1/3}(\boldsymbol{s'})$.

By Theorem 28, in $2^{0.661n+o(n)}$-time and $2^{0.5n+o(n)}$ space, we can generate $2^{0.161n+o(n)}$ vectors to construct a 1/3-BDD oracle. Thus O_{BDD} can be built using $2^{0.1608n+o(n)}$ Toffoli gates and poly(n) qubits. To see that we only need poly(n) qubits, we only keep the vectors of size smaller than $\exp(n)$ in the constuction of O_{BDD}, they thus can all be stored within poly(n) qubits. Since the vectors are sampled from a Gaussian with width at most $\exp(n)$, the error induced by throwing away the tail of the distribution is negligible. Furthermore all functions acting on these vectors can be implemented with poly(n) qubits.

We can also construct U efficiently. Hence, the algorithm needs $\mathcal{O}(2^{0.1608n+o(n)})$ Toffoli gates and poly(n) qubits for three registers. As a result by applying the quantum minimum finding algorithm from [19, Theorem 1], the quantum algorithm takes $3^{0.5n} \cdot 2^{0.1608n+o(n)} = 2^{0.9533n+o(n)}$ time and $2^{0.5n+o(n)}$ classical space with a polynomial number of qubits.

Lastly, we show that the quantum algorithm will output a shortest non-zero vector with constant probability. Since $\|\boldsymbol{Be_1}\| + 1 > \lambda_1(\mathcal{L})$, with at least 1/2 probability one will find the index $\boldsymbol{i}$ such that $f(\boldsymbol{i})$ is a shortest nonzero vector by using the quantum minimum finding algorithm from [19, Theorem 1]. Therefore it suffices to show that there is an index $\boldsymbol{i} \in \mathbb{Z}_3^n$ such that $\|f(\boldsymbol{i})\| = \lambda_1(\mathcal{L})$. By Theorem 16, the list $\{f(\boldsymbol{s})|\boldsymbol{s} \in \mathbb{Z}_3^n\}$ contains all lattice points within radius $3 \cdot \frac{1}{3}\lambda_1(\mathcal{L}) = \lambda_1(\mathcal{L})$ from $\mathbf{0}$, including the lattice vector with length $\lambda_1(\mathcal{L})$. Hence with at least 1/2 probability, the algorithm outputs a non-zero shortest lattice vector. ◀

4.2 Solving SVP by spherical caps on the sphere

We now explain how to reduce the number of queries to the α-BDD oracle. Consider a uniformly random target vector $\boldsymbol{t}$ such that $\alpha(1 - \frac{1}{n})\lambda_1(\mathcal{L}) \leq \|\boldsymbol{t}\| < \alpha\lambda_1(\mathcal{L})$, it satisfies the condition of Theorem 16, *i.e.* $\mathrm{dist}(\mathcal{L}, \boldsymbol{t}) \leq \alpha\lambda_1(\mathcal{L})$. We enumerate all lattice vectors within distance $2\alpha\lambda_1(\mathcal{L})$ to $\boldsymbol{t}$ and keep only the shortest nonzero one. We show that for $\alpha = 0.4097$, we will get the shortest nonzero vector of the lattice with probability at least $2^{-0.3298n+o(n)}$. By repeating this $\mathcal{O}(2^{0.3298n+o(n)})$ times, the algorithm will succeed with constant probability. We rely on the following construction of a 0.4097-BDD oracle.

▶ **Theorem 30.** *For any dimension $n \geq 4$, any integer $m > 0$, and a lattice $\mathcal{L} \subset \mathbb{R}^n$, there exists an algorithm that constructs a 0.4097-BDD oracle in $2^{0.9108n+o(n)}$ time and $2^{0.5n+o(n)}$ space. Each call to the oracle takes $2^{0.4108n+o(n)}$ time and space.*

Proof. See full version [1, Theorem 32]; it is similar to Theorem 24 but using Lemma 27. ◀

▶ **Theorem 31.** *There is a randomized algorithm that solves SVP in time* $2^{1.741n+o(n)}$ *and in space* $2^{0.5n+o(n)}$ *with constant probability.*

Proof. On input lattice $\mathcal{L}(\mathbf{B})$, use the LLL algorithm [41] to get a number d (the norm of the first vector of the basis) that satisfies $\lambda_1(\mathcal{L}) \leq d \leq 2^{n/2}\lambda_1(\mathcal{L})$. For $i = 1, \ldots, n^2$, let $d_i = d/(1+\frac{1}{n})^i$, and let $\alpha = 0.4097$. There exists a j such that $\lambda_1(\mathcal{L}) \leq d_j \leq (1+\frac{1}{n})\lambda_1(\mathcal{L})$. We repeat the following procedure for all $i = 1, \ldots, n^2$:

For $j = 1$ to $2^{0.3298n+o(n)}$, pick a uniformly random vector $\boldsymbol{v_{ij}}$ on the surface of the ball of radius $\alpha(1-\frac{1}{n})d_i$. By Theorem 16, we can enumerate 2^n lattice points using the function $f_{ij} : \mathbb{Z}_2^n \to \mathcal{L}$ defined by $f_{ij}(\boldsymbol{x}) = \mathbf{B}\,\boldsymbol{x} - 2 \cdot \mathsf{BDD}_\alpha(\mathcal{L}, (\mathbf{B}\,\boldsymbol{x} - \boldsymbol{v_{ij}})/2)$. At each step we only store the shortest nonzero vector. At the end, we output the shortest among them.

The running time of the algorithm is straightforward. We make 2^n queries to a α-BDD oracle that takes $2^{0.4108n+o(n)}$ time and space by Theorem 30. We further repeat this $n^2 2^{0.3298n+o(n)}$ times. Therefore the algorithm takes $2^{1.741n+o(n)}$ time and $2^{0.5n+o(n)}$ space.

To prove the correctness of the algorithm, it suffices to show that there exists an $i \in [n^2]$ for which the algorithm finds the shortest vector with high probability. Recall that there exists an i such that $\lambda_1(\mathcal{L}) \leq d_i \leq (1+\frac{1}{n})\lambda_1(\mathcal{L})$ and let that index be k. We will show that for a uniformly random vector $\boldsymbol{v}$ of length $\alpha(1-\frac{1}{n})d_k$, if we enumerate 2^n vectors by the function $f : \mathbb{Z}_2^n \to \mathcal{L}$, $f(\boldsymbol{x}) = \mathbf{B}\,\boldsymbol{x} - 2 \cdot \mathrm{BDD}_\alpha(\mathcal{L}, (\mathbf{B}\,\boldsymbol{x} - \boldsymbol{v})/2)$, then with probability $2^{-0.3298n-o(n)}$ there exists $\boldsymbol{x} \in \mathbb{Z}_2^n$ such that $f(\boldsymbol{x})$ is the shortest nonzero lattice vector.

We show that we can cover the sphere of radius λ_1 by $2^{0.3298n+o(n)}$ balls of radius $2\alpha\lambda_1 = 0.4097 * 2\lambda_1$ whose centers are at distance $\alpha(1-\frac{1}{n})d_k \leq 0.4097\lambda_1$ from the origin (see figure 1). We have two concentric circles of radius $\alpha(1-\frac{1}{n})d_k$ and λ_1, and let P be a uniformly random point on the surface of the ball of radius $\alpha(1-\frac{1}{n})d_k$. A ball of radius $2\alpha\lambda_1$ at center P will cover the spherical cap with angle ϕ of the ball of radius λ_1. By the law of the cosines, we can compute $\phi \approx \cos^{-1}(\frac{1-3\alpha^2}{2\alpha})$ and hence, by [4, Lemma 5.6], if we randomly choose $\boldsymbol{v}$, the corresponding spherical caps will cover the shortest vector with probability at least $\int_0^\phi \sin^{n-2}\theta d\theta \geq 2^{-0.3298n-o(n)}$. Besides, by Theorem 16, the list $\{f(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathbb{Z}_2^n\}$ will contain all lattice points within radius $2\alpha d_k$ from $\boldsymbol{v}$. Hence, the list will contain a shortest vector with probability $2^{-0.3298n+o(n)}$. By repeating this process $2^{0.3298n+o(n)}$ times, we can find the shortest vector with constant probability. ◀

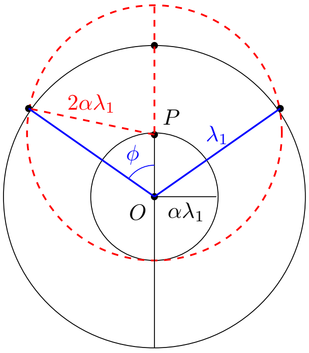


Figure 1 One can cover the sphere of radius λ_1 by balls of radius $2\alpha\lambda_1$, where $\frac{1}{3} \leqslant \alpha < \frac{1}{2}$, whose centers (here P) are at distance $\alpha\lambda_1$ from the origin O. Each such ball covers a spherical cap of half-angle ϕ.

References

1 Divesh Aggarwal, Yanlin Chen, Rajendra Kumar, and Yixin Shen. Improved (provable) algorithms for the shortest vector problem via bounded distance decoding (full version), 2020. arXiv:2002.07955.

2 Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2^n time using discrete gaussian sampling: Extended abstract. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 733–742, 2015. doi:10.1145/2746539.2746606.

3 Divesh Aggarwal, Jianwei Li, Phong Q. Nguyen, and Noah Stephens-Davidowitz. Slide reduction, revisited - filling the gaps in SVP approximation. *CoRR*, abs/1908.03724, 2019. arXiv:1908.03724.

4 Divesh Aggarwal and Noah Stephens-Davidowitz. (gap/s) eth hardness of svp. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 228–238, 2018.

5 Divesh Aggarwal and Noah Stephens-Davidowitz. Just take the average! an embarrassingly simple 2^n-time algorithm for SVP (and CVP). In *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018, New Orleans, LA, USA*, pages 12:1–12:19, 2018. doi:10.4230/OASIcs.SOSA.2018.12.

6 Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 99–108, 1996. doi:10.1145/237814.237838.

7 Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, STOC '01, pages 601–610, New York, NY, USA, 2001. ACM. doi:10.1145/380752.380857.

8 Martin R Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 717–746. Springer, 2019.

9 Yoshinori Aono, Phong Q. Nguyen, and Yixin Shen. Quantum lattice enumeration and tweaking discrete pruning. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018*, pages 405–434, Cham, 2018. Springer International Publishing.

10 Shi Bai, Thijs Laarhoven, and Damien Stehlé. Tuple lattice sieving. *IACR Cryptology ePrint Archive*, 2016:713, 2016. URL: http://eprint.iacr.org/2016/713.

11 Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 10–24, 2016. doi:10.1137/1.9781611974331.ch2.

12 Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 575–584. ACM, 2013.

13 Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 1–12, 2014. doi:10.1145/2554797.2554799.

14 Ernest F. Brickell. Breaking iterated knapsacks. In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, pages 342–358, 1984. doi:10.1007/3-540-39568-7_27.

15 Yanlin Chen, Kai-Min Chung, and Ching-Yi Lai. Space-efficient classical and quantum algorithms for the shortest vector problem. *Quantum Information & Computation*, 18(3&4):285–306, 2018. URL: http://www.rintonpress.com/xxqic18/qic-18-34/0285-0306.pdf.

16 Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. On the closest vector problem with a distance guarantee. In *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014*, pages 98–109, 2014. doi:10.1109/CCC.2014.18.

17 Rudi de Buda. Some optimal codes have structure. *IEEE Journal on Selected Areas in Communications*, 7(6):893–899, 1989. doi:10.1109/49.29612.

18 Léo Ducas. Shortest vector from lattice sieving: A few dimensions for free. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 125–145. Springer, 2018. doi:10.1007/978-3-319-78381-9_5.

19 Christoph Dürr and Peter Høyer. A quantum algorithm for finding the minimum. *CoRR*, quant-ph/9607014, 1996. arXiv:quant-ph/9607014.

20 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987. doi:10.1007/BF02579200.

21 Nicolas Gama and Phong Q. Nguyen. Finding short lattice vectors within mordell's inequality. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 207–216, 2008. doi:10.1145/1374376.1374408.

22 Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 257–278, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

23 Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178, 2009. doi:10.1145/1536414.1536440.

24 Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 197–206. ACM, 2008.

25 Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, pages 447–464, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

26 Guillaume Hanrot and Damien Stehlé. Improved analysis of kannan's shortest lattice vector algorithm. In *Advances in Cryptology – CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, pages 170–186, 2007. doi:10.1007/978-3-540-74143-5_10.

27 Ishay Haviv and Oded Regev. Tensor-based hardness of the shortest vector problem to within almost polynomial factors. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 469–477, 2007.

28 Bettina Helfrich. Algorithms to construct minkowski reduced and hermite reduced lattice bases. *Theor. Comput. Sci.*, 41(2–3):125–139, December 1985.

29 Gottfried Herold and Elena Kirshanova. Improved algorithms for the approximate k-list problem in euclidean norm. In Serge Fehr, editor, *Public-Key Cryptography – PKC 2017*, pages 16–40, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

30 Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

31 Russell Impagliazzo, Leonid A Levin, and Michael Luby. Pseudo-random generation from one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 12–24, 1989.

32 Hendrik W. Lenstra Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983. doi:10.1287/moor.8.4.538.

33 Grigorii Anatol'evich Kabatiansky and Vladimir Iosifovich Levenshtein. On bounds for packings on a sphere and in space. *Problemy Peredachi Informatsii*, 14(1):3–25, 1978.

34 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987. doi:10.1287/moor.12.3.415.

35 Subhash Khot. Hardness of approximating the shortest vector problem in lattices. *J. ACM*, 52(5):789–808, 2005. doi:10.1145/1089023.1089027.

36 Paul Kirchner and Pierre-Alain Fouque. Time-memory trade-off for lattice enumeration in a ball. Cryptology ePrint Archive, Report 2016/222, 2016. URL: https://eprint.iacr.org/2016/222.

37 Elena Kirshanova, Erik Mårtensson, Eamonn W Postlethwaite, and Subhayan Roy Moulik. Quantum algorithms for the approximate k-list problem and their application to lattice sieving. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 521–551. Springer, 2019.

38 Philip Klein. Finding the closest lattice vector when it's unusually close. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, page 937–941, USA, 2000. Society for Industrial and Applied Mathematics.

39 Thijs Laarhoven, Michele Mosca, and Joop Van De Pol. Finding shortest lattice vectors faster using quantum search. *Designs, Codes and Cryptography*, 77, December 2015. doi:10.1007/s10623-015-0067-5.

40 J. C. Lagarias and Andrew M. Odlyzko. Solving low-density subset sum problems. *J. ACM*, 32(1):229–246, 1985. doi:10.1145/2455.2461.

41 A.K. Lenstra, H.W. Lenstra, and Lászlo Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

42 Daniele Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, FOCS '98, page 92, USA, 1998. IEEE Computer Society.

43 Daniele Micciancio and Chris Peikert. Hardness of SIS and LWE with small parameters. In *Advances in Cryptology – CRYPTO 2013 – 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 21–39, 2013. doi:10.1007/978-3-642-40041-4_2.

44 Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 372–381, 2004. doi:10.1109/FOCS.2004.72.

45 Daniele Micciancio and Oded Regev. Lattice-based cryptography, 2008.

46 Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1468–1480, 2010. doi:10.1137/1.9781611973075.119.

47 Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations. *SIAM J. Comput.*, 42(3):1364–1391, 2013. doi:10.1137/100811970.

48 Daniele Micciancio and Michael Walter. Fast lattice point enumeration with minimal overhead. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 276–294, 2015. doi:10.1137/1.9781611973730.21.

49 Phong Q. Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *J. Mathematical Cryptology*, 2(2):181–207, 2008. doi:10.1515/JMC.2008.009.

50 Xavier Pujol and Damien Stehlé. Solving the shortest lattice vector problem in time $2^{2.465n}$. *IACR Cryptology ePrint Archive*, 2009:605, 2009. URL: http://eprint.iacr.org/2009/605.

51 Oded Regev. Lattices in computer science, lecture 8, Fall 2004.

52 Oded Regev. Lattice-based cryptography. In *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, pages 131–141, 2006. doi:10.1007/11818175_8.

53 Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, September 2009. doi:10.1145/1568318.1568324.

54 Claus Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53:201–224, 1987.

55 Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66:181–199, 1994. doi:10.1007/BF01581144.

56 Adi Shamir. A polynomial-time algorithm for breaking the basic merkle-hellman cryptosystem. *IEEE Trans. Information Theory*, 30(5):699–704, 1984. doi:10.1109/TIT.1984.1056964.

57 SVP Challenges. https://www.latticechallenge.org/svp-challenge/.

An FPT Algorithm for Elimination Distance to Bounded Degree Graphs

Akanksha Agrawal
Indian Institute of Technology Madras, Chennai, India

Lawqueen Kanesh
The Institute of Mathematical Sciences, HBNI, Chennai, India

Fahad Panolan
Indian Institute of Technology, Hyderabad, India

M. S. Ramanujan
University of Warwick, Coventry, UK

Saket Saurabh
University of Bergen, Norway
The Institute of Mathematical Sciences, HBNI, Chennai, India

Abstract

In the literature on parameterized graph problems, there has been an increased effort in recent years aimed at exploring novel notions of graph edit-distance that are more powerful than the size of a modulator to a specific graph class. In this line of research, Bulian and Dawar [Algorithmica, 2016] introduced the notion of elimination distance and showed that deciding whether a given graph has elimination distance at most k to any minor-closed class of graphs is fixed-parameter tractable parameterized by k [Algorithmica, 2017]. They showed that Graph Isomorphism parameterized by the elimination distance to bounded degree graphs is fixed-parameter tractable and asked whether determining the elimination distance to the class of bounded degree graphs is fixed-parameter tractable. Recently, Lindermayr et al. [MFCS 2020] obtained a fixed-parameter algorithm for this problem in the special case where the input is restricted to K_5-minor free graphs.

In this paper, we answer the question of Bulian and Dawar in the affirmative for general graphs. In fact, we give a more general result capturing elimination distance to any graph class characterized by a finite set of graphs as forbidden induced subgraphs.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Elimination Distance, Fixed-parameter Tractability, Graph Modification

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.5

Funding *Akanksha Agrawal*: Supported by New Faculty Initiative Grant, IIT Madras, Chennai, India.
Fahad Panolan: Supported by Seed grant, IIT Hyderabad (SG/IITH/F224/2020-21/SG-79).
M. S. Ramanujan: Supported by EPSRC grant EP/V007793/1.
Saket Saurabh: Supported by funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416) and Swarnajayanti Fellowship grant DST/SJF/MSA-01/2017-18.

1 Introduction

A popular methodology for studying the parameterized complexity of problems is to consider *parameterization by distance from triviality* [16]. In this methodology, the idea is to try and lift the tractability of special cases of generally hard computational problems, to tractability of instances that are "close" to these special cases (i.e., close to triviality) for appropriate

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 5; pp. 5:1–5:11
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

notions of "distance from triviality". This way of parameterizing graph problems has led to a rich collection of sophisticated algorithmic and lower bound machinery over the last two decades.

One direction in which this approach has been extended in recent years is by enhancing existing notions of distance from triviality by exploiting some form of *structure* underlying vertex modulators rather than just the size bound. This line of exploration has led to the development of several new notions of distance from triviality [12, 5, 6, 15, 14, 11]. Of primary interest to us in this line of research is the notion of *elimination distance* introduced in [5]. Bulian and Dawar [5] introduced the notion of elimination distance in an effort to define tractable parameterizations that are more general than the modulator size for graph problems. We refer the reader to Section 2 for a formal definition of this parameter. In their work, they focused on the Graph Isomorphism (GI) problem and showed that GI is fixed-parameter tractable (FPT) when parameterized by the elimination distance to graphs of bounded degree. In follow-up work, Bulian and Dawar [6] showed that deciding whether a given graph has elimination distance at most k to any minor-closed class of graphs is fixed-parameter tractable parameterized by k (i.e., can be solved in time $f(k)n^{\mathcal{O}(1)}$) and asked whether computing the elimination distance to graphs of bounded degree is fixed-parameter tractable.

Recently, Lindermayr et al. [18] showed that computing elimination distance to bounded degree graphs is fixed-parameter tractable *when the input is planar*. However their approach is specifically adapted to planar graphs (in fact, more generally, K_5-minor free graphs) and they note that their approach does not appear to extend to general graphs. In this paper, we address the general question and show that the problem is (non-uniformly) fixed-parameter tractable on general graphs. In fact, we prove a more general result and obtain the result for elimination distance to graphs of bounded degree as a consequence. Let $\mathcal{F}$ be a finite family of graphs. We say that a graph G is $\mathcal{F}$-free if G does not contain any induced subgraph isomorphic to a graph in $\mathcal{F}$.

▶ **Theorem 1.** *For every fixed finite family $\mathcal{F}$ of finite graphs and $k \in \mathbb{N}$, there is an algorithm $\mathcal{A}_k^{\mathcal{F}}$ that, given a graph G, runs in time $f(k) \cdot n^{\mathcal{O}(1)}$ for some function f and correctly decides whether G has elimination distance at most k to the class of $\mathcal{F}$-free graphs.*

We remark that the exponent of n in the above running time is a constant depending on $\mathcal{F}$ and independent of k. As a corollary (and with a slight modification of the above algorithms), we obtain the following result for determining elimination distance to bounded degree graphs.

▶ **Corollary 2.** *For every $k \in \mathbb{N}$, there is an algorithm $\mathcal{A}_k$ that, given a graph G and integer $d \in \mathbb{N}$, runs in time $f(k,d) \cdot n^{\mathcal{O}(1)}$ for some function f and correctly decides whether G has elimination distance at most k to the class of graphs of degree at most d.*

In the above statement, the exponent of n is a constant independent of both d and k.

Related work

Hols et al. [17] recently presented a comprehensive study of the classic Vertex Cover problem parameterized by the size of a smallest modulator to graphs that have bounded elimination distance to specific hereditary graph classes. They provided an elegant (partial) characterization of parameterizations that permit polynomial kernelizations for Vertex Cover. Bougeret et al. [3] introduced a measure called bridge-depth and showed that a minor-closed family of graphs $\mathcal{F}$ has bounded bridge-depth precisely when Vertex Cover admits a polynomial kernel parameterized by the size of a modulator to $\mathcal{F}$ (subject to standard complexity theoretic hypotheses). The notion of elimination distance [5] generalizes the notion of *generalized treedepth* introduced by Bouland et al. [4] in an effort to combine the treedepth and max-leaf number parameters. Building on [5] and extending the approach of combining width parameters (treedepth in the case of [5]) and modulator size, Ganian et al. [15] proposed a measure of distance to triviality for CSP that depended on the treewidth of an appropriate graph defined on backdoor sets (these can be thought of as a version of vertex modulators appropriate for use in solving ILP and CSP instances). That is, they introduced a way of combining *treewidth* and modulator size into a single parameter that is stronger than elimination distance. More recently, Eiben et al. [11] continued the line of research into combining modulators and width parameters by studying this parameter in the context of graph problems, where triviality is expressed in terms of bounded rankwidth.

2 Preliminaries

For an undirected graph G, we use n and m to denote $|V(G)|$ and $|E(G)|$ respectively, unless mentioned otherwise. For $X \subseteq V(G)$, $G[X]$ denotes the graph with vertex set X and the edge set $\{\{x,y\} \in E(G) \mid x,y \in X\}$. By $G - X$ we denote the graph $G[V(G) \setminus X]$. Let $v \in V(G)$. Then, by $N_G(v)$ we denote the set of neighbors of v in G, i.e., the set $\{u \in V(G) \mid \{u,v\} \in E(G)\}$. By $N_G[v]$, we denote the closed neighborhood of v in G, i.e., $N_G(v) \cup \{v\}$. For a set $U \subseteq V(G)$, by $N_G(U)$ we denote the set $\cup_{u \in U} N_G(u) \setminus U$, by $N_G[U]$ we denote the set $N_G(U) \cup U$. By $deg_G(v)$, we denote the degree of vertex v in G, i.e., the number of edges incident on v in G. We drop the subscript whenever the context is clear.

A *path* $P = (v_1, v_2, \cdots, v_\ell)$ in G is a subgraph of G where $V(P) = \{v_1, v_2, \cdots, v_\ell\} \subseteq V(G)$ is a set of distinct vertices and $E(P) = \{\{v_i, v_{i+1}\} \mid i \in [\ell - 1]\} \subseteq E(G)$, where $|V(P)| = \ell$ for some $\ell \in [|V(G)|]$. The above defined path P is called as $v_1 - v_\ell$ path. We say that the graph G is *connected* if for every $u, v \in V(G)$, there exists a $u - v$ path in G. A *connected component* of G is an inclusion-wise maximal connected induced subgraph of G. The set $\mathcal{C}(G)$ denotes the set of connected components of G. For a tree T and vertices $u, v \in V(T)$, we denote the unique path between u and v by $\mathsf{Pth}_T(u,v)$. A tree is called as a *rooted tree* if special vertex in tree is designated to be the *root*. Let T be a rooted tree with root $r \in V(T)$. We say that a vertex $v \in V(T) \setminus \{r\}$ is a *leaf* of T if the $deg_T(v) = 1$. Moreover, if $V(T) = \{r\}$, then r is the leaf (as well as the root) of T. A vertex which is not a leaf, is a *non-leaf* vertex. Let $t, t' \in V(T)$ such that $\{t, t'\} \in E(T)$ and t' is not contained in $t - r$ path in T, then we say that t is the *parent* of t' and t' is a *child* of t.

A vertex $t' \in V(T)$ (t' can possibly be the same as t) is a *descendant* of t, if in $T - \{\mathsf{par}_T(t)\}$, where $\mathsf{par}_T(t)$ is the parent of t, there is a $t - t'$ path. Note that when $t = r$, then $T - \{\mathsf{par}_T(t)\} = T$, as the parent of r does not exist. (Every vertex in T is a descendant of r.) By $\mathsf{desc}_T(t)$, we denote the set of all descendants of t in T. We drop the subscript T from $\mathsf{par}_T(\cdot)$ and $\mathsf{desc}_T(\cdot)$, when the context is clear.

A *rooted forest* is a forest where each connected component is a rooted tree. For a rooted forest F, a vertex $v \in V(F)$ that is not a root of any of its rooted trees is a *leaf* if it is of degree exactly one in F. We denote the set of leaves in a rooted forest by $\mathsf{Lf}(F)$.

The *depth*, denoted by $\mathsf{depth}(T)$ of a rooted tree T is the maximum number vertices in a root to leaf path in T. The *depth*, denoted by $\mathsf{depth}(F)$ of a rooted forest F is the maximum over the depths of its rooted trees.

▶ **Definition 3** (Forest embedding). A *forest embedding* of a graph G is a pair (F, f), where F is a rooted forest and $f : V(G) \to V(F)$ is a bijective function, such that for each $\{u, v\} \in E(G)$, either $f(u)$ is a descendant of $f(v)$, or $f(v)$ is a descendant of $f(u)$. The *depth* of the forest embedding (F, f) is the depth of the rooted forest F.[1]

Next, we recall the notion of *elimination-distance* introduced by Bulian and Dawar [5]. We rephrase their definition and introduce notation that will facilitate our presentation.

▶ **Definition 4** (Elimination Distance and $(\eta, \mathcal{H})$-decompositions). Consider a family $\mathcal{H}$ of graphs and an integer $\eta \in \mathbb{N}$. An $(\eta, \mathcal{H})$-decomposition of a graph G is a tuple $(X, Y, F, f : X \to V(F), g : \mathcal{C}(G[Y]) \to \mathsf{Lf}(F) \cup \{\perp\})$, where (X, Y) is a partition of $V(G)$ and F is a rooted forest of depth η, such that the following conditions are satisfied:

1. (F, f) is a forest embedding of $G[X]$,
2. each connected component of $G[Y]$ belongs to $\mathcal{H}$, and
3. for a connected component C of $G[Y]$, a vertex $v \in V(C)$, and an edge $\{u, v\} \in E(G)$, either $u \in Y$ or $f(u)$ is a vertex in the unique path in F from r to $g(C)$, where r is the root of the connected component in F containing the vertex $g(C)$.[2]

We say that G admits an $(\eta', \mathcal{H})$-decomposition if there is some $\eta \leq \eta'$, for which there is an $(\eta, \mathcal{H})$-decomposition of G. The *elimination distance* of G to $\mathcal{H}$ (or the *$\mathcal{H}$-elimination distance* of G) is the smallest integer η^* for which G admits an $(\eta^*, \mathcal{H})$-decomposition.

We remark that when G is a connected graph, no component in $\mathcal{C}(G[Y])$ is mapped to $\perp$ (i.e., g is indeed a function from $\mathcal{C}(G[Y])$ to $\mathsf{Lf}(F)$).

Consider an $(\eta, \mathcal{H})$-decomposition $\mathbb{D} = (X, Y, F, f, g)$ of a graph G. We say that $\mathbb{D}$ is an $(\eta, \mathcal{H})$-decomposition of G on forest F. We say that X is the *interior part* of $\mathbb{D}$ and Y is the *exterior part* of $\mathbb{D}$. For a leaf $u \in \mathsf{Lf}(F)$, by $\widehat{P}_u^{\mathbb{D}}$ we denote the path from u to r in the tree T, where T is the tree rooted at r in F, containing u. Moreover, by $P_u^{\mathbb{D}}$, we denote the graph $G[\{f^{-1}(w) \mid w \in V(\widehat{P}_u^{\mathbb{D}})\}]$. For a connected component $C \in \mathcal{C}(G[Y])$, by $C_{\mathsf{ext}}^{\mathbb{D}}$ we denote the graph $G[V(C) \cup \{f^{-1}(w) \mid w \in V(\mathsf{Pth}_F(g(C), r))\}]$, where r is the root of the component of F containing $g(C)$. (For the above notations we drop the superscript $\mathbb{D}$, when the context is clear.)

For a graph G, the elimination distance of G to $\mathcal{H} = \{(\{u\}, \emptyset)\}$ is the treedepth of G, denoted by $\mathsf{td}(G)$. We use the following simple observation in a later section.

▶ **Observation 5.** *Let q be a positive integer and $\mathcal{H}_q$ be a family of graphs where each graph has at most q vertices. If for a graph G, the elimination distance of G to $\mathcal{H}_q$ is η, then the treedepth of G is at most $\eta + q$.*

Consider a graph G, a non-empty set $Q \subseteq V(G)$, and integers $p, q \in \mathbb{N}$. We say that a set $A \subseteq V(G)$ such that $G[A]$ is connected, is a (Q, p, q)-connected set if $Q \subseteq A$, $|A| = p$, and $|N(A)| \leq q$. The next proposition follows from Lemma 3.1 of [13].

▶ **Proposition 6.** *Consider a graph G, a non-empty set $Q \subseteq V(G)$, and integers $p, q \in \mathbb{N}$. The number of (Q, p, q)-connected sets in G is at most 2^{p+q}. Moreover, there exists an algorithm which runs in $2^{p+q} \cdot n^{\mathcal{O}(1)}$ time and enumerates all (Q, p, q)-connected sets in G'.*

[1] Sometimes we slightly abuse the notation for simplicity, and say that, for every $\beta \geq \alpha$, (F, f) is a forest embedding of depth β, where α is the depth of F.

[2] If $g(C) = \perp$, then u must belong to Y.

Unbreakability. To formally introduce the notion of unbreakability, we rely on the definition of a separation:

▶ **Definition 7** (Separation). *A pair (X, Y) where $X \cup Y = V(G)$ is a* separation *if $E(X \setminus Y, Y \setminus X) = \emptyset$. The order of (X, Y) is $|X \cap Y|$.*

Roughly speaking, a graph is breakable if it is possible to "break" it into two large parts by removing only a small number of vertices. Formally,

▶ **Definition 8** ((s, c)-Unbreakable graph). *Let G be a graph. If there exists a separation (X, Y) of order at most c such that $|X \setminus Y| \geq s$ and $|Y \setminus X| \geq s$, called an* (s, c)-witnessing separation*, then G is* (s, c)-breakable*. Otherwise, G is* (s, c)-unbreakable*.*

Counting Monadic Second Order Logic. The syntax of Monadic Second Order Logic (MSO) of graphs includes the logical connectives $\vee$, $\wedge$, $\neg$, $\leftrightarrow$, $\Rightarrow$, variables for vertices, edges, sets of vertices and sets of edges, the quantifiers $\forall$ and $\exists$, which can be applied to these variables, and five binary relations:

1. $u \in U$, where u is a vertex variable and U is a vertex set variable;
2. $d \in D$, where d is an edge variable and D is an edge set variable;
3. $\mathbf{inc}(d, u)$, where d is an edge variable, u is a vertex variable, and the interpretation is that the edge d is incident to u;
4. $\mathbf{adj}(u, v)$, where u and v are vertex variables, and the interpretation is that u and v are adjacent;
5. equality of variables representing vertices, edges, vertex sets and edge sets.

Counting Monadic Second Order Logic (CMSO) extends MSO by including atomic sentences testing whether the cardinality of a set is equal to q modulo r, where q and r are integers such that $0 \leq q < r$ and $r \geq 2$. That is, CMSO is MSO with the following atomic sentence: $\mathbf{card}_{q,r}(S) = \mathbf{true}$ if and only if $|S| \equiv q \pmod{r}$, where S is a set. We refer to [1, 8, 9] for a detailed introduction to CMSO.

We will crucially use the following result of Lokshtanov et al. [19] that allows one to obtain a (non-uniform) FPT algorithm for CMSO-expressible graph problems by designing an FPT algorithm for the problem on unbreakable graphs.

▶ **Proposition 9** (Theorem 1, [19]). *Let ψ be a CMSO sentence and let $d > 4$ be a positive integer. For all $c \in \mathbb{N}$, there exists $s \in \mathbb{N}$ such that if there exists an algorithm that solves* CMSO$[\psi]$ *on (s, c)-unbreakable graphs in time $\mathcal{O}(n^d)$, then there exists an algorithm that solves* CMSO$[\psi]$ *on general graphs in time $\mathcal{O}(n^d)$.*

3 The algorithm for k-Elimination Distance to $\mathcal{H}_\mathcal{F}$

In the rest of the paper, we fix the family $\mathcal{F}$ and let d denote the maximum taken over the number of vertices in the graphs in $\mathcal{F}$. Recall that $\mathcal{H}_\mathcal{F}$ denotes the family of all $\mathcal{F}$-free graphs. In the k-Elimination Distance to $\mathcal{H}_\mathcal{F}$ problem, the input is a graph G and the goal is to determine whether G has elimination distance at most k to $\mathcal{H}_\mathcal{F}$. Notice that k is assumed to be fixed since it is part of the problem definition. For a graph G, we say that $X \subseteq V(G)$ is a *k-elimination distance modulator* of G to $\mathcal{F}$-free graphs (k-ed modulator to $\mathcal{H}_\mathcal{F}$) if G admits a $(k, \mathcal{H}_\mathcal{F})$-decomposition.

Lindermayr et al. [18] obtain their algorithm by repeatedly either identifying an irrelevant vertex or by identifying substructures that they call "connectivity patterns" using which they formulate an appropriate CMSO formula over a bounded treedepth structure. In their algorithm, they crucially require that the input graph is K_5-minor free. Moreover, they note that in their approach, all difficulties for elimination distance to bounded degree arise already in bounded degree graphs and conjectured that solving this problem on bounded degree graphs is sufficient to solve the problem on general graphs.

In our case, instead of focussing on bounded degree inputs as the hard case, we focus on input graphs that are well-connected everywhere (i.e., (s, c)-unbreakable for appropriate values of s and c). Towards this, we rely on Proposition 9, which requires the CMSO-expressibility of the k-Elimination Distance to $\mathcal{H}_\mathcal{F}$ problem.

▶ **Lemma 10.** *k-Elimination Distance to $\mathcal{H}_\mathcal{F}$ is CMSO-expressible.*

Proof. Notice that $X \subseteq V(G)$ is a k-ed modulator to $\mathcal{H}_\mathcal{F}$ if and only if $G - X$ is $\mathcal{F}$-free and the graph $\mathsf{Torso}(X)$ has treedepth at most k. Here, $\mathsf{Torso}(X)$ is defined as the graph obtained from $G[X]$ by making the neighborhood of every connected component of $G - X$ a clique. In other words, for every distinct $u, v \in X$, (u, v) is an edge in $\mathsf{Torso}(X)$ if and only if either $(u, v) \in E(G)$ or there is a path in G whose endpoints are u and v and whose remaining vertices are disjoint from X. We use this characterization of k-ed modulators to $\mathcal{H}_\mathcal{F}$ to write our CMSO formula.

It is straightforward to assert that a graph is an induced subgraph of G in CMSO. Therefore, one can write a sentence asserting that X is a vertex subset in G such that $G - X$ has no induced subgraph isomorphic to a graph in $\mathcal{F}$. Now, consider the class of graphs of treedepth at most k and notice that these are minor closed and hence characterized by a set of $\beta(k)$ forbidden minors for some function β. Recall that k is fixed in our setting and so, we may assume that these forbidden minors are "hardcoded" into our algorithm. Moreover, it is well-known that one can assert that a graph is a minor of a given graph, in CMSO. Hence, it only remains for us to describe how we express in CMSO, the graph $\mathsf{Torso(X)}$ for some $X \subseteq V(G)$. Here, we encode the edge-set of $\mathsf{Torso}(X)$ into a new binary relation $\mathbf{adj}^\star$ such that for $u, v \in X$ $\mathbf{adj}^\star(u, v)$ if and only if $\mathbf{adj}(u, v)$ or there exists a vertex set Y such that $Y \cap X = \{u, v\}$ and $\mathbf{conn}(Y)$ is true, where $\mathbf{conn}(Y)$ is the standard CMSO sentence that asserts that the graph $G[Y]$ is connected (see, e.g., [10]). This completes the proof of the lemma. ◀

We remark that it is possible to sidestep the requirement that the forbidden minors for treedepth k are hardcoded into our algorithm. This can be done by writing a sentence that is based on the recursive-elimination definition of treedepth (see [6] for this definition) as opposed to a forbidden minor characterization. See [18] for an example of such a formulation. In either case, the size of the final CMSO formula for k-Elimination Distance to $\mathcal{H}_\mathcal{F}$ will depend on k and $\mathcal{F}$.

3.1 k-Elimination Distance to $\mathcal{H}_\mathcal{F}$ on (α_k, k)-Unbreakable Graphs

We invoke Proposition 9 with $c = k$ and let α_k denote the value of s given by this invocation. Due to this proposition, it is sufficient for us to give an FPT algorithm for k-Elimination Distance to $\mathcal{H}_\mathcal{F}$ on (α_k, k)-unbreakable graphs and that will be our goal in the rest of this section.

We begin by recalling a straightforward enumeration algorithm to enumerate all minimal $\mathcal{H}_\mathcal{F}$-modulators of bounded size.

▶ **Observation 11.** *There is an algorithm that runs in time $d^{\alpha_k + k} n^{\mathcal{O}(d)}$ and either correctly concludes that there is no $\mathcal{H}_\mathcal{F}$-modulator of G of size at most $\alpha_k + k$ or outputs a family $\mathcal{Z} = \{Z_1, \ldots, Z_\ell\}$ of at most $d^{\alpha_k + k}$ vertex sets comprising every minimal $\mathcal{H}_\mathcal{F}$-modulator of G of size at most $\alpha_k + k$.*

The above algorithm simply locates an induced subgraph of the given graph which is isomorphic to a graph in $\mathcal{F}$ by brute force and then branches on the vertices in this subgraph. We note that in the special case of $\mathcal{F}$ where $\mathcal{F}$-free graphs are precisely the set of graphs of degree at most some r, the exponent of n in the above algorithm can be made independent of r since computing a forbidden structure, i.e., a vertex of degree at least $r+1$, is polynomial-time solvable. Applying this insight allows us to infer Corollary 2 by slightly modifying the application of Observation 11 in the proof of Theorem 1.

In what follows (Lemma 12–Lemma 14), consider an (α_k, k)-unbreakable graph G, and an integer k such that (G, k) is a yes-instance of (α_k, k)-Unbreakable Elimination Distance to $\mathcal{H}_{\mathcal{F}}$. For ease of presentation, we assume that G is a connected graph.

▶ **Lemma 12.** *Let $\mathbb{D} = (X, Y, F, f : V(X) \to V(F), g : \mathcal{C}(G[Y]) \to \mathsf{Lf}(F))$ be a $(k, \mathcal{H}_{\mathcal{F}})$-decomposition of G. Then, the following properties hold:*

1. *$G[Y]$ contains at most one connected component of size at least α_k.*
2. *If $G[Y]$ does not contain a connected component of size at least α_k, then $\mathsf{td}(G) \leq \alpha_k + k$.*
3. *If $\mathsf{td}(G) > \alpha_k + k$, then, $G[Y]$ contains exactly one connected component of size at least α_k and $|X| \leq \alpha_k + k$.*

Proof. Consider the first statement. Suppose to the contrary that there are two connected components $C_1, C_2 \in \mathcal{C}(G[Y])$, such that $|V(C_1)| \geq \alpha_k$ and $|V(C_2)| \geq \alpha_k$. Recall that the set $N(C_1)$ is contained in a single root-to-leaf path in F, and hence has size at most k. Formally, let $g(C_1) = u$ and notice that by the definition of $(k, \mathcal{H}_{\mathcal{F}})$-decomposition (Item 3 of Definition 4), $N(C_1) \subseteq V(P_u^{\mathbb{D}})$. Also, since the depth of F is at most k, $|V(P_u^{\mathbb{D}})| \leq k$, and hence $|N(C_1)| \leq k$. Then, the separation $(N[C_1], V(G) \setminus C_1)$ is an (α_k, k)-witnessing separation, a contradiction to G being (α_k, k)-unbreakable.

For the second statement, if $G[Y]$ does not contain a connected component of size at least α_k, then indeed $\mathbb{D}$ is a $(k, \mathcal{H}_{\alpha_k})$-decomposition of G, where $\mathcal{H}_{\alpha_k}$ is the family of graphs with at most α_k vertices each. That is, the elimination distance of G to $\mathcal{H}_{\alpha_k}$ is at most k. Thus, by Observation 5, $\mathsf{td}(G) \leq \alpha_k + k$.

We now prove the third statement. From the previous two statements, we have that $G[Y]$ contains exactly one connected component of size at least α_k. Let $C^\star$ be this unique connected component of size at least α_k. Now, suppose that $|X| > \alpha_k + k$. Let $g(C^\star) = x$. Then, we have that $S = N_G(C^\star) \subseteq V(P_x^{\mathbb{D}})$. Observe that $|V(P_x^{\mathbb{D}})| \leq k$ and hence $|S| \leq k$. Let $V_1 = N[V(C^\star)], V_2 = V(G) \setminus V(C^\star)$. Note that, $(X \setminus S) \subseteq V_2$, therefore $|V_2| \geq \alpha_k$. We have that, (V_1, V_2)-separation of order at most k in G and $|V_1|, |V_2| \geq \alpha_k$. This contradicts the assumption that G is (α_k, k)-unbreakable graph. ◀

The above lemma allows us to assume that either the treedepth (and hence also the treewidth) of the input graph is bounded by $\alpha_k + k$ (in which case one can use Lemma 10 and Courcelle's theorem [8] to solve the problem), or conclude that if (G, k) is a yes-instance, then the set X is a $\mathcal{H}_{\mathcal{F}}$-modulator of size at most $\alpha_k + k$. But, notice that X may not be a minimal $\mathcal{H}_{\mathcal{F}}$-modulator. In fact, this is where the difficulty lies. In the following two lemmas, we assume that G has treedepth at least $\alpha_k + k + 1$ and present crucial structural properties on which our main algorithm is based.

▶ **Lemma 13.** *Let $\mathbb{D} = (X, Y, F, f : V(X) \to V(F), g : \mathcal{C}(G[Y]) \to \mathsf{Lf}(F))$ be a $(k, \mathcal{H}_{\mathcal{F}})$-decomposition of G. Moreover, suppose that $\mathsf{td}(G) > \alpha_k + k$ and let $Z \subseteq X$ such that $G - Z$ is $\mathcal{F}$-free. Let $C^\star$ be the unique connected component of $G - X$ of size at least α_k. Then, the following hold:*

1. *There is a unique connected component of $G - Z$ that contains $V(C^\star)$.*
2. *$V(G) \setminus V(C^\star)$ has size at most $\alpha_k + k$.*

Proof. As $C^\star$ is a connected component in $G - X$ and $Z \subseteq X$, there is a unique connected component in $G - Z$ that contains $V(C^\star)$. This proves the first statement. Now we prove the second statement of the lemma. Let $g(C^\star) = x$. Then, we have that $S = N_G(C^\star) \subseteq V(P_x^{\mathbb{D}})$. Observe that $|S| \leq |V(P_x^{\mathbb{D}})| \leq k$. Let $V_1 = N[V(C^\star)]$ and $V_2 = V(G) \setminus V(C^\star)$. For the sake of contradiction, suppose that $|V(G) \setminus V(C^\star)| > \alpha_k + k$. Then, (V_1, V_2) is a separation in G of order at most k and $|V_1 \setminus V_2|, |V_2 \setminus V_1| \geq \alpha_k$. This contradicts the assumption that G is a (α_k, k)-unbreakable graph. ◀

▶ **Lemma 14.** *Let $\mathbb{D} = (X, Y, F, f : V(X) \to V(F), g : \mathcal{C}(G[Y]) \to \mathsf{Lf}(F))$ be a $(k, \mathcal{H}_{\mathcal{F}})$-decomposition of G and let $Z \subseteq X$ be a $\mathcal{H}_{\mathcal{F}}$-modulator of G. Let $C^\star$ be the unique connected component of $G - X$ of size at least α_k. Let $v^\star$ be an arbitrary vertex in $C^\star$ and let C be the component of $G - Z$ that contains $v^\star$. Let $J \subseteq N(C)$ be an arbitrary set of size $\min\{k+1, |N(C)|\}$. Let $J_C \subseteq V(C)$ be an arbitrary set of size at most $|J|$ such that $N(J_C) \supseteq J$. Then, one of the following statements hold:*

1. *There is a leaf $u \in \mathsf{Lf}(F)$ such that $J \subseteq V(\widehat{P}_u^{\mathbb{D}})$. That is, the vertices in J lie on a single root-to-leaf path in F.*
2. *$J_C \cap X \neq \emptyset$.*
3. *$J_C \not\subseteq V(C^\star)$.*

Proof. Suppose to the contrary that none of these statements hold. That is, there are vertices $v_1, v_2 \in J$ such that v_p is not a descendant of v_q in F for distinct $p, q \in \{1, 2\}$, $J_C \cap X = \emptyset$ and $J_C \subseteq V(C^\star)$. Since $N(J_C) \supseteq J$ and $J_C \subseteq V(C^\star)$, it follows that $J \subseteq N(C^\star)$. Item 3 of Definition 4 guarantees that J is contained in $\widehat{P}_{g(C^\star)}^{\mathbb{D}}$, implying that out of v_1 and v_2, one is the descendant of the other in F. This gives us a contradiction. ◀

We are now ready to present our algorithm for k-Elimination Distance to $\mathcal{H}_{\mathcal{F}}$ on (α_k, k)-unbreakable graphs.

▶ **Lemma 15.** *k-Elimination Distance to $\mathcal{H}_{\mathcal{F}}$ on (α_k, k)-unbreakable graphs can be solved in time $2^{\mathcal{O}((\alpha_k+k)^2)} n^{\mathcal{O}(1)}$.*

Proof. Let (G, k) be the input. We first check if the treewidth of G is bounded by $\alpha_k + k$ using the algorithm of Bodlaender [2]. If yes, then we solve the problem using Courcelle's theorem. Suppose that this is not the case. Suppose that the input (G, k) is a yes-instance of k-Elimination Distance to $\mathcal{H}_{\mathcal{F}}$ and let $\mathbb{D} = (X, Y, F, f : V(X) \to V(F), g : \mathcal{C}(G[Y]) \to \mathsf{Lf}(F))$ be a hypothetical $(k, \mathcal{H}_{\mathcal{F}})$-decomposition of G. Using Lemma 12, we may assume that there is a unique component $C^\star$ in $G[Y]$ that has size at least α_k. Moreover, as the treewidth of G is at least $\alpha_k + k$, the third statement of this lemma guarantees that X has size at most $\alpha_k + k$. Our algorithm begins by guessing $v^\star$ and then uses Observation 11 to guess a set $Z \subseteq X$ that is a minimal $\mathcal{H}_{\mathcal{F}}$-modulator of size at most $\alpha_k + k$. There are n choices for $v^\star$ and $d^{\alpha_k + k}$ choices for Z. If the algorithm of Observation 11 concludes that a $\mathcal{H}_{\mathcal{F}}$-modulator of size at most $\alpha_k + k$ does not exist, then the second statement of Lemma 13 is used to correctly conclude that the input is a no-instance.

It then calls upon a subroutine Alg-special that takes as input G, k and a set $\widehat{Z}$ of size at most $\alpha_k + k$ and either correctly concludes that there is a set $X \supseteq \widehat{Z}$ of size at most $\alpha_k + k$ that is a k-elimination distance modulator to $\mathcal{H}_{\mathcal{F}}$, or correctly concludes that one does not exist. Our main algorithm invokes Alg-special with input G, k and $\widehat{Z} := Z$. In the rest of the proof, we describe and analyze Alg-special.

Alg-special first checks whether $|\widehat{Z}| \leq \alpha_k + k$. If not, then it returns NO. Otherwise, it locates the component C of $G - \widehat{Z}$ that contains $v^\star$ (and hence also contains $V(C^\star)$) and computes J and J_C as described in Lemma 14. It then considers the following three exhaustive possibilites and branches over all of them:

(i) the vertices in J lie on a root-to-leaf path in F or
(ii) J_C intersects X or
(iii) at least one vertex of J_C is not contained in $C^\star$.

In Case (i), notice that $|J|$ must be at most k since the depth of F is at most k. Hence, by the definition of J, it must be the case that $J = N(C)$. We now observe that without loss of generality, C is disjoint from X. This is because C is $\mathcal{F}$-free, $N(C) \subseteq X$ and moreover, $N(C)$ lies on a root-to-leaf path in F. Thus, to solve the problem in Case (i), it is sufficient to check for the existence of a set X such that $X \subseteq V(G) \setminus V(C)$, $X \supseteq \widehat{Z}$, X has size at most $\alpha_k + k$ and X is a k-elimination distance modulator to $\mathcal{H}_\mathcal{F}$. However, since $X \subseteq V(G) \setminus V(C)$, it follows that X is contained in $V(G) \setminus V(C^\star)$, which has size at most $\alpha_k + k$ (second statement of Lemma 13). Therefore, the existence of X can be verified by going over all possible subsets of $X \subseteq V(G) \setminus V(C)$. This completes the description of the algorithm in Case (i).

To handle Case (ii), we recursively call Alg-special on input $(G, k, \widehat{Z} \cup \{v\})$ for each $v \in J_C$ and return YES if at least one of the recursive calls return YES.

Suppose that Cases (i) and (ii) do not hold. We now handle Case (iii) as follows. We know that for at least one vertex of J_C, say, $y^\star$, the connected component of $G[Y]$ containing $y^\star$ is not the same as $C^\star$. Let $C_{y^\star}$ be this component. From the third statement of Lemma 12, we have that exactly one component of $G[Y]$ has size at least α_k. Hence, $C_{y^\star}$ has size at most α_k. Moreover, $N(C_{y^\star})$ must have size at most k since it is contained in $V(P^{\mathbb{D}}_{g(C_{y^\star})})$, which has size at most k. Hence, we guess $y^\star \in J_C$, enumerate all $(y^\star, \alpha_k, k)$-connected sets B and recursively call Alg-special on $(G, k, \widehat{Z} \cup N(B))$. Notice that since $y^\star$ and $v^\star$ lie in the same component of $G - \widehat{Z}$, $N(B)$ contains at least one vertex from $V(C)$ and therefore, $|\widehat{Z} \cup N(B)| > |\widehat{Z}|$, indicating that we make progress towards the upper bound of $\alpha_k + k$ on $\widehat{Z}$.

The correctness follows from the fact that the branching is exhaustive (see Lemma 14). Now we bound the running time of the algorithm Alg-special. Notice that when $\widehat{Z} > \alpha_k + k$, the algorithm immediately outputs NO. Moreover, notice that the algorithm only recurses in Cases (ii) and (iii), where the size of $\widehat{Z}$ strictly increases. The number of recursive calls made in either of these cases is dominated by Case (iii), which is upper bounded by $2^{\mathcal{O}(\alpha_k + k)}$ (see Proposition 6). Hence, we conclude that the number of nodes in the branching tree is bounded by $2^{\mathcal{O}((\alpha_k+k)^2)}$. The running time at each node is upper bounded by the brute-force solution in Case (i) (which is bounded by $2^{\mathcal{O}((\alpha_k+k))} n^{\mathcal{O}(1)}$), plus the time to compute J_C and run the algorithm of Proposition 6 $|J_C|$ times. Hence, we conclude that the running time at each node is bounded by $2^{\mathcal{O}((\alpha_k+k))} n^{\mathcal{O}(1)}$, giving us a bound of $2^{\mathcal{O}((\alpha_k+k)^2)} n^{\mathcal{O}(1)}$ on the running time of Alg-special. This completes the proof of the lemma. ◀

Lemma 15, Lemma 10 and Proposition 9 imply Theorem 1.

4 Discussions and future work

We believe that our non-uniform FPT algorithm can be made uniform by replacing the black box invocation of the meta-theorem in Proposition 9 (which is the cause for non-uniformity) with a hand-crafted application of the recursive understanding technique (see, for example, [7]). Indeed, one of the main motivations behind this meta-theorem as given by the authors of [19], is to simplify classification results by avoiding the tedious arguments required by the recursive understanding approach.

Two further natural directions for further research on this topic arise. On the one hand, for which other graph classes $\mathcal{H}$ is deciding elimination distance to $\mathcal{H}$, fixed-parameter tractable? A second direction is to investigate the parameterized complexity of various graphs problems parameterized by the elimination distance to $\mathcal{F}$-free graphs and go beyond the work of Bulian and Dawar for GI parameterized by elimination distance to bounded degree graphs. The success of the distance-from-triviality programme indicates that the study of parameterization of basic graph problems by elimination distance to well-studied graph classes is a promising direction for future research.

References

1 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.

2 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

3 Marin Bougeret, Bart M. P. Jansen, and Ignasi Sau. Bridge-depth characterizes which structural parameterizations of vertex cover admit a polynomial kernel. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8–11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 16:1–16:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.16.

4 Adam Bouland, Anuj Dawar, and Eryk Kopczynski. On tractable parameterizations of graph isomorphism. In Dimitrios M. Thilikos and Gerhard J. Woeginger, editors, *Parameterized and Exact Computation – 7th International Symposium, IPEC 2012, Ljubljana, Slovenia, September 12–14, 2012. Proceedings*, volume 7535 of *Lecture Notes in Computer Science*, pages 218–230. Springer, 2012. doi:10.1007/978-3-642-33293-7_21.

5 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.

6 Jannis Bulian and Anuj Dawar. Fixed-parameter tractable distances to sparse graph classes. *Algorithmica*, 79(1):139–158, 2017. doi:10.1007/s00453-016-0235-7.

7 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.

8 Bruno Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Inform. and Comput.*, 85:12–75, 1990.

9 Bruno Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In *Handbook of graph grammars and computing by graph transformation, Vol. 1*, pages 313–400. World Sci. Publ, River Edge, NJ, 1997.

10 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.

11 Eduard Eiben, Robert Ganian, Thekla Hamm, and O-joung Kwon. Measuring what matters: A hybrid approach to dynamic programming with treewidth. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26–30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*, pages 42:1–42:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.MFCS.2019.42.

12 Eduard Eiben, Robert Ganian, and Stefan Szeider. Meta-kernelization using well-structured modulators. In Thore Husfeldt and Iyad A. Kanj, editors, *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16–18, 2015, Patras, Greece*, volume 43 of *LIPIcs*, pages 114–126. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.IPEC.2015.114.

13 Fedor V. Fomin and Yngve Villanger. Treewidth computation and extremal combinatorics. *Comb.*, 32(3):289–308, 2012. doi:10.1007/s00493-012-2536-z.

14 Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. Going beyond primal treewidth for (M)ILP. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4–9, 2017, San Francisco, California, USA*, pages 815–821. AAAI Press, 2017. URL: `http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14272`.

15 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Combining treewidth and backdoors for CSP. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8–11, 2017, Hannover, Germany*, volume 66 of *LIPIcs*, pages 36:1–36:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. `doi:10.4230/LIPIcs.STACS.2017.36`.

16 Jiong Guo, Falk Hüffner, and Rolf Niedermeier. A structural view on parameterizing problems: Distance from triviality. 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 162–173. Springer, 2004. `doi:10.1007/978-3-540-28639-4_15`.

17 Eva-Maria C. Hols, Stefan Kratsch, and Astrid Pieterse. Elimination distances, blocking sets, and kernels for vertex cover. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10–13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, pages 36:1–36:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.STACS.2020.36`.

18 Alexander Lindermayr, Sebastian Siebertz, and Alexandre Vigny. Elimination distance to bounded degree on planar graphs. In Javier Esparza and Daniel Král’, editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24–28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 65:1–65:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.MFCS.2020.65`.

19 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Reducing CMSO model checking to highly connected graphs. 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 135:1–135:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. `doi:10.4230/LIPIcs.ICALP.2018.135`.

A Unified Framework of Quantum Walk Search

Simon Apers
Université libre de Bruxelles, Brussels, Belgium
CWI, Amsterdam, The Netherlands

András Gilyén
California Institute of Technology, Pasadena, CA, USA

Stacey Jeffery
QuSoft, CWI, Amsterdam, The Netherlands

Abstract

Many quantum algorithms critically rely on quantum walk search, or the use of quantum walks to speed up search problems on graphs. However, the main results on quantum walk search are scattered over different, incomparable frameworks, such as the *hitting time framework*, the *MNRS framework*, and the *electric network framework*. As a consequence, a number of pieces are currently missing. For example, recent work by Ambainis et al. (STOC'20) shows how quantum walks starting from the stationary distribution can always *find* elements quadratically faster. In contrast, the electric network framework allows quantum walks to start from an arbitrary initial state, but it only *detects* marked elements.

We present a new quantum walk search framework that unifies and strengthens these frameworks, leading to a number of new results. For example, the new framework effectively *finds* marked elements in the electric network setting. The new framework also allows to interpolate between the hitting time framework, minimizing the number of walk steps, and the MNRS framework, minimizing the number of times elements are checked for being marked. This allows for a more natural tradeoff between resources. In addition to quantum walks and phase estimation, our new algorithm makes use of *quantum fast-forwarding*, similar to the recent results by Ambainis et al. This perspective also enables us to derive more general complexity bounds on the quantum walk algorithms, e.g., based on Monte Carlo type bounds of the corresponding classical walk. As a final result, we show how in certain cases we can avoid the use of phase estimation and quantum fast-forwarding, answering an open question of Ambainis et al.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Quantum complexity theory

Keywords and phrases Quantum Algorithms, Quantum Walks, Graph Theory

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.6

Related Version *Full Version*: https://arxiv.org/abs/1912.04233

Funding *Simon Apers*: Work done while being partially supported by the CWI-Inria International Lab and the Dutch Research Council (NWO) through QuantERA ERA-NET Cofund project QuantAlgo 680-91-034.
András Gilyén: Funding provided by Samsung Electronics Co., Ltd., for the project "The Computational Power of Sampling on Quantum Computers", as well as by the Institute for Quantum Information and Matter, an NSF Physics Frontiers Center (NSF Grant PHY-1733907).
Stacey Jeffery: Supported by an NWO Veni Innovational Research Grant under project number 639.021.752, an NWO WISE Grant, and QuantERA project QuantAlgo 680-91-03. SJ is a CIFAR Fellow in the Quantum Information Science Program.

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 6; pp. 6:1–6:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Quantum walk search refers to the use of quantum walks to solve a search problem on a graph. Such algorithms can often be thought of as speeding up a classical algorithm based on a random walk, which makes them relatively easy to design, using only classical intuition. In the last two decades, this topic has received a great deal of attention, with a rich literature attesting to the progress on understanding quantum walk algorithmic techniques [1, 25, 3, 22, 19, 7, 13, 2] and developing applications [11, 21, 16, 8, 9, 23, 17, 14, 18]. Despite this long line of progress, the main results on quantum walk search lie somewhat scattered in different frameworks, and a number of pieces are currently missing.

The quantum walk search frameworks that we consider are the *hitting time framework* originally due to Szegedy [25], the *MNRS framework* due to Magniez, Nayak, Roland and Santha [22], the *electric network framework* due to Belovs [7], and the *controlled quantum amplification framework* by Dohotaru and Høyer [13]. We summarize these frameworks, as well as the corresponding complexities, in Table 1. In this work we unify these different frameworks, leading to a number of new results. For example, algorithms developed using the electric network framework could only *detect* marked elements. Our unified approach can be used to develop algorithms that *find* marked elements, while incurring at most a logarithmic overhead. We also derive quantum speedups with respect to Monte Carlo type bounds on the hitting times, as opposed to the usual Las Vegas type (expectation) bounds.

Our result bridges the conceptual gaps between the recent result of Ref. [2] and the original approaches by Szegedy [25] and Krovi, Magniez, Ozols and Roland [19]. The latter showed that combining quantum walks with phase estimation or time averaging allows one to quadratically improve the hitting time of a single marked element, when starting from the stationary distribution. Ambainis et al. [2] used a novel technique called *quantum fast-forwarding* [6] to improve these results to yield quadratic speedups on the hitting time of arbitrary sets. In this work we show that a similar result can be obtained using only simple quantum walks, thereby proving a conjecture from [2].

Different Frameworks

While the frameworks we consider are similar, each has advantages and disadvantages. The earliest *hitting time framework* was due to Szegedy [25], inspired by an algorithm of Ambainis for element distinctness [1]. To illustrate this framework, imagine a classical algorithm that begins by sampling a state from the stationary distribution π of some random walk, described by a transition matrix P. The algorithm starts from a vertex distributed according to π, and simulates the random walk. After every step of the walk it checks whether the current vertex is "marked". The algorithm terminates after $\mathcal{O}(\mathrm{HT}(P, M))$ steps, with $\mathrm{HT}(P, M)$ the *hitting time*, i.e., the expected number of steps from π before a marked vertex in M, the marked set, is reached. As such, the algorithm has a constant probability of having found a marked vertex. To bound the complexity of this algorithm, let the *setup cost* $\mathcal{S}$ denote the complexity of sampling from π, the *update cost* $\mathcal{U}$ denote the complexity of simulating a step of the walk, and the *checking cost* $\mathcal{C}$ denote the complexity of checking whether a vertex is marked. The complexity of the resulting algorithm is then of order $\mathcal{S} + \mathrm{HT}(P, M)(\mathcal{U} + \mathcal{C})$. The hitting time framework essentially shows how to construct a quantum algorithm with complexity

$$\mathsf{S} + \sqrt{\mathrm{HT}(P, M)}(\mathsf{U} + \mathsf{C}),$$

where S, U and C are quantum analogues of $\mathcal{S}$, $\mathcal{U}$ and $\mathcal{C}$, respectively, denoting the costs in terms of *coherent* quantum samples. One of the major drawbacks of the original framework was that, while the quantum algorithm could *detect* the presence of a marked vertex, it was not always guaranteed to *find* one. In the special case where there is only a single marked element, Krovi, Magniez, Ozols, and Roland [19] showed how to also find the marked element in the same complexity. To this end they introduced the concept of *interpolated walks*. Combining interpolated walks with another technique called *quantum fast-forwarding*, introduced in [6], Ref. [2] more recently showed how to also find a marked element in the general case. We will refer to this final result as the hitting time framework.

The second framework that we consider is the *MNRS framework* introduced by Magniez, Nayak, Roland and Santha [22]. This framework is always capable of *finding* a marked vertex, but it can be understood as the quantum analogue of a slightly different random walk algorithm. Consider a random walk that begins in the stationary distribution. Rather than checking if the current vertex is marked after every step, the walk takes $1/\delta$ steps between checks, where δ is the *spectral gap* of P. Since $1/\delta$ is approximately the *mixing time* of the random walk, effectively this process repeatedly samples from the stationary distribution until a marked vertex is found during a check. If ε is the probability that a vertex sampled from the stationary distribution is marked, then a marked element is found with constant probability after $\mathcal{O}(1/\varepsilon)$ samples.[1] As such, the complexity of this classical algorithm is $\mathcal{S} + \frac{1}{\varepsilon}(\frac{1}{\delta}\mathcal{U} + \mathcal{C})$. The MNRS framework shows how to get a quantum algorithm for finding a marked vertex with complexity

$$\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\Big(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\Big).$$

Since $\mathrm{HT}(P, M) \leq \frac{1}{\varepsilon\delta}$, this requires at least as many steps of the walk as the hitting time framework. On the other hand, $\mathrm{HT}(P, M) \geq \frac{1}{\varepsilon}$, and so the number of checks can be significantly smaller than in the hitting time framework. In fact, this amount of checks performed in the MNRS framework is easily seen to be optimal by a lower bound on black-box search.[2]

The third framework that we consider is the *electric network framework* by Belovs [7] (published in [8]). This is a generalization of the hitting time framework, allowing for the walker to start from an arbitrary initial distribution σ (such as a single vertex), rather than necessarily the stationary distribution. If $\mathsf{S}(\sigma)$ is the complexity of sampling (coherently) from σ, then the resulting quantum algorithm has complexity

$$\mathsf{S}(\sigma) + \sqrt{C_{\sigma,M}}(\mathsf{U}(\sigma) + \mathsf{C}),$$

where $\mathsf{U}(\sigma)$ is the complexity of implementing a step of a slightly modified random walk. The quantity $C_{\sigma,M}$ (see Section 3.1) is a generalization of the *commute time*. When σ is supported on a single vertices u, then $C_{\sigma,M}$ equals the commute time from u to M, which is the expected number of steps starting from u to reach some $m \in M$ and then return to u. When σ equals the stationary distribution then $C_{\sigma,M} = HT(P, M)$, thus retrieving the hitting time framework. The obvious advantage of the electric network framework is that it

[1] This yields a trivial classical search algorithm with complexity $(\mathcal{S} + \mathcal{C})/\varepsilon$ which does not walk just uses sampling. By using amplitude amplification this gives rise to a quantum algorithm with complexity $(\mathsf{S} + \mathsf{C})/\sqrt{\varepsilon}$. In case the cost of setup is much larger than that of checking, a random walk that uses the setup only once can be advantageous, potentially leading to interesting quantum walk based speedups.

[2] Consider for instance a quantum walk search algorithm on the complete graph on N vertices. Finding a single marked element then requires $\Omega(\sqrt{N})$ checks by the optimality of Grover's search algorithm.

Table 1 Comparison of different quantum walk frameworks.

Framework	Complexity	Finds?
Hitting time framework [25, 19, 2]	$\mathsf{S} + \sqrt{\mathrm{HT}(P, M)}(\mathsf{U} + \mathsf{C})$	Yes
MNRS framework [22]	$\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C})$	Yes
Electric network framework [8, 7]	$\mathsf{S}(\sigma) + \sqrt{C_{\sigma,M}}(\mathsf{U}(\sigma) + \mathsf{C})$	No
Controlled quantum amplification [13]	$\mathsf{S} + \sqrt{\mathrm{HT}(P, \{m\})}\mathsf{U} + \frac{1}{\sqrt{\varepsilon}}\mathsf{C}$	Yes

does not necessarily require quantum samples from the stationary distribution of P, which might be very costly, and can instead begin in a much easier to produce state. A major disadvantage of this framework, however, is that the quantum algorithm only *detects* the presence of marked vertices, as in the original hitting time framework, rather than actually finding marked vertices.

Finally, we also consider the *controlled quantum amplification framework* by Dohotaru and Høyer [13]. They use an extra qubit to control the quantum walk operator[3], leading to an additional degree of freedom. For the case of a *unique* marked element $M = \{m\}$, and starting from a quantum sample of the stationary distribution, they achieve complexity

$$\mathsf{S} + \sqrt{\mathrm{HT}(P, \{m\})}\mathsf{U} + \frac{1}{\sqrt{\varepsilon}}\mathsf{C},$$

which simultaneously has the lowest number of walk steps (matching the hitting time framework) and the lowest number of checks (matching the MNRS framework). The clear downside of this approach is that it is restricted to cases where there is a single marked element, and one needs to start from the stationary distribution.

2 Contributions

2.1 Finding in the electric network framework

The electric network framework [7] generalizes the hitting time framework [25] by allowing for arbitrary initial distributions. The downside is that algorithms in this framework only detect rather than actually find marked vertices. On the other hand, the improved hitting time framework of [2] shows how to actually find marked vertices in the hitting time framework, provided that the walk starts from a quantum sample of the stationary distribution. Both works hence provide complementary but incomparable improvements over the initial hitting time framework.

In this work we fill this gap by generalizing the results of [2] to the electric network setting, designing a quantum algorithm that not only detects but also *finds* marked elements for any starting distribution σ. This improved version strictly generalizes the results of [2], and it loses at most a log factor in the complexity compared to the original electric network framework [7]. In particular, we show the following:

▶ **Theorem 1** (Informal). *For any distribution σ, there is a quantum walk search algorithm that finds a marked element from M with constant probability in complexity (up to log factors)*

$$\mathsf{S}(\sigma) + \sqrt{C_{\sigma,M}}(\mathsf{U}(\sigma) + \mathsf{C}).$$

[3] In fact they consider more general operators, but we will focus on their result about quantum walk search.

To analyze our new algorithm, we use techniques similar to those employed in [2] for finding in the hitting time framework. However, there is an additional difficulty we must overcome. The hitting time, $\mathrm{HT}(P, M)$, has a useful interpretation in terms of the classical random walk – that is, with high probability, a marked vertex is encountered within the first $\mathcal{O}(\mathrm{HT}(P, M))$ steps – and this fact is crucial in the analysis of the quantum algorithm in [2]. In contrast, to the best of our knowledge, the generalized quantity $C_{\sigma,M}$ is not well understood. If σ is supported on a single vertex u, and M contains a single vertex, m, then $C_{\sigma,M}$ is exactly the *commute time* between u and m. This means that within the first $\mathcal{O}(C_{\sigma,M})$ steps, with high probability, a walker starting from u has visited m, and then returned to u. For general σ and M, no such interpretation was known. We prove that, under certain conditions, a similar interpretation holds: with high probability, a walker starting from σ will hit M, and then return to the support of σ, within the first $\mathcal{O}(C_{\sigma,M})$ steps. We can ensure that the required conditions hold by using a slightly modified walk as in [7], adding a weighted edge to each vertex in $\mathrm{supp}(\sigma)$. The new combinatorial understanding of $C_{\sigma,M}$ then enables us to employ a similar analysis to that of [2].

2.2 A Unified Framework

While the electric network framework is a generalization of the hitting time framework, the MNRS framework is incomparable. Since $\mathrm{HT}(P, M) \le \frac{1}{\varepsilon\delta}$, the hitting time framework always finds a marked vertex using a number of quantum walk steps (updates) less than or equal to that used by the MNRS framework. On the other hand, $\mathrm{HT}(P, M) \ge \frac{1}{\varepsilon}$, and hence the MNRS framework may make fewer calls to the check operation. When the complexity of implementing the checking operation is much larger than that of the update operation, the MNRS framework may hence be preferable to both the hitting time framework and the electric framework. The controlled quantum amplification framework achieves the best of both worlds, but only for a unique marked element.

In this work we present a new framework that unifies all these individual approaches. For the sake of intuition, we first describe this framework when the initial state π is used, which can be seen as a unification between the hitting time framework, the MNRS framework and the controlled quantum amplification framework. To this end, recall that the hitting time framework is the quantum analogue of a random walk algorithm that takes $\mathrm{HT}(P, M)$ steps of the random walk described by P, checking at each step if the current vertex is marked. In contrast, the MNRS framework is the quantum analogue of a random walk algorithm that takes $\frac{1}{\delta}$ steps of P, thus approximately sampling from the stationary distribution π, and then checks if the sampled vertex is marked. Since ε is the probability that a sampled vertex is marked, this process is repeated $\frac{1}{\varepsilon}$ times.

We can define a natural interpolation between both classical algorithm. To this end, take any t, and consider a classical random walk that repeatedly takes t steps, and then checks whether the current vertex is marked. The expected number of iterations is then $\mathrm{HT}(P^t, M)$, the hitting time of the t-step random walk, described by transition matrix P^t. This classical algorithm finds a marked vertex in complexity $\mathcal{S} + \mathrm{HT}(P^t, M)(t\mathcal{U} + \mathcal{C})$. We give a quantum analogue of this algorithm, generalized to arbitrary[4] initial distributions.

[4] In case a the initial distribution σ differs from π we need to account for the complexity $\mathsf{U}(\sigma)$ of the modified update operator, see the full version [5] for details.

▶ **Theorem 2** (Informal). *For any $t \in \mathbb{N}$ and any distribution σ, there is a quantum walk search algorithm that finds a marked element from M with constant probability in complexity (up to log factors)*

$$\mathsf{S}(\sigma) + \sqrt{C_{\sigma,M}(P^t)}(\sqrt{t}\mathsf{U}(\sigma) + \mathsf{C}).$$

This theorem and the corresponding quantum walk algorithm gives a common generalization of all major previous quantum walk algorithms, and recovers their complexity in the corresponding special cases (up to log factors). Indeed setting $t = 1$ we recover Theorem 1. When $\sigma = \pi$, then $C_{\sigma,M}(P^t) = \mathrm{HT}(P^t, M)$, and hence we find the quantum analogue of the aforementioned random walk algorithm. As such, when $\sigma = \pi$ and $t = 1$, we recover the hitting time framework. When $\sigma = \pi$ and $t = \frac{1}{\delta}$, we recover the MNRS framework, since a $1/\delta$-step random walk essentially samples from π at every step, and so $\mathrm{HT}(P^{1/\delta}, M) \in \mathcal{O}\big(\frac{1}{\varepsilon}\big)$. Setting $t = \varepsilon\mathrm{HT}(P, \{m\})$, we recover the controlled quantum amplification framework. To see this, we use a result from [13, Section 6] which proves that $\mathrm{HT}(P^t, \{m\}) = 1/\varepsilon$ if $t \in \Omega(\varepsilon\mathrm{HT}(P, \{m\}))$. For multiple marked elements, and other intermediate values of t, we obtain new types of algorithms. We summarize these special cases in the table below.

Table 2 The new quantum walk search framework.

New quantum walk search framework:	$\mathsf{S}(\sigma) + \sqrt{C_{\sigma,M}(P^t)}(\sqrt{t}\mathsf{U}(\sigma) + \mathsf{C})$
Hitting time framework	$\sigma = \pi$, $t = 1$
MNRS framework	$\sigma = \pi$, $t = \frac{1}{\delta}$
Electric network framework	any σ, $t = 1$
Controlled quantum amplification	$\sigma = \pi$, $M = \{m\}$, $t = \varepsilon\mathrm{HT}(P, M)$

An improved application: backtracking

Electric network based quantum walks have several applications, but arguably the most well-known application is Montanaro's quantum speedup for backtracking algorithms [23]. Montanaro's algorithm uses quantum walk based search in order to find a marked vertex in a bounded degree tree structure, with an unknown structure, starting from the root. The binary tree corresponds to a search tree typically coming from a constraint satisfaction problem. Since already partial assignments can lead to unsatisfiable constraints, the corresponding parts of the search tree are pruned in order to increase performance. This means that the graph on which the walk is performed is a priori unknown, but can be explored locally.

Since earlier versions of the electric network based walks did not always allow finding a marked element, Montanaro needed to construct a modified algorithm that could always find a marked vertex by directly exploiting the special tree structure. If the search tree has size T, and the depth of the tree is at most n, then Montanaro's algorithm finds a marked vertex in time

$$\widetilde{\mathcal{O}}\Big(\sqrt{T}n^{3/2}\log(n)\Big).$$

In order to understand the performance of our walk for finding a marked element, we need to understand the quantities in Theorem 2. Since we have a simple graph, we can assume that every edge has weight 1 initially, and $C_{r,M} = TR_{r,M}$, where T is the number of edges in the tree, and $R_{r,M}$ is the effective resistance between the root r and the set marked vertices M (cf. Section 3.1). Due to Theorem 2 (for more formal details, see [5]) we can find a marked element in complexity $\mathcal{O}\Big(\sqrt{TR_{r,M}\log(TR_{r,M})\log\log(TR_{r,M})}\Big)$. If there is a marked element (or solution), then there is a path of length $\leq n$ for m the root to a marked element, which means that the effective resistance is $R_{r,M} \leq n$. Also by using $\log(T) \in \mathcal{O}(n)$ we can see that a corollary of our results is that we can find a marked element in complexity

$$\mathcal{O}\Big(\sqrt{T\log(n)}n\Big),$$

which gives a $\sqrt{n\log(n)}$ factor improvement over the result of Montanaro.

For comparison, we shall note that another improvement over Montanaro's algorithm was achieved by [15]. Their algorithm has complexity

$$\mathcal{O}\Big(\sqrt{TR_{\max}}\log^4(kR_{r,M})\Big), \tag{1}$$

where $R_{\max}$ is the maximal finite ($< \infty$) effective resistance over subtrees of the graph, and k is the number of marked vertices. If there are $\mathcal{O}(1)$ marked vertices, then the complexity (1) can be bounded as $\widetilde{\mathcal{O}}\Big(\sqrt{Tn}\log^4(n)\Big)$, which is asymptotically even lower than our bound. However, the interplay between $R_{\max}$, $R_{r,M}$, n and k suggest that our complexity will be typically lower when there is a fair number ($> 2^{\sqrt[4]{n\log(n)}}$) of marked vertices.

Finally, we note that another improvement over Montanaro's algorithm was suggested by Ambainis and Kokainis [4], employing a clever binary search on the size of the search subtrees T in order to improve the overall runtime. This improvement relies on adaptively modifying the search graph, rather than optimizing the walk on a fixed graph, and therefore this approach does not directly fit into our framework.

2.3 Bounds based on Monte Carlo type guarantees of the classical walk

We study quantum walk speedups with respect to Monte Carlo type hitting time bounds of the corresponding classical random walks. Monte Carlo type hitting times can be upper bounded in terms of the Las Vegas hitting time of the same walk, but can be also substantially smaller (see the full version, [5], for an explicit example). Therefore, a quadratic speedup in terms of Monte Carlo type hitting times is preferable compared to the usual Las Vegas bounds.

Monte Carlo type hitting time bounds and quadratic speedups for the detection problem were studied in [20], however, to our knowledge quadratic speedups for finding a marked vertex were only established for vertex-transitive graphs with a unique marked vertex. Using our techniques we are able to show that a quadratic speedup (up to a log factor) for finding a marked vertex can be always achieved for any random walk on undirected weighted graphs.

We define the *Monte Carlo hitting time for probability* p, denoted $\mathrm{HT}_p(P,M)$, as the minimum number of time-steps t, such that P visits a marked vertex within the first t steps with probability at least p, when started from the stationary distribution π.

▶ **Theorem 3.** *Given an upper bound* $\mathrm{HT}_p(P,M) \leq \mathrm{HT}_p$ *on the Monte Carlo hitting time of a reversible Markov Chain* P*, there is a quantum algorithm that finds a marked vertex with high probability in complexity*

$$\sqrt{\frac{\log(\mathrm{HT}_p)}{p}}\left(\mathsf{S} + \sqrt{\mathrm{HT}_p\log(\log(\mathrm{HT}_p)/p)}(\mathsf{U}+\mathsf{C})\right).$$

Versions of the above theorem including the trade-off analogous to Theorem 2 also easily follow from our framework. Additionally, in the full version, we also prove some related Monte Carlo type bounds for the general case when we start in an arbitrary distribution σ as opposed to the stationary.

2.4 Simpler algorithm for hitting time and electric network framework

Similar to the recent work by Ambainis et al. [2], our new quantum algorithm makes use of a somewhat involved technique called quantum fast-forwarding. For the case $t = 1$ (recovering the hitting time and electric network framework), we show that our algorithm can be much simplified while maintaining essentially the same complexity guarantees. The simple algorithm works by (classically) choosing random interpolation parameters, and applying the interpolated quantum walk operator for an appropriately chosen (random) number of steps, starting from $|\sqrt{\sigma}\rangle$, the quantum state whose amplitudes are the square roots of the probabilities of σ. It was already conjectured in [2] that such a simple approach would work (for the special case of the hitting time framework). In this work we prove that this simple algorithm indeed finds a marked vertex, and incurs at most a logarithmic overhead over the complexity of the more involved fast-forwarding algorithm. Interestingly, our proof relies on the proof of correctness of the fast-forwarding algorithm.

2.5 Related independent work

While finalizing this manuscript, the authors became aware of the concurrent and independent work of Stephen Piddock, who developed an alternative refinement of Belovs' results for finding marked elements in the electric network framework [24].

3 Summary of technical contributions

We now give a high-level summary of some of our technical results. For the sake of intuition, we suppose that the random walk P is symmetric – this assumption is not necessary for our results, but serves to simplify the overview of this section. For the fully detailed statements and proofs of our results, see the technical version of this article [5].

3.1 Finding in the electric network framework

The electric network framework is almost a strict generalization of the hitting time framework, except that in the hitting time framework we know how to not only detect, but actually find a marked vertex, whereas the electric network framework only detects the presence of a marked vertex. Our first contribution is to show how to find in the electric network framework. The resulting framework thus generalizes both the hitting time and electric network frameworks.

Interpolated walks

Letting P be the transition matrix of a random walk, and $s \in [0, 1]$ an *interpolation parameter*, we define $P(s)$ as the transition matrix of the *interpolated walk* – the random walk that acts as P, except that when a marked vertex is encountered, with probability s, the walker remains in that vertex, and with probability $1 - s$, the walker transitions according to P. That is, if we let P_M denote the *absorbing walk* that behaves as P, except that for any vertex $u \in M$, a walker at u remains there with probability 1, then we can write[5]

$$P(s) = (1 - s)P + sP_M.$$

[5] Note that even if P was symmetric, the interpolated walk $P(s)$ can be non-symmetric, however it is reversible. Fortunately, our results generalize to any reversible (non-symmetric) Markov Chain as well.

Note that if s is sufficiently close to 1, after $10\mathrm{HT}(P,M)$ steps of $P(s)$ starting from the stationary distribution π, the walker will be in a marked vertex with high probability. In other words, letting Π_M denote the orthogonal projector onto marked vertices, $\Pi_M P(s)^{10\mathrm{HT}(P,M)}\pi$ will be large in ℓ_1-norm.

If $\mathsf{U}(P)$ is the complexity of taking one step of the random walk P, and C is the complexity of checking if a vertex is marked, then one step of the random walk $P(s)$ can be implemented in complexity $\mathsf{U}(P(s)) = \mathcal{O}(\mathsf{U}(P)+\mathsf{C})$.

Quantum fast-forwarding and finding in the hitting time framework

The original hitting time framework also suffered from the drawback that it could detect but not find marked vertices, until this was recently improved in Ref. [2]. This improvement was based on a technique called *quantum fast-forwarding* [6], which shows how, for a symmetric random walk P, to map any state $|\psi\rangle$ to within ξ-distance of some state of the form $|0\rangle P^t|\psi\rangle + |1\rangle|\gamma\rangle$, where $\||\gamma\rangle\|^2 = 1 - \|P^t|\psi\rangle\|^2$, in complexity $\mathcal{O}\Big(\sqrt{t\log\frac{1}{\xi}}\mathsf{U}(P)\Big)$.

The algorithm of Ref. [2] first checks if the initial state is marked, then applies fast-forwarding to the absorbing walk $P(s)$ for some appropriately chosen s, using $|\sqrt{\pi}\rangle$ as the initial state, resulting in a state of the form $|0\rangle P(s)^t(I-\Pi_M)|\sqrt{\pi}\rangle + |1\rangle|\gamma\rangle$ for $t \approx \mathrm{HT}(P,M)$, using roughly $\sqrt{\mathrm{HT}(P,M)}$ calls to the update operator for $P(s)$, for a total cost of about $\sqrt{t}\mathsf{U}(P(s)) \approx \sqrt{\mathrm{HT}(P,M)}(\mathsf{U}(P)+\mathsf{C})$. The probability that measuring this state results in a marked vertex is $\|\Pi_M P(s)^t(I-\Pi_M)|\sqrt{\pi}\rangle\|^2$. Ref. [2] lower bounds this probability by the square of the probability that a random walk, beginning from the distribution π, is in a marked vertex after t steps, and then in an unmarked vertex again after t' steps, for *any* $t' > t$. That is, for all $t' > t$:

$$\left\|\Pi_M P(s)^t(I-\Pi_M)|\sqrt{\pi}\rangle\right\|^2 \geq \Pr_{Y_0\sim\pi}(Y_0 \notin M, Y_t \in M, Y_{t'} \notin M)^2. \tag{2}$$

Then all that remains is to prove a statement about the classical random walk: for some appropriately chosen s, and some value $t' > t$, with high probability, after $t \approx \mathrm{HT}(P,M)$ steps, a random walker is in a marked vertex – easy to achieve by taking s to be sufficiently close to 1 – and after t' steps, the walker is not in a marked vertex – necessitating that s is not *too* close to 1. The proof of this, while not straightforward, relies only on combinatorial arguments about the classical random walk.

The electric network framework

A similar statement to Equation (2) can be proven for the electric network framework, but utilizing it requires significantly more work in understanding the classical random walk.

The electric network framework can be understood in analogy to a classical random walk that begins in an arbitrary distribution σ, and walks until a marked vertex is found. The quantum analogue of this process begins in the quantum state $|\sqrt{\sigma}\rangle = \sum_u \sqrt{\sigma_u}|u\rangle$, and takes $\sqrt{C_{\sigma,M}}$ steps of a random walk $P'(s)$ that is similar to an interpolated walk $P(s)$, except that each vertex $u \in \mathrm{supp}(\sigma)$ has an extra edge, connecting it to a new degree-1 vertex, with weight proportional to σ_u, see Figure 1 (and the full version, [5], for more explanation).

Using fast-forwarding, we can find a marked vertex with probability $\|\Pi_M P'(s)^t|\sqrt{\sigma}\rangle\|^2$, in complexity $\mathsf{S}(\sigma) + \sqrt{t}(\mathsf{U}+\mathsf{C})$ as in [2]. Analogous to Equation (2), we can show that for any $t' > t$, this probability is lower bounded by:

$$\left\|\Pi_M P'(s)^t|\sqrt{\sigma}\rangle\right\|^2 \geq \Pr_{Y_0\sim\sigma}(Y_t \in M, Y_{t'} \in \mathrm{supp}(\sigma))^2. \tag{3}$$

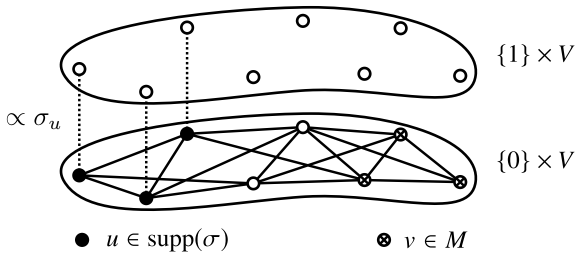


Figure 1 Modified graph.

Taking $t \approx C_{\sigma,M}$ to be roughly the commute time, we get an algorithm with the right complexity, and it remains only to show that there is some appropriate choice of s, and some value $t' > t$ such that with high probability, a random walk beginning in the distribution σ is in a marked state after $C_{\sigma,M}$, and has returned to the support of σ after t' steps. This requirement on returning to the support of σ sheds a light on why the *commute* time is the relevant quantity for the analysis of the quantum walk. Although, the commute time has some intuitive combinatorial meaning in some special cases, for example when σ is supported on a single vertex, in the general case it is difficult to grasp the intuitive interpretation of $C_{\sigma,M}$. Our main technical contribution here is to give such an interpretation, that can then be used to lower bound the expression in Equation (3).

Interpretation of $C_{\sigma,M}$

A weighted graph can be thought of as modeling an electrical network, where an edge e of weight w_e represents a resistor with resistance $1/w_e$ (or equivalently, conductance w_e). The effective resistance $R_{\sigma,M}$ denotes the minimum energy of any *unit flow* entering the graph with current σ_u at vertex u, and exiting the graph through the vertices of M. Then if W denotes the *total weight* of all edges in the graph, we define $C_{\sigma,M} = WR_{\sigma,M}$. To motivate this perhaps strange-seeming definition, we highlight two special cases:

- When $\sigma = \pi$ is the stationary distribution of the graph, $C_{\sigma,M} = \text{HT}(P, M)$.
- When σ is supported on a unique vertex s, and $M = \{t\}$ is a singleton, then $C_{\sigma,M}$ is the *commute time* between s and t, or the expected number of steps starting from s to reach t and then return to s.

However, to the best of our knowledge, no such operational interpretation of $C_{\sigma,M}$ in the general case is known. In order to lower bound the expression in Equation (3), we would like to be able to say that, for appropriately chosen parameter s, with high probability, after $t \approx C_{\sigma,M}$ steps, a random walker beginning in distribution σ will be in a marked vertex, and after sufficiently many more steps, the random walker will have returned to the support of σ. In the special cases outlined above, there is intuitive reason to believe that such a statement should hold, but in the general case, there is no interpretation that would lead us to believe that such a statement is true.

We show that, under certain assumptions (that are always satisfied for the modified graphs described above, cf. Figure 1), we can interpret $C_{\sigma,M}$ as a kind of commute time between the support of σ and the marked set M. In particular, in the full version, [5], we prove the following, which is sufficient to give a lower bound on Equation (3):

▷ Claim 4 (Informal). Whenever σ is proportional to π on its support, and $\pi(\text{supp}(\sigma)) \approx \frac{1}{C_{\sigma,M}}$, with high probability, a random walk starting from σ first hits M and then returns to $\text{supp}(\sigma)$ within $10C_{\sigma,M}$ steps.

The condition that σ be proportional to π on its support – that is, there exists α such that $\sigma_u = \alpha\pi_u$ whenever $\sigma_u \neq 0$ – may seem strong, but we can satisfy it by modifying the graph by adding an edge to each $u \in \operatorname{supp}(\sigma)$ to a new vertex of degree 1, with edge weight proportional to σ_u. If we call this new vertex u (and appropriately rename the vertex formerly called u), then this change has negligible impact on the dynamics of a random walk starting from σ, but it ensures that the stationary probability of each of these new vertices u is proportional to σ_u. This graph modification is already made in the original electric network framework, so we automatically satisfy the first condition required for the claim. The second condition is satisfied by appropriately scaling the weights of the new edges.

3.2 A unified framework

Once we show how to find in the electric network framework, it is a strict generalization of the hitting time framework. However, it is still incomparable to the MNRS framework. Our next contribution is to give a new framework that captures both the MNRS framework and the electric network framework as a special case, and also recovers the main application of the controlled quantum amplification framework regarding the optimized checking cost.

The main idea of the unified framework is to apply the electric framework machinery, but to the Markov chain that performs t-steps of P at once. The main technical challenge is to perform t consecutive update steps of the walk with using the update unitary only about $\sqrt{t}$ times. This can be performed using fast-forwarding, but we need to be careful, because after fast-forwarding we still need to apply the modifications to the walk required by the algorithm for the electric network framework.

Fortunately, we are able to show that these operators, namely interpolating the walk and modifying the graph on the support of σ, are compatible with fast-forwarding, if one considers a suitable and general-enough notion of the update operation. To show this we need to view the quantum walk operators from an angle which is different from the perspective of earlier quantum walk results, that did not consider fast-forwarding. To be able to grasp and extract the essential features of the update operator, we view the update operator as a block-encoding of the Markov chain (or more precisely its discriminant matrix). Then we describe the effects of the desired modifications using this formalism and then show how to implement them.

3.3 A plain quantum walk algorithm for finding marked elements

As we described above, our main quantum walk algorithm and its analysis relies on the use of quantum fast-forwarding. This makes it more complicated than the original "plain" quantum walk algorithms in e.g. [25, 22, 7]. We nevertheless show that the correctness of our algorithm implies the correctness of a much simpler algorithm, at least in the regimes corresponding to the hitting time framework and the electric network framework. Namely, if we simply pick a random interpolation parameter, run the corresponding plain quantum walk for at most about $\sqrt{C_{\sigma,M}}$ steps, and finally measure the walk register, then we find a marked element with constant probability. This algorithm was proposed in [2, Section 4] for the hitting time framework, but was only conjectured to be correct.

To derive this result, we literally "dissect" the more complicated fast-forwarding algorithm by considering an explicit construction of the quantum fast-forwarding routine based on the linear combination of unitaries technique [12, 10]. The structural properties of this quantum circuit then imply that the simpler routine should also succeed. Indeed, quantum fast-forwarding can be represented as a convex combination of plain quantum walk steps, which makes it equivalent to a stochastic algorithm which picks a random iteration number t according to the convex combination coefficients, and then runs the plain quantum walk for t steps. To illustrate this, in the full version, Ref. [5], we give a proof of the correctness of the fast-forwarding technique, and analyze the structure of the constructed circuit.

3.4 Open questions

Our unified framework resolves most open questions about quantum walk based search, and recovers essentially all major prior upper bounds (up to log factors). However, there could be room for improving the "Monte Carlo type" bounds in the case of arbitrary initial distributions. Also in the case of arbitrary initial distributions we lack a good combinatorial intuition for the quantity $C_{\sigma,M}$ appearing in our upper bounds. Finally, we lack a good general lower bound matching our upper bounds in terms of U and C simultaneously.

References

1 Andris Ambainis. Quantum walk algorithm for element distinctness. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 22–31, 2004. `arXiv:quant-ph/0311001`, `doi:10.1109/FOCS.2004.54`.

2 Andris Ambainis, András Gilyén, Stacey Jeffery, and Martins Kokainis. Quadratic speedup for finding marked vertices by quantum walks. In *Proceedings of the 52nd ACM Symposium on the Theory of Computing (STOC)*, page 412–424, 2020. `arXiv:1903.07493`, `doi:10.1145/3357713.3384252`.

3 Andris Ambainis, Julia Kempe, and Alexander Rivosh. Coins make quantum walks faster. In *Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1099–1108, 2005. `arXiv:quant-ph/0402107`.

4 Andris Ambainis and Martins Kokainis. Quantum algorithm for tree size estimation, with applications to backtracking and 2-player games. In *Proceedings of the 49th ACM Symposium on the Theory of Computing (STOC)*, page 989–1002, 2017. `arXiv:1704.06774`, `doi:10.1145/3055399.3055444`.

5 Simon Apers, András Gilyén, and Stacey Jeffery. A unified framework of quantum walk search, 2019. `arXiv:1912.04233`.

6 Simon Apers and Alain Sarlette. Quantum fast-forwarding: Markov chains and graph property testing. *Quantum Information and Computation*, 19(3&4):181–213, 2019. `arXiv:1804.02321`, `doi:10.26421/QIC19.3-4`.

7 Aleksandrs Belovs. Quantum walks and electric networks, 2013. `arXiv:1302.3143`.

8 Aleksandrs Belovs, Andrew M. Childs, Stacey Jeffery, Robin Kothari, and Frédéric Magniez. Time-efficient quantum walks for 3-distinctness. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 105–122, 2013. `doi:10.1007/978-3-642-39206-1_10`.

9 Daniel J. Bernstein, Stacey Jeffery, Tanja Lange, and Alexander Meurer. Quantum algorithms for the subset-sum problem. In *Proceedings of the 5th International Conference on Post-Quantum Cryptography (PQCrypto)*, pages 16–33, 2013. `doi:10.1007/978-3-642-38616-9_2`.

10 Dominic W. Berry, Andrew M. Childs, and Robin Kothari. Hamiltonian simulation with nearly optimal dependence on all parameters. In *Proceedings of the 56th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 792–809, 2015. `arXiv:1501.01715`, `doi:10.1109/FOCS.2015.54`.

11 Harry Buhrman and Robert Špalek. Quantum verification of matrix products. In *Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 880–889, 2006. `arXiv:quant-ph/0409035`.

12 Andrew M. Childs and Nathan Wiebe. Hamiltonian simulation using linear combinations of unitary operations. *Quantum Information and Computation*, 12(11&12):901–924, 2012. arXiv:1202.5822, doi:10.26421/QIC12.11-12.

13 Cătălin Dohotaru and Peter Høyer. Controlled quantum amplification. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 18:1–18:13, 2017. doi:10.4230/LIPIcs.ICALP.2017.18.

14 Alexander Helm and Alexander May. Subset sum quantumly in 1.17^n. In *Proceedings of the 13th Conference on the Theory of Quantum Computation, Communication, and Cryptography (TQC)*, pages 5:1–5:15, 2018. doi:10.4230/LIPIcs.TQC.2018.5.

15 Michael Jarret and Kianna Wan. Improved quantum backtracking algorithms using effective resistance estimates. *Physical Review A*, 97(2):022337, 2018. arXiv:1711.05295, doi:10.1103/PhysRevA.97.022337.

16 Stacey Jeffery, Robin Kothari, and Frédéric Magniez. Nested quantum walks with quantum data structures. In *Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1474–1485, 2012. arXiv:1210.1199.

17 Ghazal Kachigar and Jean-Pierre Tillich. Quantum information set decoding algorithms. In *Proceedings of the 8th International Conference on Post-Quantum Cryptography (PQCrypto)*, pages 69–89, 2017. arXiv:1703.00263, doi:10.1007/978-3-319-59879-6_5.

18 Elena Kirshanova. Improved quantum information set decoding. In *Proceedings of the 9th International Conference on Post-Quantum Cryptography (PQCrypto)*, pages 507–527, 2018. arXiv:1808.00714, doi:10.1007/978-3-319-79063-3_24.

19 Hari Krovi, Frédéric Magniez, Maris Ozols, and Jérémie Roland. Quantum walks can find a marked element on any graph. *Algorithmica*, 74(2):851–907, 2016. arXiv:1002.2419, doi:10.1007/s00453-015-9979-8.

20 Frédéric Magniez, Ashwin Nayak, Peter C. Richter, and Miklos Santha. On the hitting times of quantum versus random walks. *Algorithmica*, 63(1):91–116, 2012. Earlier version in SODA'09. arXiv:0808.0084 doi:10.1007/s00453-011-9521-6.

21 Frédéric Magniez, Miklos Santha, and Mario Szegedy. Quantum algorithms for the triangle problem. *SIAM Journal on Computing*, 37(2):413–427, 2007. arXiv:quant-ph/0310134, doi:10.1137/050643684.

22 Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011. Earlier version in STOC'07. arXiv:quant-ph/0608026 doi:10.1137/090745854.

23 Ashley Montanaro. Quantum-walk speedup of backtracking algorithms. *Theory of Computing*, 14(15):1–24, 2018. arXiv:1509.02374, doi:10.4086/toc.2018.v014a015.

24 Stephen Piddock. Quantum walk search algorithms and effective resistance, 2019. arXiv:1912.04196.

25 Márió Szegedy. Quantum speed-up of Markov chain based algorithms. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 32–41, 2004. arXiv:quant-ph/0401053, doi:10.1109/FOCS.2004.53.

Achieving Anonymity via Weak Lower Bound Constraints for k-Median and k-Means

Anna Arutyunova ✉
Universität Bonn, Germany

Melanie Schmidt ✉
Universität Köln, Germany

Abstract

We study k-clustering problems with lower bounds, including k-median and k-means clustering with lower bounds. In addition to the point set P and the number of centers k, a k-clustering problem with (uniform) lower bounds gets a number B. The solution space is restricted to clusterings where every cluster has at least B points. We demonstrate how to approximate k-median with lower bounds via a reduction to facility location with lower bounds, for which $O(1)$-approximation algorithms are known.

Then we propose a new constrained clustering problem with lower bounds where we allow points to be assigned *multiple times* (to different centers). This means that for every point, the clustering specifies a set of centers to which it is assigned. We call this *clustering with weak lower bounds*. We give an 8-approximation for k-median clustering with weak lower bounds and an $O(1)$-approximation for k-means with weak lower bounds.

We conclude by showing that at a constant increase in the approximation factor, we can restrict the number of assignments of every point to 2 (or, if we allow fractional assignments, to $1+\epsilon$). This also leads to the first bicritera approximation algorithm for k-means with (standard) lower bounds where bicriteria is interpreted in the sense that the lower bounds are violated by a constant factor.

All algorithms in this paper run in time that is polynomial in n and k (and d for the Euclidean variants considered).

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases Clustering with Constraints, lower Bounds, k-Means, Anonymity

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.7

Related Version *Full Version*: https://arxiv.org/abs/2009.03078 [7]

Funding *Anna Arutyunova*: DFG grant RO 5439/1-1.
Melanie Schmidt: DFG grant SCHM 2765/1-1.

Acknowledgements We thank anonymous reviewers for their detailed comments to a previous version.

1 Introduction

We study k-clustering problems with lower bound constraints. Imagine the following approach to publish a reduced version of a large data set: Partition the data into clusters of similar objects, then replace every cluster by one (weighted) point that represents it best. Publish these weighted representatives. For example, it is a fairly natural approach for data that can be modeled as vectors from $\mathbb{R}^d$ to replace a data set by a set of mean vectors, where every mean vector represents a cluster. When representing a cluster by one point, the mean vector minimizes the squared error of the representation. This is a common use case of k-means clustering.

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 7; pp. 7:1–7:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we ask the following: If we want to publish the representatives, it would be very convenient if the clusters were of sufficient size to ensure a certain level of anonymity of the individual data points that they represent. Can we achieve this, say, in the case of k-means clustering or for the related k-median problem?

Using clustering with lower bounds on the cluster sizes to achieve anonymity is an idea posed by Aggarwal et al. [3]. They introduce it in the setting of radii-based clustering, and define the *r-gather problem*: Given a set of points P from a metric space, find a clustering and centers for the clusters such that the maximum distance between a point and its center is minimized and such that every cluster has at least r points. They also define the (k, r)-center problem which is the same problem as the r-gather problem except that the number of clusters is also bounded by the given number k. So the (k, r)-center problem takes the k-center clustering objective but restricts the solution space to clusterings where every cluster has at least r points. Aggarwal et al. [3] give a 2-approximation for both problems.

We pose the same question, but for sum-based objectives such as k-median and k-means. Here instead of the maximum distance between a point and its center, the (squared) distances are added up for all points. For a set of points P from a metric space and a number k, the k-median problem is to find a clustering and centers such that the sum of the distances of every point to its closest center is minimized. For k-means clustering, the distances are squared, the metric is usually Euclidean, and the centers are allowed to come from all of $\mathbb{R}^d$. Now for k-median/k-means clustering with lower bounds, the situation differs in two aspects. We are given an additional parameter B and solutions now satisfy the additional constraint that every cluster has at least B points[1]. To achieve this, points are no longer necessarily assigned to their closest center but the solution now involves an assignment function of points to centers. The objective then is to minimize the (squared) sum of distances from every point to its assigned center. To the best of the authors' knowledge, k-median and k-means with lower bounds have not been studied, but for k-median, an $O(1)$-approximation follows from known work (see below).

For the related (also sum-based) facility location problem, finding solutions with lower bounds on the cluster sizes appeared in very different contexts. Given sets P and F from a finite metric space and *opening costs* for the points in F, the *facility location problem* asks to partition P into clusters and to assign a center from F to each cluster such that the sum of the distances of every point to its cluster plus the sum of the opening costs of open centers is minimized. For facility location with lower bounds, an additional parameter B is given and every cluster has to have at least B points. Karger and Minkoff [20] as well as Guha, Meyerson and Munagala [13] use relaxed versions of facility location with (uniform) lower bounds as subroutines for solving network design problems. This inspired the seminal work of Svitkina [26], who gives a constant-factor approximation algorithm for the facility location problem with (uniform) lower bounds. Ahmadian and Swamy [5] improve the approximation ratio to 82.6. Ahmadian and Swamy [6] state that the algorithm by Svitkina can be adapted for k-median by adequately replacing the first reduction step at the cost of an increase in the approximation factor.

It is often the case that restricting the number of clusters to k instead of having facility costs makes the design of approximation algorithms much more cumbersome, in particular when constraints are involved. For example, the related problem of finding a facility location

[1] In the introduction, we stick to uniform lower bounds since this is what we want for anonymity. In the technical part, we also discuss non-uniform lower bounds.

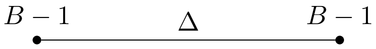


Figure 1 On the difference between lower-bounded clustering and weakly lower-bounded clustering.

solution where every cluster has to satisfy an *upper* bound, usually referred to as *capacitated facility location*, can be 3-approximated (see Aggarwal et al [2]), but finding a constant-factor approximation for capacitated k-median clustering is a long standing open problem [1, 16].

We demonstrate that the situation for *lower* bounds is different. By a relatively straightforward approach that we borrow from the area of approximation algorithms for hierarchical clustering, we show that approximation algorithms for facility location with lower bounds can be converted into approximation algorithms for k-median with lower bounds (at the cost of an increase in the approximation ratio), and this reduction works also for more general k-clustering problems including k-means. This leaves us with two challenges:

1. The resulting approximation algorithm has a very high approximation ratio.
2. For k-means clustering with lower bounds, no bicriteria or true approximation algorithm is known, and the results for standard facility location with lower bounds do not extend to the case for squared Euclidean distances: Both known algorithms for facility location use the triangle inequality an uncontrolled number of times to bound the cost of multiple reassignment steps. Thus the relaxed triangle inequality is not sufficient, as the resulting bound would depend on this number. Also the bicriteria algorithms by Karger and Minkoff [20] and Guha, Meyerson and Munagala [13] require repeated application of the triangle inequality. Thus, k-means with lower bounds needs a new technique.

To tackle these challenges, we define a new variation of lower-bounded clustering that we call *weakly lower-bounded* k-clustering. Here we allow points to be *allocated multiple times*. However a point may not be assigned more than once to the same center. This means that our 'clustering' is not a partitioning into subsets, but consists of not necessarily disjoint clusters (whose union is P). Each cluster has to respect the lower bound. To explain this idea, consider Figure 1. There are two locations with $B-1$ points each, and the distance between the two locations is Δ. For clustering with a lower bound of B, we can only open one center, which results in a clustering cost of $(B-1)\Delta$ for k-median (and $\Omega(B\Delta^2)$ for k-means). For clustering with weak lower bound B, we allow to assign points multiple times (but only to different centers). For each allocation, we pay the connection cost. In Figure 1, this allows us to open two centers while assigning one point from every location to the other location. This costs 2Δ for k-median (and $\Omega(\Delta^2)$ for k-means) for the two extra assignments. So even though we pay for more connections, the overall cost is smaller. This means that clustering with weak lower bounds can have an arbitrarily smaller cost than clustering with lower bounds, and in a way, this is a benefit: it means that we potentially pay less for having the lower bounds satisfied. Of course it also means that the gap between the optimal costs of the two problem variants with (standard) lower bounds and weak lower bounds is unbounded. We obtain the following results.

- We design an 8-approximation algorithm for weakly lower-bounded k-median and an $O(1)$-approximation algorithm for weakly lower-bounded k-means. The algorithms are conceptually simpler than their counterparts for lower-bounded facility location.
- Then we show that we can adapt the solutions such that every point is assigned to *only two centers* at the cost of a constant factor increase in the approximation ratio. We say that a solution has b-weak lower bounds if every point is assigned to at most b centers, so our results satisfy 2-weak lower bounds.

- Furthermore, we show that for $\epsilon \in (0,1)$ we can also get $O(1/\epsilon)$-approximate solutions that satisfy $(1+\epsilon)$-weak lower bounds if we allow fractional assignments of points.
- Finally, we show that our result on 2-weak lower bounds also implies a $(O(1), O(1))$-bicriteria approximation result for lower bounds, where the lower bounds are satisfied only to an extent of $B/O(1)$. Applying this result to squared Euclidean distances yields a bicriteria approximation for k-means with lower bounds, which is the first to the best of the authors' knowledge.
- Our results also extend to non-uniform lower bounds.

Recall our anonymization goal. When using weakly lower-bounded clustering, we still get the number of clusters that we desire and we also fully satisfy the anonymity requirement. We achieve this by distorting the data slightly by allowing data points to influence two clusters. In the fractional case, we get a solution where every data point is assigned to one main cluster and then contributes an ϵ-connection to a different cluster. By this small disturbance of the data set, we can meet the anonymity lower bound requirement for all clusters.

Techniques. The proof that k-clustering can be reduced to facility location builds upon a known nesting technique from the area of approximation algorithms for hierarchical clustering and is relatively straightforward. Our conceptional contribution is the definition of weakly lower-bounded clustering as a means to achieve anonymity. To obtain constant-factor approximations for weakly lower-bounded clustering, the idea is to incorporate an estimate for the cost of establishing lower bounds via facility costs, approximate a k-clustering problem with facility costs and then enforce lower bounds on a solution by connecting the closest $B - \ell$ points to a center which previously only had ℓ points. Similar ideas are present in the literature, which we adapt to our new problem formulation.

The main technical contribution in our paper is the proof that a solution assigning points to arbitrarily many centers can be converted into a solution where every point is assigned at most twice (or $(1+\epsilon)$-times, respectively), not only for k-median, but also for k-means. The latter means that the proof cannot use subsequent reassignment steps as it is the case in previous algorithms but has to carefully ensure that points are only reassigned once. We can also bypass this problem in the construction of a bicriteria algorithm. Previous bicriteria algorithms for lower bounds do not extend to k-means due to using multiple reassignments.

Related work. Approximation algorithms for clustering have been studied for decades. The unconstrained k-center problem can be 2-approximated [11, 14] and this is tight under P $\neq$ NP [15]. The (k,r)-center problem we discussed above is introduced and 2-approximated in [3]. We also call this problem k-center with lower bounds. McCutchen and Khuller [25] study k-center with lower bounds in a streaming setting and provide a $(6+\epsilon)$-approximation. One can also consider *non-uniform* lower bounds, i.e., every center has an individual lower bound that has to be satisfied if the center is opened. This variant is studied by Ahmadian and Swamy in [6] and they give a 3-approximation (for the slightly more general *k-supplier* problem with non-uniform lower bounds).

The facility location problem has a rich history of approximation algorithms and the currently best algorithm due to Li [22], achieving an approximation ratio of 1.488, is very close to the best known lower bound of 1.463 [12]. Bicriteria approximation algorithms for facility location with lower bounds are developed by Karger and Minkoff [20] and Guha, Meyerson and Munagala [13]. Svitkina [26] gives the first $O(1)$-approximation algorithm. The core of the algorithm is a reduction to facility location with capacities, embedded in a long chain of pre- and postprocessing steps. Ahmadian and Swamy [5] improve the approximation guarantee to 82.6. For the case of non-uniform lower bounds, Li [23] gives an $O(1)$-approximation algorithm. Although we did not discuss this in the introduction because it is less relevant to the anonymity motivation, this result also implies an $O(1)$-approximation for k-median with non-uniform lower bounds, as we show in the full version of this paper [7, Appendix A].

The k-median and k-means problems are APX-hard with the best known lower bounds being $1+2/e$ [18] and 1.0013 [8, 21]. The k-median problem can be $(2.675+\epsilon)$-approximated [9] and the best known approximation ratio for the k-means problem is $6.357+\epsilon$ [4]. To the best of the authors' knowledge, k-median and k-means with lower bounds have not been studied before. For the k-median problem, $O(1)$-approximations follow relatively easy from the work on facility location as outlined in the full version [7, Appendix A] and there is a possible adaptation of the algorithm by Svitkina as mentioned above. The authors are neither aware of an approximation algorithm or bicriteria algorithm for facility location with lower bounds that works for squared metrics, nor of one for k-means with lower bounds. We propose a bicriteria result that is applicable to k-means.

Finding a polynomial constant-factor approximation algorithm for the k-median problem with *upper bounds*, i.e., with capacities, is a long standing open problem. Recently, efforts have been made to obtain FPT approximation algorithms for the problem [1, 10].

2 Preliminaries

A k-clustering problem gets a finite set of input points P, a possibly infinite set of possible centers F, and a number $k \in \mathbb{N}$ and asks for a set of centers $C \subset F$ with $|C| \leq k$ and a mapping $a : P \to C$ such that

$$\operatorname{cost}(P, C, a) = \operatorname{cost}(C, a) = \sum_{x \in P} d(x, a(x))$$

is minimized, where $d : (P \cup F) \times (P \cup F) \to \mathbb{R}^+$ is a distance function that is symmetric and satisfies that $d(x, y) = 0$ iff $x = y$. For the *generalized k-median problem*, the distance d satisfies the α-relaxed triangle inequality, i.e., for all $x, y, z \in P \cup F$, it holds that $d(x, y) \leq \alpha d(x, z) + \alpha d(y, z)$.

We define the *k-median problem* as a generalized k-median problem with $P = F$ (finite) and $\alpha = 1$, and the *k-means problem* by setting $F = \mathbb{R}^d$ and $P \subset F$, and choosing d as the squared Euclidean distance, for which $\alpha = 2$. For these two problems, choosing the mapping $a : P \to C$ is always optimally done by assigning every point to (one of) its closest center(s). A *generalized facility location problem* has the same input as a generalized k-median problem except that it gets facility costs $f : F \to \mathbb{R}$ instead of a number k. The goal is to find a set of centers $C \subset F$ without cardinality constraint that minimizes $\sum_{x \in P} d(x, a(x)) + \sum_{c \in C} f(c)$. We use the term facility location not only if d is a metric but also in the case of a distance function satisfying the α-relaxed triangle inequality, analogously to the generalized k-median problem defined above.

We study generalized k-median and generalized facility location problems under side constraints which means that the choice of the mapping a is restricted. The side constraints that we study are versions of lower bounded clustering, i.e., they demand that every center gets a minimum number of points that are assigned to it. For clustering with (uniform) lower bounds, the input contains a number B and every cluster in the solution has to have at least B points. Non-uniform lower bounds are meaningful in the case of a finite set F and then, non-uniform lower bounds are given via a function $B : F \to \mathbb{N}$. If any points are assigned to a center $c \in F$ in a feasible solution, then it has to be at least $B(c)$ points.

When adding constraints, there is a subtle detail in the definition of generalized k-median problems for the case $P = F$: The question whether the center of a cluster has to be part of the cluster. Notice that without constraints, this makes no difference because assigning a center to a different center than itself cannot be beneficial. When we add lower bounds, this can change. We assume that choosing a center outside of the cluster is allowed and specifically say when the solution is such that centers are members of their clusters.

Our new problem variant called *weakly lower-bounded generalized k-median* is defined as follows. Given an instance of the same form as for the unconstrained generalized k-median problem plus lower bounds $B : F \to \mathbb{N}$, the goal is to compute a set of at most k centers $C \subset F$ and an assignment $a \colon P \to \mathcal{P}(C)$ such that the lower bound is satisfied, i.e., $|\{x \in P \mid c \in a(x)\}| \geq B(c)$ for all $c \in C$ and every point is assigned at least once. If a point is assigned multiple times the distance of the point to all assigned centers is paid by the solution. The total cost of a solution is given by

$$\operatorname{cost}(C, a) = \sum_{x \in P} \sum_{c \in a(x)} d(x, c).$$

If a solution of a weakly lower-bounded clustering problem satisfies that every point is assigned to at most b centers, then we say that the solution satisfies b-weak lower bounds.

3 Reducing Lower-Bounded k-Clustering to Facility Location

In this section, we observe that by using a known technique from the area of approximation algorithms for hierarchical clustering, we can turn an approximation algorithm for generalized facility location with lower bounds into an algorithm for generalized k-median with lower bounds. The technique is called *nesting*. Given two solutions S_1 and S_2 for the same generalized facility location problem with different number of centers $k_1 > k_2$, nesting describes how to find a solution S with k_2 centers which has a cost bounded by a constant times the costs of S_1 and S_2 and which is *hierarchically compatible* with S_1, i.e., the clusters in S result from merging clusters in S_1. We use this by computing a solution S_1 with an approximation algorithm for generalized facility location satisfying the lower bounds and a solution S_2 for unconstrained generalized k-median and then combining them via a nesting step. The resulting solution S has at most k centers and the clusters result from clusters that satisfy the lower bound – thus they satisfy the lower bound as well. For uniform lower bounds, the execution of this plan is very straightforward, for non-uniform lower bounds we have to be a bit more careful and adjust the nesting appropriately. Since most of this section follows relatively straightforwardly from known work, we defer the details to the full version [7, Appendix A]. Although the reduction is applicable to generalized k-median, this only helps to obtain constant-factor approximations for k-median because no approximation algorithms for generalized facility location with lower bounds are known for $\alpha > 1$. We get the following statement from combining the (adjusted) nesting results from Lin et al. [24] and the approximation algorithms for facility location with uniform lower bounds by Ahmadian and Swamy [5] and non-uniform lower bounds by Li [23].

▶ **Corollary 1.** *There exist polynomial-time $O(1)$-approximation algorithms for the k-median problem with uniform and non-uniform lower bounds.*

As a final note we observe that the crucial property of lower bound constraints we use here is *mergeability*: If a uniform lower bound is satisfied for a solution, then merging clusters results in a solution that is still feasible. This is in stark contrast to for example capacitated clustering. Our reduction works for mergeable constraints in general.

4 Generalized k-Median with Weak Lower Bounds

Now we consider a relaxed version of generalized k-median with lower bounds where points in P can be assigned multiple times. This relaxation does make sense since we have lower bounds on the centers, so it can be more valuable to assign points to multiple centers to satisfy the lower bounds instead of closing the respective centers. To see this we refer to Figure 1. We call this problem *generalized k-median with weak lower bounds.*

For ease of presentation, it is sensible to assume that F is finite. We observe that we can always set $F = P$ at a constant increase in the cost function if we are given a uniform lower bound. In particular, we assume in this section that $F = P$ holds for k-means.

▶ **Lemma 2.** *Let P be a point set and F be a possibly infinite set of centers. Let $a : P \to F$ be a mapping and define $a'(x) = \arg\min_{y\in P} d(y, a(x))$. Then it holds that*

$$\sum_{x\in P} d(x, a'(x)) \leq 2\alpha \cdot \sum_{x\in P} d(x, a(x)).$$

Proof. The lemma follows from the relaxed triangle inequality:

$$\sum_{x\in P} d(x, a'(x)) \leq \alpha \sum_{x\in P} \big(d(x, a(x)) + d(a(x), a'(x))\big) \leq 2\alpha \cdot \sum_{x\in P} d(x, a(x)). \qquad ◀$$

To achieve anonymity it is enough to have a uniform lower bound. However if we assume $F = P$ from the beginning, then our results also hold for *non-uniform lower bounds*, so we consider this more general case in this section.

For standard k-median/k-means with weak lower bounds we give an 8-approximate algorithm and an $O(1)$-approximate algorithm respectively. Furthermore we show that a solution to generalized k-median with weak lower bounds can be transformed into a solution to generalized k-median with *2-weak lower bounds* in polynomial time. We show that this transformation increases the cost only by a factor of $\alpha(\alpha + 1)$. We combine this with the approximation algorithm for standard k-median/k-means with weak lower bounds and obtain an approximation algorithm for standard k-median/k-means with 2-weak lower bounds. If we allow fractional assignments we show how to obtain a solution which assigns every point by an amount of at most $1 + \epsilon$ for arbitrary $\epsilon \in (0, 1)$, losing $\lceil \frac{1}{\epsilon} \rceil \alpha(\alpha + 1) + 1$ in the approximation factor.

Computing a solution. To approximate generalized k-median with weak non-uniform lower bounds, we reduce this problem to generalized k-median with center costs. In this variant of generalized k-median, the input contains both a number k *and* center opening costs $f : F \to \mathbb{R}^+$. The objective is then

$$\mathrm{cost}^f(C, a) = \sum_{x\in P} d(x, a(x)) + \sum_{c\in C} f(c)$$

while the solution space is constrained to center sets of size at most k as for generalized k-median. The reduction that we use works by introducing a center cost of

$$f(c) = \sum_{p\in D_c} d(p, c) \tag{1}$$

for every point $c \in F$. This cost is paid if c becomes a center. Here D_c is the set consisting of the $B(c)$ nearest points in P to c. The idea for this reduction is adapted from the bicriteria algorithm for lower-bounded facility location presented by Guha, Meyerson and Munagala [13] and Karger, Minkoff [20].

Note that for a center c in a feasible solution (C, a) to generalized k-median with weak lower bounds, the term $\sum_{p \in D_c} d(p, c)$ is a lower bound on the assignment cost caused by c. This leads to the following lemma.

▶ **Lemma 3.** *Let OPT' be an optimal solution to the generalized k-median problem with center costs as defined in* (1) *and $OPT = (O, h)$ be an optimal solution to generalized k-median with weak lower bounds. It holds that $\mathrm{cost}^f(OPT') \leq 2\,\mathrm{cost}(OPT)$.*

Proof. For $p \in P$ let $c_p = \mathrm{argmin}\{d(p, c) \mid c \in h(p)\}$ be the closest center to which p is assigned in OPT. We define $h'(p) = c_p$ for all $p \in P$ and obtain a feasible solution (O, h') to the generalized k-median problem with center cost. Furthermore we have

$$\begin{aligned}\mathrm{cost}^f(OPT') \leq \mathrm{cost}^f(O, h') &= \sum_{c \in O} f(c) + \sum_{p \in P} d(p, h'(p)) \\ &= \sum_{c \in O} \sum_{p \in D_c} d(p, c) + \sum_{p \in P} d(p, h'(p)) \\ &\leq 2 \sum_{p \in P} \sum_{c \in h(p)} d(p, c) \\ &= 2\,\mathrm{cost}(OPT).\end{aligned}$$

The second inequality follows from the fact that $\sum_{c \in O} \sum_{p \in D_c} d(p, c)$ and $\sum_{p \in P} d(p, h'(p))$ are both lower bounds on the assignment cost of OPT. ◀

Let (C, a) be a solution for the generalized k-median problem with center costs. To turn it into a solution for generalized k-median with weak lower bounds we have to modify the assignment. Let $c \in C$ and $n_c = |a^{-1}(c)|$. We additionally assign $m_c = \max\{0, B(c) - n_c\}$ points to c to satisfy the lower bound. Let $S_c \subset D_c$ be the set of points in D_c which are not assigned to c. We choose m_c points from S_c and assign them to c. This is feasible since we are allowed to assign points multiple times. Let (C, a') be the corresponding solution.

▶ **Lemma 4.** *It holds that $\mathrm{cost}(C, a') \leq \mathrm{cost}^f(C, a)$.*

Proof. The additional assignment cost for each center $c \in C$ can be upper bounded by $\sum_{p \in D_c} d(p, c)$. We obtain

$$\begin{aligned}\mathrm{cost}(C, a') \leq& \sum_{c \in C} \sum_{p \in D_c} d(p, c) + \sum_{p \in P} d(p, a(p)) \\ =& \,\mathrm{cost}^f(C, a).\end{aligned}$$ ◀

Lemma 3 and Lemma 4 imply the following corollary.

▶ **Corollary 5.** *Given a γ-approximation for the generalized k-median problem with center costs, we get a 2γ-approximation for the generalized k-median problem with weak lower bounds in polynomial time.*

For k-median, we combine Corollary 5 with Corollary 5.5 from [27] which shows that an algorithm by Jain et al. [17] can be used to obtain a 4-approximation for the k-median problem with center costs. This gives an 8-approximation for k-median with weak lower bounds. For k-means, we use the algorithm by Jain and Vazirani [19] which was originally designed for k-median. However, as outlined in the journal version [19], it can be used for k-means when $F = P$, and also for k-median with center costs. The two extensions are not conflicting and can both be applied to obtain an $O(1)$-approximation for k-means with center costs for the case $F = P$.

4.1 Reducing the Number of Assignments per Client

We see that the solution for standard k-median/k-means with weak lower bounds computed above can assign a point to all centers in the worst case. The number of assigned centers per point cannot be bounded by a constant. This may not be desirable in the context of publishing anonymized representatives since the distortion of the original data set is not bounded.

However, we show that any solution to the generalized k-median problem with weak lower bounds can be transformed into a solution assigning every point at most twice. This increases the cost by a factor of $\alpha(\alpha+1)$. Recall that α is the constant appearing in the relaxed triangle inequality. This leads to the following theorem.

▶ **Theorem 6.** *Given a solution (C,a) to generalized k-median with weak lower bounds, we can compute a solution $(\widetilde{C},\widetilde{a})$ to generalized k-median with 2-weak lower bounds (assigning every point at most twice) in polynomial time such that* $\operatorname{cost}(\widetilde{C},\widetilde{a}) \leq \alpha(\alpha+1)\operatorname{cost}(C,a)$.

Reassignment process. We start by setting $\widetilde{C}=C$ and $\widetilde{a}=a$ and modify both $\widetilde{C}$ and $\widetilde{a}$ until we obtain a feasible solution to generalized k-median with 2-weak lower bounds. During the process, the centers in $\widetilde{C}$ are called *currently open*, and when a center is deleted from $\widetilde{C}$, we say it is *closed*. The centers are processed in an arbitrary but fixed order, i.e., we assume that $C=\{c_1,\dots,c_{k'}\}$ for some $k' \leq k$ and process them in order $c_1,\dots,c_{k'}$. We say that c_i is *smaller* than c_j if $i<j$.

Let $c=c_i$ be the currently processed center. By P_c, we denote the set of points assigned to c under $\widetilde{a}$. We divide P_c into three sets $P_c^1=\{q\in P_c \mid |\widetilde{a}(q)|=1\}$, $P_c^2=\{q\in P_c \mid |\widetilde{a}(q)|=2\}$ and $P_c^3=\{q\in P_c \mid |\widetilde{a}(q)|\geq 3\}$. Furthermore with $C(P_c^3)$ we denote all centers which are connected to at least one point in P_c^3 under $\widetilde{a}$.

If P_c^3 is empty, we are done and proceed with the next center in $\widetilde{C}$. Otherwise we need to empty P_c^3. Observe that points in P_c^3 are assigned to multiple centers, so if we delete the connection between one of these points and c, the point is still served by some other center. However, doing so may violate the lower bound at c. So we have to replace this connection.

As long as P_c^3 is non-empty, we do the following. We pick a center $d=\min C(P_c^3)\backslash\{c\}$ and a point $x\in P_c^3$ connected to d. We want to assign a point y from P_d^1 to c to free x. For technical reasons, we restrict the choice of y: We exclude all points from the subset $\overline{P_d^1} := \{q\in P_d^1 \mid |a(q)|\geq 3 \text{ and } a(q)\cap\{c_1,\dots,c_{i-1}\}\cap\widetilde{C}\neq\emptyset\}$, i.e., all points which were assigned to at least 3 centers under the initial assignment a, and where one of these at least 3 centers is still open *and* smaller than c.

If $P_d^1\backslash\overline{P_d^1}$ is non-empty, we pick a point $y\in P_d^1\backslash\overline{P_d^1}$ arbitrarily. We set $\widetilde{a}(y)=\{d,c\}$ and $\widetilde{a}(x)=\widetilde{a}(x)\backslash\{c\}$. So x is no longer connected to c, but to satisfy the lower bound at c we replace x by y (Figure 2). If $P_d^1\backslash\overline{P_d^1}$ is empty, our replacement plan does not work. Instead, we close d. This means that x is now assigned to one center less, and, if this happens repeatedly, x will at some point no longer be in P_c^3. Since we close d, all points in P_d^1 have

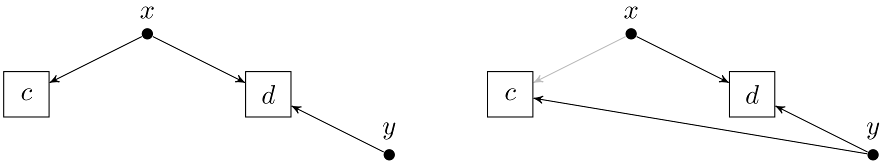


Figure 2 Connection between $x\in P_c^3$ and c is deleted. A point $y\in P_d^1$ replaces x.

Algorithm 1 Reducing the number of assigned centers per point to two.

```
1  define an ordering on the centers $c_1 \le c_2 \ldots \le c_{k'}$
2  set $\widetilde{C} := C$ and $\widetilde{a} := a$
3  for all $c \in C$
4      $P_c := \{q \in P \mid c \in \widetilde{a}(q)\}$
5      $P_c^3 := \{q \in P_c \mid |\widetilde{a}(q)| \ge 3\}$,  $P_c^i := \{q \in P_c \mid |\widetilde{a}(q)| = i\}$ for $i = 1, 2$
6      $C(P_c^3) := \bigcup_{q \in P_c^3} \widetilde{a}(q)$
7  for $i = 1$ to $l$ do
8      while $P_{c_i}^3 \neq \emptyset$ do
9          $d = \min C(P_{c_i}^3) \backslash \{c_i\}$
10         $\overline{P_d^1} = \{q \in P_d^1 \mid |a(q)| \ge 3$ and $a(q) \cap \{c_1, \ldots, c_{i-1}\} \cap \widetilde{C} \neq \emptyset\}\}$
11         if $P_d^1 \backslash \overline{P_d^1} = \emptyset$ then
12             for all $q \in P_d^1$
13                 let $e = \min(a(q) \cap \widetilde{C})$
14                 set $\widetilde{a}(q) = \{e\}$
15             delete $d$ from $\widetilde{C}$ and all connections to $d$ in $\widetilde{a}$
16         else
17             pick $x \in P_{c_i}^3$ connected to $d$ and $y \in P_d^1 \backslash \overline{P_d^1}$
18             set $\widetilde{a}(x) = \widetilde{a}(x) \backslash \{c_i\}$, $\widetilde{a}(y) = \{c_i, d\}$
```

to be reassigned because they are only connected to d. For each $q \in P_d^1$, we reassign q to the smallest currently open center in $a(q)$. Notice that such a center exists and is smaller than c because $P_d^1 = \overline{P_d^1}$ and for every $q \in \overline{P_d^1}$, there is at least one center in $a(q) \cap \widetilde{C}$ which is smaller than c.

The entire process is described in Algorithm 1. It satisfies the following invariants.

▶ **Lemma 7.** *Algorithm 1 computes a feasible solution $(\widetilde{C}, \widetilde{a})$ to generalized k-median with 2-weak lower bounds. Furthermore the following properties hold during all steps of the algorithm.*

1. *The algorithm never establishes connections for points currently assigned more than once.*
2. *For any center $c \in C$, P_c does not change before c is processed or closed.*
3. *If a connection between $x \in P$ and the currently processed center $c \in \widetilde{C}$ is deleted by the algorithm, we have from this time on $x \notin P_c^3$ until termination. Moreover P_c^3 remains empty after c is processed.*
4. *While the algorithm processes $c \in C$ we always have $c < \min C(P_c^3) \backslash \{c\}$. Moreover all currently open centers which are smaller than c remain open until termination.*
5. *If the algorithm establishes a new connection in Line 14 or Line 18 it remains until termination.*

Proof. The process terminates: For every iteration of the while loop starting in Line 8, either a point is deleted from $P_{c_i}^3$ or there is at least one point $x \in P_{c_i}^3$ for which $|\widetilde{a}(x)|$ is reduced by one. Furthermore $|\widetilde{a}(x)|$ does never increase for any $x \in P_{c_i}^3$.

The final solution satisfies lower bounds: Every time we delete a connection between a point and a center it either happens because the center is closed or we replace this connection by assigning a new point to it. So the lower bounds are satisfied at all open centers.

All points stay connected to a center: Assume that the algorithm deletes the connection between a point p and the center d it is exclusively assigned to. This only happens if at this time d is closed by the algorithm. Then p is assigned to another center as defined in Line 14.

We conclude that the solution is feasible.

Property 1: The algorithm establishes connections in Line 14 and Line 18 which always involve a point currently assigned once.

Property 2: Let $c \in C$. Connections are only changed for the center that is currently processed or for a smaller center which has been processed already. Thus, the algorithm does not add or delete any connections involving c before c is processed or closed.

Property 3: Assume that after the connection between $x \in P_c^3$ and c is deleted by the algorithm, x is again part of P_c^3. That would require that the algorithm establishes a new connection for a point which is connected more than once, which does not happen by Property 1. For the same reason P_c^3 remains empty after c is processed by the algorithm.

Property 4: Assume c is currently processed by the algorithm and $d = \min C(P_c^3)\backslash\{c\}$. We know that at this time P_d^3 is non-empty, which is by Property 3 only possible if d is processed after c. Thus we have $c < d$. This also means that centers can only be closed by the algorithm if they are not processed so far.

Property 5: If a connection is deleted, the respective point is either connected to more than two centers or to a center which is closed at this time. A connection in Line 14 or Line 18 is established by the algorithm between a point which is at this time assigned exactly once and a center which is already processed or currently processed by the algorithm. Thus the point is from this time on never assigned to more than two centers and the center remains open until termination by Property 4. So the necessary conditions for a deletion of this connection are never fulfilled. ◀

We now want to bound the cost of new connections created by the algorithm by the cost of the original solution. Notice that only Line 18 generates new connections, Line 14 re-establishes connections that were originally present. So let N_c be the set of all points newly assigned to c by the algorithm in Line 18 while center c is processed. For $y \in N_c$ let d_y be the respective center in Line 9 of Algorithm 1 and x_y the point in Line 17 contained in P_c^3 and connected to d_y. Using the α-relaxed triangle inequality, we obtain the following upper bound.

$$\begin{aligned} d(y,c) &\le \alpha(d(y,x_y)+d(x_y,c)) \le \alpha\Big(\alpha\big(d(y,d_y)+d(d_y,x_y)\big)+d(x_y,c)\Big) \\ &= \alpha^2\big(d(y,d_y)+d(d_y,x_y)\big)+\alpha d(x_y,c). \end{aligned} \tag{2}$$

We can apply (2) to all $c \in \widetilde{C}$ and all $y \in N_c$. This yields the following upper bound on the cost of the final solution $(\widetilde{C},\widetilde{a})$.

$$\begin{aligned} \operatorname{cost}(\widetilde{C},\widetilde{a}) &= \sum_{c\in\widetilde{C}} \sum_{\substack{y\in P:\\ c\in\widetilde{a}(y)}} d(y,c) = \sum_{c\in\widetilde{C}} \Big(\sum_{y\in P_c\backslash N_c} d(y,c) + \sum_{y\in N_c} d(y,c) \Big) \\ &\le \sum_{c\in\widetilde{C}} \Big(\sum_{y\in P_c\backslash N_c} d(y,c) + \sum_{y\in N_c} \alpha^2(d(y,d_y)+d(d_y,x_y)) + \alpha d(x_y,c) \Big). \end{aligned} \tag{3}$$

Expression (3) is what we want to pay for. We show in Observation 8 below that all involved distances contribute to the original cost as well. So in principle, we can bound each summand by a term in the original cost. But what we need to do is to bound the number of times that each term in the original cost gets charged. To organize the counting, we count how many times a specific tuple of a point z and a center f occurs as $d(z,f)$ in (3). Since it is important at which position a tuple appears, we give names to the different occurrences (also see Figure 3).

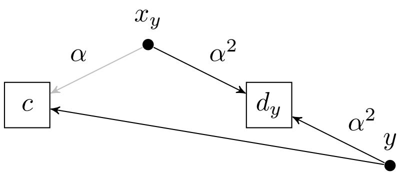


Figure 3 Bounding the distance between y and c. The respective distances appear with a factor of α or α^2. Tuple (x_y, c) is of Type 1 and $(x_y, d_y), (y, d_y)$ are of Type 2.

We say that that a tuple appears as a tuple of Type 0 if it appears as $d(y, c)$ in (3), as tuple of Type 1 if it appears as $d(x_y, c)$, and as tuple of Type 2 if it appears as $d(y, d_y)$ or $d(d_y, x_y)$. We distinguish the latter type further by calling a tuple occurring as $d(y, d_y)$ a tuple of Type 2.1 and a tuple occurring as $d(x_y, d_y)$ a tuple of Type 2.2. We say that $(y, d_y), (d_y, x_y)$ and (x_y, c) *contribute* to the cost of (y, c), where by the *cost* of (y, c) we mean the upper bound on $d(y, c)$ in (2) which we want to pay for.

▶ **Observation 8.** *If a tuple (z, f), $z \in P, f \in C$, occurs as Type 0, 1 or 2, then $f \in a(z)$, so in particular, $d(z, f)$ occurs as a term in the cost of the original solution.*

Proof. For a center c the set $P_c \backslash N_c$ consists of points which are assigned to c by the initial assignment a or assigned to c while c is not processed by the algorithm. The latter can only happen if a connection is reestablished in Line 14 which requires that the connection was already present in (C, a). So Type 0 tuples satisfy the statement.

For Type 1 and 2 tuples, consider $y \in N_c$ for some center c and the respective tuples $(x_y, c), (y, d_y), (x_y, d_y)$. Notice that both y and x_y are connected to d_y the step before y is assigned to c. By Property 4 of Lemma 7 we have $c < d_y$. Thus we know by Property 2 of Lemma 7 that P_{d_y} is not changed by the algorithm at least until y is assigned to c. So $d_y \in a(y)$ and $d_y \in a(x_y)$ which proves that Type 2 tuples satisfy the statement. Moreover it holds that $c \in a(x_y)$ since there is a time where $x_y \in P_c^3$. This can, by Property 1 of Lemma 7, only happen if the connection between x_y and c is already part of (C, a). Thus, Type 1 tuples satisfy the statement. ◀

As indicated above, a tuple (z, f) can contribute to the cost of multiple tuples. Notice that a tuple occurs at most once as a tuple of Type 0 in (3). To bound the cost of $(\widetilde{C}, \widetilde{a})$ we bound the number of times a tuple appears as Type 1 or Type 2 tuple in (3).

▶ **Lemma 9.** *For all $z \in P, f \in C$, the tuple (z, f) can appear in* (3) *at most once as a tuple of Type 1 and at most once as a tuple of Type 2.*

Proof. In the following, the tuple whose cost the tuple (z, f) contributes to will always be named (y, c), and we denote the time at which y is newly assigned to c by t.

Type 1: Assume (z, f) contributes to the cost of (y, c) as a tuple of Type 1. Then $f = c$. Notice that at the time step before t we must have $z \in P_c^3$ and afterwards, z is never again contained in P_c^3 by Property 3 of Lemma 7. Thus the pair (z, c) can never again be responsible for any reassignment to c, i.e., $(z, c) = (z, f)$ does not contribute to any further cost as a tuple of Type 1.

Type 2.1: Assume that (z, f) contributes to the cost of (y, c) as a tuple of Type 2.1. Then $z = y$. At the time step before t, we have $y \in P_f^1$, $f \in C(P_c^3)$, and at time t, we have $y \in P_c^2 \cap P_f^2$. By Property 5 of Lemma 7, newly established connections are never deleted, so after time t, it always holds that $y \in P_c$. So even if y is in P_f at a later time, it cannot be in P_f^1 since it is also connected to c. So $(y, f) = (z, f)$ does not contribute to any further cost as tuple of Type 2.1. Furthermore by Property 1 of Lemma 7 we know that y is always assigned to fewer than three centers after t which means that (y, f) does not contribute as tuple of Type 2.2 to the cost of any connection established by the algorithm after t either.

Type 2.2: Finally we consider the case where (z, f) contributes to the cost of (y, c) as a tuple of Type 2.2. At time t, the algorithm processes c. By the way the algorithm chooses f and z, we know that $z \in P_c^3$ (at the beginning of the process, i.e., before t) and $f = \min C(P_c^3)\backslash\{c\}$. After t, Property 3 of Lemma 7 implies $z \notin P_c^3$, which means that as a tuple of Type 2.2, it can never again contribute to the cost of any tuple containing c. Assume instead that it contributes (as Type 2.2) to the cost of a tuple (y', c') for a center $c' \neq c$, and some point $y' \in P$. This is supposed to happen after t, so y' is newly assigned to c' at some time $t' > t$. Before c' is processed, we must always have $z \in P_{c'}^3$ by Property 1 and 2 of Lemma 7. So in particular, at time $t < t'$ we have $c' \in C(P_c^3)\backslash\{c\}$. Moreover we know that at some time while c' is processed by the algorithm we have $f = \min C(P_{c'}^3)\backslash\{c'\}$. Using Property 4 of Lemma 7 we conclude that $c' < f$. Which is a contradiction since the algorithm chose f and not c' at time t, i.e., $f = \min C(P_c^3)\backslash\{c\}$ must hold. Thus, (z, f) cannot contribute to the cost of (y', c') as a tuple of Type 2.2.

It is left to show that (z, f) cannot contribute to the cost of any (y', c') as a tuple of Type 2.1 at some time $t' > t$. For a contribution as Type 2.1, we would have $z = y'$ and $y' \in P_f^1$. We show that in this case y' is in fact contained in $\overline{P_f^1}$. Remember that at time t we have $y' = z \in P_c^3$ and that this only happens if $|a(y')| \geq 3$ by Property 1 of Lemma 7. Moreover c is still open by Property 4 of Lemma 7 and is smaller than c'. Thus $c \in a(y') \cap \{e \mid e < c'\} \cap \widetilde{C}$, which proves $y' \in \overline{P_f^1}$. Therefore the algorithm does not assign y' to c' (see Lines 11-15) and (z, f) does not contribute as tuple of Type 2.1 to the cost of any connection established by the algorithm after t. ◀

We now know that a tuple only appears at most once as any of the three tuple types. For the final counting, we define $T0$, $T1$ and $T2$ as the sets of all tuples of Type 0, 1 and 2, respectively. We could already prove a bound on the cost now, but to make it slightly smaller and prove Theorem 6, we need one final statement.

▶ **Lemma 10.** *The set $T0 \cap T1 \cap T2$ is empty.*

Proof. Let $(z, f) \in T0 \cap T1 \cap T2$. Since (z, f) is of Type 0, the point z must be connected to f in the final assignment $\widetilde{a}$. We distinguish whether the connection between z and f was deleted at some point by the algorithm or not. If it is not deleted, (z, f) cannot be of Type 1 since this would require that z is temporarily not assigned to f. Otherwise the connection between z and f was deleted while f was processed and later reestablished by the algorithm in Line 14.

By assumption the tuple is also of Type 2. Assume it is of Type 2.1 and contributes to the cost of a tuple (y, c) with $z = y$. We know that $c < f$ by Property 4 of Lemma 7. Consider the time when z is newly assigned to c. The step before we have $z \in P_f^1$. On the other hand while f is processed we have $z \in P_f^3$ in contradiction to Property 1 of Lemma 7.

Assume finally that (z, f) is of Type 2.2 and contributes to the cost of a tuple (y, c). Again we have $c < f$. Consider the time y is newly assigned to c. The step before we have $z \in P_c^3$ and, by Property 1 and 2 of Lemma 7, also $z \in P_f^3$. At the time the connection between z and f is reestablished by the algorithm, both centers are contained in $a(z) \cap \widetilde{C}$. This is a contradiction to $c < f = \min(a(z) \cap \widetilde{C})$. This completes the proof. ◀

Proof of Theorem 6. Slightly abusing the notation we write $d(e)$ for a tuple $e = (z, f)$ by which we mean the distance $d(z, f)$. Combining Lemma 9 and 10 we obtain

$$\begin{aligned}
\mathrm{cost}(\widetilde{C}, \widetilde{a}) &\leq \sum_{c \in \widetilde{C}} \Big(\sum_{y \in P_c \setminus N_c} d(y, c) + \sum_{y \in N_c} \alpha^2 (d(y, d_y) + d(d_y, x_y)) + \alpha d(x_y, c) \Big) && (3)\\
&= \sum_{e \in T0} d(e) + \alpha^2 \sum_{e \in T2} d(e) + \alpha \sum_{e \in T1} d(e) && (4)\\
&\leq (\alpha^2 + \alpha) \, \mathrm{cost}(C, a). && (5)
\end{aligned}$$

By Lemma 9 we know that a tuple only appears at most once as any of the three tuple types. We replace (3) by summing up the cost of all tuples in T_i for $i = 0, 1, 2$ with the respective factor for each type and obtain (4).

Finally by Observation 8 the cost $d(e)$ for $e \in T_0 \cup T_1 \cup T_2$ occurs as a term in the original solution and $T_0 \cap T_1 \cap T_2 = \emptyset$ by Lemma 10, which proves (5). ◀

So it is possible to reduce the number of assignments per point to two at a constant factor increase in the approximation factor. We can go even further and allow points to be fractionally assigned to centers which poses the question if it is possible to bound the assigned amount by a number smaller than two. Indeed we can prove for every $\epsilon \in (0, 1)$ that we can modify a solution to generalized k-median with weak lower bounds such that every point is assigned by an amount of at most $1 + \epsilon$ and the cost increases by a factor of $\mathcal{O}(\frac{1}{\epsilon}\alpha^2)$. Note that even if we allow fractional assignments of points to centers, the centers remain either open or closed, which differentiates our result from a truly fractional solution, where it is also allowed to open centers fractionally. Furthermore, the new assignment assigns every point to at most two centers. It is assigned by an amount of one to one center and potentially by an additional amount of ϵ to a second center.

Since we consider fractional assignments we modify our notation and denote with $\widetilde{a}_x^c \in [0, 1]$ the amount by which $x \in P$ is assigned to $c \in \widetilde{C}$, where $\widetilde{C}$ is the set of centers. Let $\widetilde{a}_x = \sum_{c \in \widetilde{C}} \widetilde{a}_x^c$ be the amount by which $x \in P$ is assigned to $\widetilde{C}$. The assignment $\widetilde{a}$ is feasible if $\widetilde{a}_x \geq 1$ for all $x \in P$ and $\sum_{x \in P} \widetilde{a}_x^c \geq B(c)$ for all $c \in \widetilde{C}$, and its cost is

$$\mathrm{cost}(\widetilde{C}, \widetilde{a}) = \sum_{c \in \widetilde{C}} \sum_{x \in P} \widetilde{a}_x^c d(x, c).$$

We omit the proof of the following theorem as it is similar to the proof of Theorem 6, but to satisfy lower bounds we can only assign an amount of ϵ from points which are already assigned once. Therefore we consider suitable sets with $\lceil \frac{1}{\epsilon} \rceil$ points, which leads to the increase of $\mathcal{O}(\frac{1}{\epsilon})$ in the approximation factor. For more details we refer to [7, Appendix C].

▶ **Theorem 11.** *Given $0 < \epsilon < 1$ and a solution (C, a) to generalized k-median with weak lower bounds, we can compute a solution $(\widetilde{C}, \widetilde{a})$ to generalized k-median with $(1 + \epsilon)$-weak lower bounds, i.e., $\widetilde{a}_x \leq 1 + \epsilon$ for all $x \in P$ in polynomial time such that* $\mathrm{cost}(\widetilde{C}, \widetilde{a}) \leq (\lceil \frac{1}{\epsilon} \rceil \alpha(\alpha + 1) + 1) \, \mathrm{cost}(C, a)$.

On pages 7-8 we reduce generalized k-median with weak lower bounds to generalized k-median with center cost and obtain an 8 or $O(1)$-approximation for k-median or k-means with weak lower bounds, respectively. We combine this with Theorem 6 to get a solution with 2-weak lower bounds whose cost is a constant factor away from the problem with weak lower bounds. Since weak lower bounds are a relaxation of 2-weak lower bounds, we get:

Algorithm 2 A $(\beta, \gamma \max\{\frac{\alpha\beta}{1-\beta}+1, \frac{\alpha^2\beta}{1-\beta}\})$-bicriteria approximation algorithm to generalized k-median with lower bounds.

```
Input  : $\gamma$-approximate solution $(C, a)$ to generalized $k$-median with 2-weak lower
         bounds, $C = \{c_1, \dots, c_{k'}\}$
Output : Bicriteria solution $(C', a')$ to generalized $k$-median with lower bounds.
set $C' = \emptyset$, $a'(x) = \perp$ for all $x \in P$
$N = P$
for $i = 1$ to $k'$ do
    $A_i = \{x \in P \mid c_i \in a(x)\}$
    $B_i = \{x \in A_i \mid a(x) \subset \{c_1 \dots, c_i\}\} \cap N$
    if $A_i \cap N \geq \beta B(c_i)$ then
        set $a'(x) = c_i$ for all $x \in A_i \cap N$
        $N = N \backslash A_i$
        $C' = C' \cup \{c_i\}$
    else
        set $a'(x) = \arg\min_{c \in C'} d(x, c)$ for all $x \in B_i$
```

▶ **Corollary 12.** *Let OPT be an optimal solution to k-median/k-means with 2-weak lower bounds. We can compute a solution (C, a) in polynomial time for*

1. *k-median with 2-weak lower bounds with* $\mathrm{cost}(C, a) \leq 16\,\mathrm{cost}(OPT)$
2. *k-means with 2-weak lower bounds with* $\mathrm{cost}(C, a) \leq O(1)\,\mathrm{cost}(OPT)$.

Combining the results from Section 4 with Theorem 11 we obtain:

▶ **Corollary 13.** *Let OPT be an optimal solution to k-median/k-means with $(1+\epsilon)$-weak lower bounds. We can compute a solution (C, a) in polynomial time for*

1. *k-median with $(1+\epsilon)$-weak lower bounds with* $\mathrm{cost}(C, a) \leq (16\lceil\frac{1}{\epsilon}\rceil + 8)\,\mathrm{cost}(OPT)$
2. *k-means with $(1+\epsilon)$-weak lower bounds with* $\mathrm{cost}(C, a) \leq O(\frac{1}{\epsilon})\,\mathrm{cost}(OPT)$.

4.2 A Bicriteria Algorithm to Generalized k-Median with Lower Bounds

A (β, δ)-bicriteria solution for generalized k-median with lower bounds consists of at most k centers $C' \subset F$ and an assignment $a' \colon P \to C$ such that at least $\beta B(c)$ points are assigned to $c \in C'$ by a' and $\mathrm{cost}(C', a') \leq \delta\,\mathrm{cost}(OPT)$. Here OPT denotes an optimal solution to generalized k-median with lower bounds.

Given a $\beta \geq \frac{1}{2}$ and a γ-approximate solution to generalized k-median with 2-weak lower bounds (C, a), we can compute a $(\beta, \gamma \max\{\frac{\alpha\beta}{1-\beta}+1, \frac{\alpha^2\beta}{1-\beta}\})$-bicriteria solution in the following way. Let $C = \{c_1, \dots, c_{k'}\}$ for some $k' \leq k$. We process the centers in order $c_1, \dots, c_{k'}$ and decide if they are open or closed. We say that c_i is *smaller* than c_j if $i < j$. If we decide that a center c is open we directly assign at least $\lceil \beta B(c) \rceil$ points to c. In the beginning all points are unassigned.

Consider center c_i. Let A_i be the set of all points assigned to c_i under a. We know that $|A_i| \geq B(c_i)$. If at least $\lceil \beta B(c_i) \rceil$ points in A_i are not assigned so far, c_i remains open and all currently unassigned points from A_i are assigned to c_i (Figure 4). If less than $\lceil \beta B(c_i) \rceil$ points from A_i are unassigned, the center is closed.

Let C' denote the centers from $\{c_1, \dots, c_{i-1}\}$ which are open and B_i the set of unassigned points from A_i which are not connected to any center larger than c_i under a. To guarantee that all points are assigned at the end, we have to care about points in B_i. By assumption there are at most $\lfloor \beta B(c_i) \rfloor$ such points. We simply assign a point $p \in B_i$ to the nearest center $\arg\min_{c \in C'} d(c, p)$ in C'.

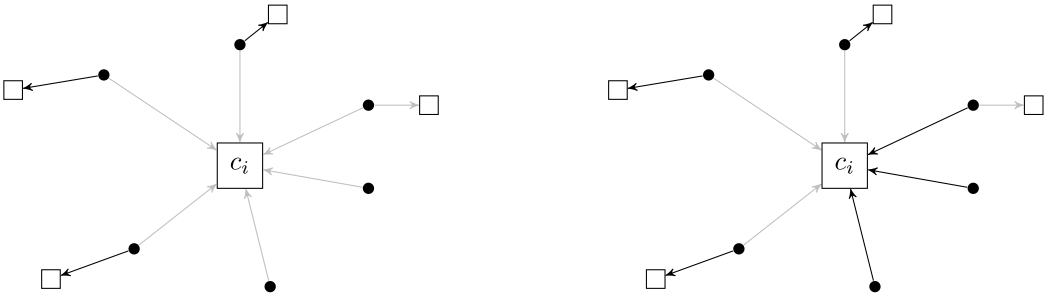


Figure 4 Shows the case where A_i contains at least $\lceil \beta B(c_i) \rceil$ unassigned points. The three points on the left are already assigned to other centers and the three points on the right are newly assigned to c_i. The gray connections come from a.

The whole procedure is described in Algorithm 2. For the proof of the claimed approximation factor we refer to [7, Appendix D].

▶ **Theorem 14.** *Given a γ-approximate solution (C, a) to generalized k-median with 2-weak lower bounds and a fixed $\beta \in [0.5, 1)$, Algorithm 2 computes a $(\beta, \gamma \max\{\frac{\alpha\beta}{1-\beta} + 1, \frac{\alpha^2\beta}{1-\beta}\})$-bicriteria solution to generalized k-median with lower bounds in polynomial time. In particular, there exists a polynomial-time $(\frac{1}{2}, O(1))$-bicriteria approximation algorithm for k-means with lower bounds.*

References

1 Marek Adamczyk, Jaroslaw Byrka, Jan Marcinkowski, Syed Mohammad Meesum, and Michal Wlodarczyk. Constant-factor FPT approximation for capacitated k-median. In *Proceedings of the 27th Annual European Symposium on Algorithms (ESA)*, pages 1:1–1:14, 2019. doi:10.4230/LIPIcs.ESA.2019.1.

2 Ankit Aggarwal, Anand Louis, Manisha Bansal, Naveen Garg, Neelima Gupta, Shubham Gupta, and Surabhi Jain. A 3-approximation algorithm for the facility location problem with uniform capacities. *Mathematical Programming*, 141(1-2):527–547, 2013. doi:10.1007/s10107-012-0565-4.

3 Gagan Aggarwal, Rina Panigrahy, Tomás Feder, Dilys Thomas, Krishnaram Kenthapadi, Samir Khuller, and An Zhu. Achieving anonymity via clustering. *ACM Transactions on Algorithms (TALG)*, 6(3):49:1–49:19, 2010. doi:10.1145/1798596.1798602.

4 Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k-means and Euclidean k-median by primal-dual algorithms. In *Proceedings of the 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 61–72, 2017. doi:10.1109/FOCS.2017.15.

5 Sara Ahmadian and Chaitanya Swamy. Improved approximation guarantees for lower-bounded facility location. In *Proceedings of the 10th International Workshop on Approximation and Online Algorithms (WAOA)*, pages 257–271, 2012. doi:10.1007/978-3-642-38016-7_21.

6 Sara Ahmadian and Chaitanya Swamy. Approximation algorithms for clustering problems with lower bounds and outliers. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming, (ICALP)*, pages 69:1–69:15, 2016. doi:10.4230/LIPIcs.ICALP.2016.69.

7 Anna Arutyunova and Melanie Schmidt. Achieving anonymity via weak lower bound constraints for k-median and k-means. arXiv:2009.03078v2.

8 Pranjal Awasthi, Moses Charikar, Ravishankar Krishnaswamy, and Ali Kemal Sinop. The hardness of approximation of Euclidean k-means. In *Proceedings of the 31st International Symposium on Computational Geometry (SoCG)*, pages 754–767, 2015. doi:10.4230/LIPIcs.SOCG.2015.754.

9 Jaroslaw Byrka, Thomas W. Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median and positive correlation in budgeted optimization. *ACM Transaction on Algorithms (TALG)*, 13(2):23:1–23:31, 2017. doi:10.1145/2981561.

10 Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight FPT approximations for k-median and k-means. In *Proceedings of the 46th International Colloqium on Automata, Languages, and Programming (ICALP)*, pages 42:1–42:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.42.

11 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science (TCS)*, 38:293–306, 1985. doi:10.1016/0304-3975(85)90224-5.

12 Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of Algorithms*, 31(1):228–248, 1999. doi:10.1006/jagm.1998.0993.

13 Sudipto Guha, Adam Meyerson, and Kamesh Munagala. Hierarchical placement and network design problems. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 603–612, 2000. doi:10.1109/SFCS.2000.892328.

14 Dorit S. Hochbaum and David B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM*, 33(3):533–550, 1986. doi:10.1145/5925.5933.

15 Wen-Lian Hsu and George L. Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics (DAM)*, 1(3):209–215, 1979. doi:10.1016/0166-218X(79)90044-1.

16 Tanmay Inamdar and Kasturi Varadarajan. Capacitated sum-of-radii clustering: An FPT approximation. In *Proceedings of the 28th Annual European Symposium on Algorithms (ESA)*, volume 173 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 62:1–62:17, 2020. doi:10.4230/LIPIcs.ESA.2020.62.

17 Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM*, 50(6):795–824, 2003. doi:10.1145/950620.950621.

18 Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 731–740, 2002. doi:10.1145/509907.510012.

19 Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001. doi:10.1145/375827.375845.

20 David R. Karger and Maria Minkoff. Building Steiner trees with incomplete global knowledge. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 613–623, 2000. doi:10.1109/SFCS.2000.892329.

21 Euiwoong Lee, Melanie Schmidt, and John Wright. Improved and simplified inapproximability for k-means. *Information Processing Letters (IPL)*, 120:40–43, 2017. doi:10.1016/j.ipl.2016.11.009.

22 Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Information and Computation*, 222:45–58, 2013. doi:10.1016/j.ic.2012.01.007.

23 Shi Li. On facility location with general lower bounds. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2279–2290, 2019. doi:10.1137/1.9781611975482.138.

24 Guolong Lin, Chandrashekhar Nagarajan, Rajmohan Rajaraman, and David P. Williamson. A general approach for incremental approximation and hierarchical clustering. *SIAM Journal on Computing (SICOMP)*, 39(8):3633–3669, 2010. doi:10.1137/070698257.

25 Richard Matthew McCutchen and Samir Khuller. Streaming algorithms for k-center clustering with outliers and with anonymity. In *Proceedings of the 11th International Workshop on Approximation, Randomization and Combinatorial Optimization (APPROX)*, pages 165–178, 2008. doi:10.1007/978-3-540-85363-3_14.

26 Zoya Svitkina. Lower-bounded facility location. *ACM Transactions on Algorithms (TALG)*, 6(4):69, 2010. doi:10.1145/1824777.1824789.

27 Jens Vygen. Lecture notes – approximation algorithms for facility location problems, 2004/2005. accessed May 8th, 2019. URL: http://gett.or.uni-bonn.de/~vygen/files/fl.pdf.

Bidimensional Linear Recursive Sequences and Universality of Unambiguous Register Automata

Corentin Barloy
École Normale Supérieure de Paris, PSL, France

Lorenzo Clemente
University of Warsaw, Poland

Abstract

We study the universality and inclusion problems for register automata over equality data $(\mathbb{A}, =)$. We show that the universality $L(B) = (\Sigma \times \mathbb{A})^*$ and inclusion problems $L(A) \subseteq L(B)$ can be solved with 2-EXPTIME complexity when both automata are without guessing and B is unambiguous, improving on the currently best-known 2-EXPSPACE upper bound by Mottet and Quaas. When the number of registers of both automata is fixed, we obtain a lower EXPTIME complexity, also improving the EXPSPACE upper bound from Mottet and Quaas for fixed number of registers. We reduce inclusion to universality, and then we reduce universality to the problem of counting the number of orbits of runs of the automaton. We show that the orbit-counting function satisfies a system of bidimensional linear recursive equations with polynomial coefficients (linrec), which generalises analogous recurrences for the Stirling numbers of the second kind, and then we show that universality reduces to the zeroness problem for linrec sequences. While such a counting approach is classical and has successfully been applied to unambiguous finite automata and grammars over finite alphabets, its application to register automata over infinite alphabets is novel.

We provide two algorithms to decide the zeroness problem for bidimensional linear recursive sequences arising from orbit-counting functions. Both algorithms rely on techniques from linear non-commutative algebra. The first algorithm performs variable elimination and has elementary complexity. The second algorithm is a refined version of the first one and it relies on the computation of the Hermite normal form of matrices over a skew polynomial field. The second algorithm yields an EXPTIME decision procedure for the zeroness problem of linrec sequences, which in turn yields the claimed bounds for the universality and inclusion problems of register automata.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases unambiguous register automata, universality and inclusion problems, multi-dimensional linear recurrence sequences

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.8

Related Version *Full Version*: `https://arxiv.org/abs/2101.01033` [2]

Funding *Corentin Barloy*: Partially supported by the Polish NCN grant 2017/26/D/ST6/00201.
Lorenzo Clemente: Partially supported by the Polish NCN grant 2017/26/D/ST6/00201.

Acknowledgements We would like to thank Daniel Robertz for kindly providing us with the `LDA` package for Maple 16.

1 Introduction

Register automata. *Register automata* extend finite automata with finitely many registers holding values from an infinite *data domain* $\mathbb{A}$ which can be compared against the data appearing in the input. The study of register automata arises naturally in automata theory as a conservative generalisation of finite automata over finite alphabets Σ to richer but well-behaved classes of infinite alphabets. The seminal work of Kaminski and Francez introduced *finite-memory automata* as the study of register automata over the data domain $(\mathbb{A}, =)$

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 8; pp. 8:1–8:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

consisting of an infinite set $\mathbb{A}$ and the equality relation [21]. The recent book [3] studies automata theory over other data domains such as $(\mathbb{Q}, \leq)$, and more generally homogeneous [24] or even ω-categorical relational structures. Another motivation for the study of register automata comes from the area of database theory: XML documents can naturally be modelled as finite unranked trees where data values from an infinite alphabet are necessary to model the *attribute values* of the document (cf. [27] and the survey [33]).

The central verification question for register automata is the *inclusion problem*, which, for two given automata A, B, asks whether $L(A) \subseteq L(B)$. In full generality the problem is undecidable and this holds already in the special case of the *universality problem* $L(B) = (\Sigma \times \mathbb{A})^*$ [27, Theorem 5.1], when B has only two registers [3, Theorem 1.8] (or even just one register in the more powerful model *with guessing* [3, Exercise 9], i.e., non-deterministic reassignment in the terminology of [22]). One way to obtain decidability is to restrict the automaton B. One such restriction requires that B is *deterministic*: Since deterministic register automata are effectively closed under complementation, the inclusion problem reduces to non-emptiness of $L(A) \cap (\Sigma \times \mathbb{A})^* \setminus L(B)$, which can be checked in PSPACE. Another, incomparable, restriction demands that B has only one register: In this case the problem becomes decidable [21, Appendix A][1] and non-primitive recursive [18, Theorem 5.2].

Unambiguity. *Unambiguous automata* are a natural class of automata intermediate between deterministic and nondeterministic automata. An automaton is unambiguous if there is at most one accepting run on every input word. Unambiguity has often been used to generalise decidability results for deterministic automata at the price of a usually modest additional complexity. For instance, the universality problem for deterministic finite automata (which is PSPACE-complete in general [38]) is NL-complete, while for the unambiguous variant it is in PTIME [37, Corollary 4.7], and even in NC^2 [39]. An even more dramatic example is provided by universality of context-free grammars, which is undecidable in general [20, Theorem 9.22], PTIME-complete for deterministic context-free grammars, and decidable for unambiguous context-free grammars [31, Theorem 5.5] (even in PSPACE [12, Theorem 10]). (The more general equivalence problem is decidable for deterministic context-free grammars [34], but it is currently an open problem whether equivalence is decidable for unambiguous context-free grammars, as well as for the more general *multiplicity equivalence* of context-free grammars [23].) Other applications of unambiguity for universality and inclusion problems in automata theory include Büchi automata [5, 1], probabilistic automata [17], Parikh automata [7, 4], vector addition systems [16], and several others (cf. also [14, 15]).

Number sequences and the counting approach. The universality problem for a language over finite words $L \subseteq \Sigma^*$ is equivalent to whether its associated *word counting function* $f_L(n) := |L \cap \Sigma^n|$ equals $|\Sigma|^n$ for every n. The most classical way of exploiting unambiguity of a computation model A (finite automaton, context-free grammar, ...) is to use the fact that it yields a bijection between the recognised language $L(A)$ and the set of accepting runs. In this way, $f_L(n)$ is also the number of accepting runs of length n, and for the latter recursive descriptions usually exist. When the class of number sequences to which f_L belongs contains $|\Sigma|^n$ and is closed under difference, this is equivalent to the *zeroness* problem for $g(n) := |\Sigma|^n - f_L(n)$, which amounts to decide whether $g = 0$. This approach has been

[1] Decidability even holds for the so-called "two-window register automata", which combined with the restriction in [21] demanding that the last data value read must always be stored in some register boils down to a slightly more general class of "$1\frac{1}{2}$-register automata".

pioneered by Chomsky and Schützenberger [11] who have shown that the generating function $g_L(x) = \sum_{n=0}^{\infty} f_L(n) \cdot x^n$ associated to an unambiguous context-free language L is algebraic (cf. [6]). A similar observation by Stearns and Hunt [37] shows that $g_L(x)$ is rational [36, Chapter 4], when L is regular, and more recently by Bostan et al. [4] who have shown that $g_L(x)$ is holonomic [35] when L is recognised by an unambiguous Parikh automaton. Since the zeroness problem for rational, algebraic, and holonomic generating functions is decidable, one obtains decidability of the corresponding universality problems.

Unambiguous register automata. Returning to register automata, Mottet and Quaas have recently shown that the inclusion problem in the case where B is an unambiguous register automaton over equality data (without guessing) can be decided in 2-EXPSPACE, and in EXPSPACE when the numbers of registers of B is fixed [25, Theorem 1]. Note that already decidability is interesting, since unambiguous register automata without guessing are not closed under complement in the class of nondeterministic register automata without guessing [22, Example 4], and thus the classical approach via complementing B fails for register automata[2]. (In fact, even for finite automata complementation of unambiguous finite automata cannot lead to a PTIME universality algorithm, thanks to Raskin's recent super-polynomial lower-bound for the complementation problem for unambiguous finite automata in the class of non-deterministic finite automata [30]). Mottet and Quaas obtain their result by showing that inclusion can be decided by checking a reachability property of a suitable graph of triply-exponential size obtained by taking the product of A and B, and then applying the standard NL algorithm for reachability in directed graphs.

Our contributions. In view of the widespread success of the counting approach to unambiguous models of computation, one may wonder whether it can be applied to register automata as well. This is the topic of our paper. A naïve counting approach for register automata immediately runs into trouble since there are infinitely many data words of length n. The natural remedy is to use the fact that $\mathbb{A}^n$, albeit infinite, is *orbit-finite* [3, Sec. 3.2], which is a crucial notion generalising finiteness to the realm of relational structures used to model data. In this way, we naturally count the number of *orbits* of words/runs of a given length, which in the context of model theory is sometimes known as the *Ryll-Nardzewski function* [32]. For example, in the case of equality data $(\mathbb{A}, =)$, the number of orbits of words of length n is the well-known *Bell number* $B(n)$, and for $(\mathbb{Q}, \leq)$ one obtains the *ordered Bell numbers* (a.k.a. *Fubini numbers*); cf. Cameron's book for more examples [9, Ch. 7].

When considering orbits of runs, the run length n seems insufficient to obtain recurrence equations. To this end, we also consider the number of distinct data values k that appear on the word labelling the run. For instance, in the case of equality data, the corresponding orbit-counting function is the well-known sequence of *Stirling numbers of the second kind* $S(n,k) : \mathbb{Q}^{\mathbb{N}^2}$, which satisfies $S(0,0) = 1$, $S(m,0) = S(0,m) = 0$ for $m \geq 1$, and

$$S(n,k) = S(n-1,k-1) + k \cdot S(n-1,k), \qquad \text{for } n,k \geq 1. \tag{1}$$

These intuitions lead us to define the class of *bidimensional linear recursive sequences with polynomial coefficients* (linrec; cf. (2)) which are a class of number sequences in $\mathbb{Q}^{\mathbb{N}^2}$ satisfying a system of shift equations with polynomial coefficients generalising (1). Linrec are sufficiently

[2] In the more general class of register automata with guessing, an unproved conjecture proposed by Colcombet states that unambiguous register automata with guessing are effectively closed under complement [15, Theorem 12], implying decidability of the universality and containment problems for unambiguous register automata with guessing and, a posteriori, unambiguous register automata without guessing as considered in this paper. No published proof of this conjecture has appeared as of yet.

general to model the orbit-counting functions of register automata and yet amenable to algorithmic analysis. Our first result is a complexity upper bound for the zeroness problem for a class of linrec sequences which suffices to model register automata.

▶ **Theorem 1.** *The zeroness problem for linrec sequences with univariate polynomial coefficients from $\mathbb{Q}[k]$ is in* EXPTIME*.*

This is obtained by modelling linrec equations as systems of linear equations with *skew polynomial coefficients* (introduced by Ore [29]) and then using complexity bounds on the computation of the Hermite normal form of skew polynomial matrices by Giesbrecht and Kim [19]. Our second result is a reduction of the universality and inclusion problems to the zeroness problem of a system of linrec equations of exponential size. Together with Theorem 1, this yields improved upper bounds on the former problems.

▶ **Theorem 2.** *The universality $L(B) = (\Sigma \times \mathbb{A})^*$ and the inclusion problem $L(A) \subseteq L(B)$ for register automata A, B without guessing with B unambiguous are in* 2-EXPTIME*, and in* EXPTIME *for a fixed number of registers of A, B. The same holds for the equivalence problem $L(A) = L(B)$ when both automata are unambiguous.*

The rest of the paper is organised as follows. In Section 2, we introduce linrec sequences (cf. [2, Appendix A.3] for a comparison with well known sequence families from the literature such as the C-recursive, P-recursive, and the more recent polyrec sequences [8]). In Section 3, we introduce unambiguous register automata and we present an efficient reduction of the inclusion (and thus equivalence) problem to the universality problem, which allows us to concentrate on the latter in the rest of the paper. In Section 4, we present a reduction of the universality problem to the zeroness problem for linrec. In Section 5, we show with a simple argument based on elimination that the zeroness problem for linrec is decidable, and in Section 6 we derive a complexity upper bound using non-commutative linear algebra. Finally, in Section 7 we conclude with further work and an intriguing conjecture. Full proofs, additional definitions, and examples are provided in the full version of the paper [2].

Notation. Let $\mathbb{N}$, $\mathbb{Z}$, and $\mathbb{Q}$ be the set of non-negative integers, resp., rationals. The *height* of an integer $k \in \mathbb{Z}$ is $|k|_\infty = |k|$, and for a rational number $a \in \mathbb{Q}$ uniquely written as $a = \frac{p}{q}$ with $p \in \mathbb{Z}, q \in \mathbb{N}$ co-prime we define $|a|_\infty = \max\{|p|_\infty, |q|_\infty\}$. Let $\mathbb{Q}[n, k]$ denote the ring of bivariate polynomials. The *(combined) degree* $\deg P$ of $P = \sum_{i,j} a_{ij} n^i k^j \in \mathbb{Q}[n, k]$ is the maximum $i + j$ s.t. $a_{ij} \neq 0$ and the *height* $|P|_\infty$ is $\max_{i,j} |a_{ij}|_\infty$. For a nonempty set A and $n \in \mathbb{N}$, let A^n be the set of sequences of elements from A of length n, In particular, $A^0 = \{\varepsilon\}$ contains only the empty sequence ε. Let $A^* = \bigcup_{n\in\mathbb{N}} A^n$ be the set of all finite sequences over A. We use the *soft-Oh* notation $\tilde{O}(f(n))$ to denote $\bigcup_{c\geq 0} O(f(n) \cdot \log^c f(n))$.

2 Bidimensional linear recursive sequences with polynomial coefficients

Let $f(n, k) : \mathbb{Q}^{\mathbb{N}^2}$ be a bidimensional sequence. For $L \in \mathbb{N}$, the *first L-section* of f is the one-dimensional sequence $f(L, k) : \mathbb{Q}^{\mathbb{N}}$ obtained by fixing its first component to L; the *second L-section* $f(n, L)$ is defined similarly. The two *shift operators* $\partial_1, \partial_2 : \mathbb{Q}^{\mathbb{N}^2} \to \mathbb{Q}^{\mathbb{N}^2}$ are

$$(\partial_1 f)(n, k) = f(n + 1, k) \quad \text{and} \quad (\partial_2 f)(n, k) = f(n, k + 1), \quad \text{for all } n, k \geq 0.$$

An *affine operator* is a formal expression of the form $A = p_{00} + p_{01} \cdot \partial_1 + p_{10} \cdot \partial_2$ where $p_{00}, p_{01}, p_{10} \in \mathbb{Q}[n, k]$ are bivariate polynomials over n, k with rational coefficients. Let

$\{f_1, \ldots, f_m\}$ be a set of variables denoting bidimensional sequences[3]. A *system of linear shift equations* over $f_1, \ldots, f_m$ consists of m equations of the form

$$\left\{\begin{array}{rcl} \partial_1\partial_2 f_1 &=& A_{1,1}\cdot f_1 + \cdots + A_{1,m}\cdot f_m, \\ & \vdots & \\ \partial_1\partial_2 f_m &=& A_{m,1}\cdot f_1 + \cdots + A_{m,m}\cdot f_m, \end{array}\right. \tag{2}$$

where the $A_{i,j}$'s are affine operators. A bidimensional sequence $f : \mathbb{Q}^{\mathbb{N}^2}$ is *linear recursive of order m, degree d, and height h* (abbreviated, linrec) if the following two conditions hold:

1) there are auxiliary bidimensional sequences $f_2, \ldots, f_m : \mathbb{Q}^{\mathbb{N}^2}$ which together with $f = f_1$ satisfy a system of linear shift equations as in (2) where the polynomial coefficients have (combined) degree $\leq d$ and height $\leq h$.

2) for every $1 \leq i \leq m$ there are constants denoted $f_i(0, \geq 1), f_i(\geq 1, 0) \in \mathbb{Q}$ s.t. $f_i(0, k) = f_i(0, \geq 1)$ and $f_i(n, 0) = f_i(\geq 1, 0)$ for every $n, k \geq 1$.

If we additionally fix the initial values $f_1(0,0), \ldots, f_m(0,0)$, then the system (2) has a unique solution, which is computable in PTIME.

▶ **Lemma 3.** *The values $f_i(n,k)$'s are computable in deterministic time $\tilde{O}(m \cdot n \cdot k)$.*

In the following we will use the following effective closure under section.

▶ **Lemma 4.** *If $f : \mathbb{Q}^{\mathbb{N}^2}$ is linrec of order $\leq m$, degree $\leq d$, and height $\leq h$, then its L-sections $f(L,k), f(n,L) : \mathbb{Q}^{\mathbb{N}}$ are linrec of order $\leq m \cdot (L+3)$, degree $\leq d$, and height $\leq h \cdot L^d$.*

We are interested in the following central algorithmic problem for linrec.

Zeroness problem.
Input: A system of linrec equations (2) together with all initial conditions.
Output: Is it the case that $f_1 = 0$?

In Section 4 we use linrec sequences to model the orbit-counting functions of register automata, which we introduce next.

3 Unambiguous register automata

We consider register automata over the relational structure $(\mathbb{A}, =)$ consisting of a countable set $\mathbb{A}$ equipped with equality as the only relational symbol. Let $\bar{a} = a_1 \cdots a_n \in \mathbb{A}^n$ be a finite sequence of n data values. An *$\bar{a}$-automorphism* of $\mathbb{A}$ is a bijection $\alpha : \mathbb{A} \to \mathbb{A}$ s.t. $\alpha(a_i) = a_i$ for every $1 \leq i \leq n$, which is extended pointwise to $\bar{a} \in \mathbb{A}^n$ and to $L \subseteq \mathbb{A}^*$. For $\bar{b}, \bar{c} \in \mathbb{A}^n$, we write $\bar{b} \sim_{\bar{a}} \bar{c}$ whenever there is an $\bar{a}$-automorphism α s.t. $\alpha(\bar{b}) = \bar{c}$. The *$\bar{a}$-orbit* of $\bar{b}$ is the equivalence class $[\bar{b}]_{\bar{a}} = \{\bar{c} \in \mathbb{A}^n \mid \bar{b} \sim_{\bar{a}} \bar{c}\}$, and the set of $\bar{a}$-orbits of sequences in $L \subseteq \mathbb{A}^*$ is $\mathsf{orbits}_{\bar{a}}(L) = \{[\bar{b}]_{\bar{a}} \mid \bar{b} \in L\}$. In the special case when $\bar{a} = \varepsilon$ is the empty tuple, we just speak about *automorphism* α and *orbit* $[\bar{b}]$. A set X is *orbit-finite* if $\mathsf{orbits}(X)$ is a finite set [3, Sec. 3.2]. All definitions above extend to $\mathbb{A}_\perp := \mathbb{A} \cup \{\perp\}$ with $\perp \notin \mathbb{A}$ in the expected way. A *constraint* φ is a quantifier-free[4] formula generated by $\varphi, \psi ::\equiv x = \perp \mid x = y \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \neg\varphi$, where x, y are variables and $\perp$ is a special constant denoting an undefined value. The semantics of a constraint

[3] We abuse notation and silently identify variables denoting sequences with the sequences they denote.

[4] Since $(\mathbb{A}, =)$ is a homogeneous relational structure, and thus it admits quantifier elimination, we would obtain the same expressive power if we would consider more general first-order formulas instead.

$\varphi(x_1, \ldots, x_n)$ with n free variables $x_1, \ldots, x_n$ is the set of tuples of n elements which satisfies: $[\![\varphi]\!] = \{a_1, \ldots, a_n \in \mathbb{A}_\perp^n \mid \mathbb{A}_\perp, x_1 : a_1, \ldots, x_n : a_n \models \varphi\}$. A *register automaton* of *dimension* $d \in \mathbb{N}$ is a tuple $A = (d, \Sigma, \mathsf{L}, \mathsf{L}_I, \mathsf{L}_F, \rightarrow)$ where d is the number of registers, Σ is a finite alphabet, L is a finite set of *control locations*, of which we distinguish those which are *initial* $\mathsf{L}_I \subseteq \mathsf{L}$, resp., *final* $\mathsf{L}_F \subseteq \mathsf{L}$, and "$\rightarrow$" is a set of rules of the form $p \xrightarrow{\sigma, \varphi} q$, where $p, q \in \mathsf{L}$ are control locations, $\sigma \in \Sigma$ is an input symbol from the finite alphabet, and $\varphi(x_1, \ldots, x_d, y, x_1', \ldots, x_d')$ is a constraint relating the current register values x_i's, the current input symbol (represented by the variable y), and the next register values of x_i''s.

▶ **Example 5.** Let A over $|\Sigma| = 1$ have one register x, and four control locations p, q, r, s, of which p is initial and s is final. The transitions are $p \xrightarrow{x=\perp \wedge x'=y} q$, $p \xrightarrow{x=\perp \wedge x'=y} r$, $q \xrightarrow{x \neq y \wedge x'=x} q$, $q \xrightarrow{x=y \wedge x'=x} s$, $r \xrightarrow{x=y \wedge x'=x} r$, and $r \xrightarrow{x \neq y \wedge x'=x} s$. The automaton accepts all words of the form $a(\mathbb{A} \setminus \{a\})^* a$ or $aa^*(\mathbb{A} \setminus \{a\})$ with $a \in \mathbb{A}$.

A register automaton is *orbitised* if every constraint φ appearing in some transition thereof denotes an orbit $[\![\varphi]\!] \in \mathsf{orbits}(\mathbb{A}_\perp^{2 \cdot d+1})$. For example, when $d = 1$ the constraint $\varphi \equiv x = x'$ is not orbitised, however $[\![\varphi]\!] = [\![\varphi_0]\!] \cup [\![\varphi_1]\!]$ splits into two disjoint orbits for the orbitised constraints $\varphi_0 \equiv x = x' \wedge x = y$ and $\varphi_1 \equiv x = x' \wedge x \neq y$. The automaton from Example 5 is orbitised. Every register automaton can be transformed in orbitised form by replacing every transition $p \xrightarrow{\sigma, \varphi} q$ with exponentially many transitions $p \xrightarrow{\sigma, \varphi_1} q, \ldots, p \xrightarrow{\sigma, \varphi_n} q$, for each orbit $[\![\varphi_i]\!]$ of $[\![\varphi]\!] \subseteq \mathbb{A}_\perp^{2 \cdot d+1}$.

A *register valuation* is a tuple of (possibly undefined) values $\bar{a} = (a_1, \ldots, a_d) \in \mathbb{A}_\perp^d$. A *configuration* is a pair $(p, \bar{a})$, where $p \in \mathsf{L}$ is a control location and $\bar{a} \in \mathbb{A}_\perp^d$ is a register valuation; it is *initial* if $p \in \mathsf{L}_I$ is initial and all registers are initially undefined $\bar{a} = (\perp, \ldots, \perp)$, and it is *final* whenever $p \in \mathsf{L}_F$ is so. The *semantics* of a register automaton A is the infinite transition system $[\![A]\!] = (C, C_I, C_F, \rightarrow)$ where C is the set of configurations, of which $C_I, C_F \subseteq C$ are the initial, resp., final ones, and $\rightarrow \subseteq C \times (\Sigma \times \mathbb{A}) \times C$ is the set of all transitions of the form

$$(p, \bar{a}) \xrightarrow{\sigma, a} (q, \bar{a}'), \qquad \text{with } \sigma \in \Sigma, a \in \mathbb{A}, \text{ and } \bar{a}, \bar{a}' \in \mathbb{A}_\perp^d,$$

s.t. there exists a rule $p \xrightarrow{\sigma, \varphi} q$ where satisfying the constraint $\mathbb{A}_\perp, \bar{x} : \bar{a}, y : a, \bar{x}' : \bar{a}' \models \varphi$. A *data word* is a sequence $w = (\sigma_1, a_1) \cdots (\sigma_n, a_n) \in (\Sigma \times \mathbb{A})^*$. A *run over* a data word w *starting at* $c_0 \in C$ and *ending at* $c_n \in C$ is a sequence π of transitions of $[\![A]\!]$ of the form $\pi = c_0 \xrightarrow{\sigma_1, a_1} c_1 \xrightarrow{\sigma_2, a_2} \cdots \xrightarrow{\sigma_n, a_n} c_n$. We denote with $\mathsf{Runs}(c_0; w; c_n)$ the set of runs over w starting at c_0 and ending in c_n, and with $\mathsf{Runs}(C_I; w; c_n)$ the set of *initial runs*, i.e., those runs over w starting at some initial configuration $c_0 \in C_I$ and ending in c_n. The run π is *accepting* if $c_n \in C_F$. The language $L(A, c)$ recognised from configuration $c \in C$ is the set of data words labelling some accepting run starting at c; the language recognised from a set of configurations $D \subseteq C$ is $L(A, D) = \bigcup_{c \in D} L(A, c)$, and the language recognised by the register automaton A is $L(A) = L(A, C_I)$. Similarly, the *backward language* $L^{\mathsf{R}}(A, c)$ is the set of words labelling some run starting at an initial configuration and ending at c. Thus, we also have $L(A) = L^{\mathsf{R}}(A, C_F)$. A register automaton is *deterministic* if for every input word there exists at most one initial run, and *unambiguous* if for every input word there is at most one initial and accepting run. A register automaton is *without guessing* if, for every initial run $(p, \perp^d) \xrightarrow{w} (q, \bar{a})$ every non-$\perp$ data value in $\bar{a}$ occurs in the input w, written $\bar{a} \subseteq w$. In the rest of the paper we will study exclusively automata without guessing. A deterministic automaton is unambiguous and without guessing. These semantic properties can be decided in PSPACE with simple reachability analyses (cf. [15]).

▶ **Example 6.** The automaton from Example 5 is unambiguous and without guessing. An example of language which can only be recognised by ambiguous register automata is the set of words where the same data value appears two times $L = \{u \cdot a \cdot v \cdot a \cdot w \mid a \in \mathbb{A}; u, v, w \in \mathbb{A}^*\}$.

▶ **Lemma 7.** *If A is an unambiguous register automaton, then there is a bijection between the language it recognises $L(A) = L(A, C_I) = L^{\mathsf{R}}(A, C_F)$ and the set of runs starting at some initial configuration in C_I and ending at some final configuration in C_F.*

We are interested in the following decision problem.

Inclusion problem.
Input: Two register automata A, B over the same input alphabet Σ.
Output: Is it the case that $L(A) \subseteq L(B)$?

The *universality problem* asks $L(A) = (\Sigma \times \mathbb{A})^*$, and the *equivalence problem* $L(A) = L(B)$. In general, universality reduces to equivalence, which in turn reduces to inclusion. In our context, inclusion reduces to universality and thus all three problems are equivalent.

▶ **Lemma 8.** *Let A and B be two register automata.*

1. *The inclusion problem $L(A) \subseteq L(B)$ with A orbitised and without guessing reduces in* PTIME *to the case where A is deterministic. The reduction preserves whether B is 1) unambiguous, 2) without guessing, and 3) orbitised.*
2. *The inclusion problem $L(A) \subseteq L(B)$ with A deterministic reduces in* PTIME *to the universality problem for some register automaton C. If B is unambiguous, then so is C. If B is without guessing, then so is C. If A and B are orbitised, then so is C.*

4 Universality of unambiguous register automata without guessing

We reduce universality of unambiguous register automata without guessing to zeroness of bidimensional linrec sequences with univariate polynomial coefficients. The *width* of a sequence of data values $\bar{a} = a_1 \cdots a_n \in \mathbb{A}^n$ is $\#\bar{a} = |\{a_1, \ldots, a_n\}|$, for a word $w = (\sigma_1, a_1) \cdots (\sigma_n, a_n) \in (\Sigma \times \mathbb{A})^*$ we set $\#w = \#(a_1 \cdots a_n)$, and for a run π over w we set $\#\pi = \#w$. Let the *Ryll-Nardzewski function* $G_{p,\bar{a}}(n, k)$ of a configuration $(p, \bar{a}) \in C = \mathsf{L} \times \mathbb{A}^d_\perp$ count the number of $\bar{a}$-orbits of initial runs of length n and width k ending in $(p, \bar{a})$:

$$G_{p,\bar{a}}(n, k) = |\{[\pi]_{\bar{a}} \mid w \in (\Sigma \times \mathbb{A})^n, \pi \in \mathsf{Runs}(C_I; w; p, \bar{a}), \#w = k\}| . \tag{3}$$

▶ **Lemma 9.** *Let $\bar{a}, \bar{b} \in \mathbb{A}^d_\perp$. If $[\bar{a}] = [\bar{b}]$, then $G_{p,\bar{a}}(n, k) = G_{p,\bar{b}}(n, k)$ for every $n, k \geq 0$.*

We thus overload the notation and write $G_{p,[\bar{a}]}$ instead of $G_{p,\bar{a}}$. Since $\mathbb{A}^d_\perp$ is orbit-finite, this yields finitely many variables $G_{p,[\bar{a}]}$'s. By slightly abusing notation, let $G_{C_F}(n, k) = \sum_{[(p,\bar{a})] \in \mathsf{orbits}(C_F)} G_{p,[\bar{a}]}(n, k)$ be the sum of the Ryll-Nardzewski function over all orbits of accepting configurations. When the automaton is unambiguous, thanks to Lemma 7, $G_{C_F}(n, k)$ is also the number of orbits of accepted words of length n and width k.

▶ **Lemma 10.** *Let A be an unambiguous register automaton w/o guessing over Σ and let $S_\Sigma(n, k)$ be the number of orbits of all words of length n and width k. We have $L(A) = (\mathbb{A} \times A)^*$ if, and only if, $\forall n, k \in \mathbb{N} \cdot G_{C_F}(n, k) = S_\Sigma(n, k)$.*

In other words, universality of A reduces to zeroness of $G := S_\Sigma - G_{C_F}$. The sequence S_Σ is linrec since it satisfies the recurrence in Figure 2 with initial conditions $S_\Sigma(0, 0) = 1$ and $S_\Sigma(n + 1, 0) = S_\Sigma(0, k + 1) = 0$ for $n, k \geq 0$. We show that all the sequences of the form

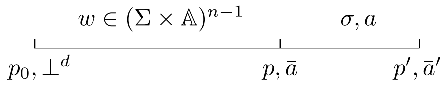


Figure 1 Last-step decomposition.

$$\begin{aligned}
G_{p',[\bar a']}(n+1,k+1) = {} & \sum_{[p,\bar a \xrightarrow{\sigma,a} p',\bar a']:\ a\in\bar a} \underbrace{G_{p,[\bar a]}(n,k+1)}_{\mathbf{I}} + {} \\
& \sum_{[p,\bar a \xrightarrow{\sigma,a} p',\bar a']:\ a\not\in\bar a} \left(\underbrace{G_{p,[\bar a]}(n,k)}_{\mathbf{II}} + \underbrace{\max(k+1-\#[\bar a],0)\cdot G_{p,[\bar a]}(n,k+1)}_{\mathbf{III}}\right), \\
S_\Sigma(n+1,k+1) = {} & |\Sigma|\cdot S_\Sigma(n,k) + |\Sigma|\cdot(k+1)\cdot S_\Sigma(n,k+1), \\
G(n,k) = {} & S_\Sigma(n,k) - \sum_{[p,\bar a]\in\mathsf{orbits}(C_F)} G_{p,[\bar a]}(n,k).
\end{aligned}$$

Figure 2 Linrec automata equations.

$G_{p,[\bar a]}$ are also linrec and thus also G will be linrec. We perform a last-step decomposition of an initial run; cf. Figure 1. Starting from some initial configuration $(p_0, \bot^d)$, the automaton has read a word w of length $n-1$ leading to $(p, \bar a)$. Then, the automaton reads the last letter (σ, a) and goes to $(p', \bar a')$ via the transition $t = (p, \bar a \xrightarrow{\sigma,a} p', \bar a')$. The question is in how many distinct ways can an orbit of the run over w be extended into an orbit of the run over $w \cdot (\sigma, a)$. We distinguish three cases.

I: Assume that a appears in register $\bar a_i = a$. Since the automaton is without guessing, $a \in w$ has appeared earlier in the input word and $\bar a' \subseteq \bar a$ (ignoring $\bot$'s). Thus, each $\bar a$-orbit of runs $[p_0, \bot^d \xrightarrow{w} p, \bar a]_{\bar a}$ yields, via the fixed t, an $\bar a'$-orbit of runs $[p_0, \bot^d \xrightarrow{w} p, \bar a \xrightarrow{\sigma,a} p', \bar a']_{\bar a'}$ of the same width in just one way.

II: Assume that a is globally fresh $a \not\in w$, and thus in particular $a \not\in \bar a$ since the automaton is without guessing. Each $\bar a$-orbit of runs $[p_0, \bot^d \xrightarrow{w} p, \bar a]_{\bar a}$ of width $\#w$ yields, via the fixed t, a single $\bar a'$-orbit of runs $[p_0, \bot^d \xrightarrow{w} p, \bar a \xrightarrow{\sigma,a} p', \bar a']_{\bar a'}$ of width $\#(w \cdot a) = \#w + 1$.

III: Assume that $a \in w$ is not globally fresh, but it does not appear in any register $a \not\in \bar a$. Since the automaton is without guessing, every value in $\bar a$ appears in w. Consequently, a can be any of the $\#w$ distinct values in w, with the exception of $\#\bar a$ values. Each $\bar a$-orbit of runs $[p_0, \bot \xrightarrow{w} p, \bar a]_{\bar a}$ of width $\#w$ yields $\#w - \#\bar a \geq 0$ $\bar a'$-orbits of runs $[p_0, \bot^d \xrightarrow{w} p, \bar a \xrightarrow{\sigma,a} p', \bar a']_{\bar a'}$ of the same width.

(As expected, we do not need unambiguity at this point, since we are counting orbits of runs.) We obtain the equations in Figure 2, where the sums range over orbits of transitions. This set of equations is finite since there are finitely many orbits $[\bar a] \in \mathsf{orbits}(\mathbb{A}_\bot^d)$ of register valuations, and moreover we can effectively represent each orbit by a constraint [3, Ch. 4]. Strictly speaking, the equations are not linrec due to the "max" operator, however they can easily be transformed to linrec by considering $G_{p,[\bar a]}(n, K)$ separately for $1 \leq K < d$; in the interest of clarity, we omit the full linrec expansion. The initial condition is $G_{p,[\bar a]}(0,0) = 1$ if $p \in I$ initial, and $G_{p,[\bar a]}(0,0) = 0$ otherwise. The two 0-sections satisfy $G_{p,[\bar a]}(n+1,0) = 0$ for $n \geq 0$ (if the word is nonempty, then there is at least one data value) and $G_{p,[\bar a]}(0,k+1) = 0$ for $k \geq 0$ (an empty word does not have any data value).

▶ **Lemma 11.** *The sequences $G_{p,[\bar a]}$'s satisfy the system of equations in Figure 2.*

▶ **Example 12.** The equations corresponding to the automaton in Example 5 are as follows. (Since the automaton is orbitised, we can omit the orbit.) We have $G_p(0,0) = 1$, $G_q(0,0) = G_r(0,0) = G_s(0,0) = 0$ and for $n, k \geq 0$:

$$\begin{aligned}
&G_p(n+1,k+1) = 0,\\
&G_q(n+1,k+1) = \underbrace{G_p(n,k)}_{\mathbf{II}} + \underbrace{(k+1)\cdot G_p(n,k+1)}_{\mathbf{III}} + \underbrace{G_q(n,k)}_{\mathbf{II}} + \underbrace{k\cdot G_q(n,k+1)}_{\mathbf{III}},\\
&G_r(n+1,k+1) = \underbrace{G_p(n,k)}_{\mathbf{II}} + \underbrace{(k+1)\cdot G_p(n,k+1)}_{\mathbf{III}} + \underbrace{G_r(n,k+1)}_{\mathbf{I}},\\
&G_s(n+1,k+1) = \underbrace{G_q(n,k+1)}_{\mathbf{I}} + \underbrace{G_r(n,k)}_{\mathbf{II}} + \underbrace{k\cdot G_r(n,k+1)}_{\mathbf{III}}.
\end{aligned}$$

▶ **Lemma 13.** *Let A be an unambiguous register automaton over equality atoms without guessing with d registers and ℓ control locations. The universality problem for A reduces to the zeroness problem of the linrec sequence G defined by the system of equations in Figure 2 containing $O(\ell \cdot 2^{d\cdot\log d})$ variables and equations and constructible in* PSPACE. *If A is already orbitised, then the system of equations has size $O(\ell)$.*

5 Decidability of the zeroness problem

In this section, we present an algorithm to solve the zeroness problem of bidimensional linrec sequences with univariate polynomial coefficients, which is sufficient for linrec sequences from Figure 2. We first give a general presentation on elimination for bivariate polynomial coefficients, and then we use the univariate assumption to obtain a decision procedure. We model the non-commutative operators appearing in the definition of linrec sequences (2) with Ore polynomials (a.k.a. skew polynomials) [29][5]. Let R be a (not necessarily commutative) ring and σ an automorphism of R. The ring of *(shift) skew polynomials* $R[\partial;\sigma]$ is defined as the ring of polynomials but where the multiplication operation satisfies the following commutation rule: For a coefficient $a \in R$ and the unknown ∂, we have

$$\partial \cdot a = \sigma(a) \cdot \partial.$$

(The usual ring of polynomials is recovered when σ is the identity.) The multiplication extends to monomials as $a\partial^k \cdot b\partial^l = a\sigma^k(b)\cdot\partial^{k+l}$ and to the whole ring by distributivity. The *degree* of a skew monomial $a \cdot \partial^k$ is k, and the degree $\deg P$ of a skew polynomial P is the maximum of the degrees of its monomials. The degree function satisfies the expected identities $\deg(P\cdot Q) = \deg P + \deg Q$ and $\deg(P+Q) \leq \max(\deg P, \deg Q)$. A skew polynomial is *monic* if the coefficient of its monomial of highest degree is 1. The crucial and only property that we need in this section is that skew polynomial rings admit a Euclidean pseudo-division algorithm, which in turns allows one to find common left multiples. A skew polynomial ring $R[\partial;\sigma]$ has *pseudo-division* if for any two skew polynomials $A, B \in R[\partial;\sigma]$ with $\deg A \geq \deg B$ there is a coefficient $a \in R$ and skew polynomials $Q, R \in R[\partial;\sigma]$ s.t. $a \cdot A = P \cdot B + Q$ and $\deg Q < \deg B$. We say that a ring R has the *common left multiple* (CLM) property if for every $a, b \neq 0$, there exists $c, d \neq 0$ such that $c \cdot a = d \cdot b$.

[5] The general definition of the Ore polynomial ring $R[\partial;\sigma,\delta]$ uses an additional component $\delta : R \to R$ in order to model differential operators. We present a simplified version which is enough for our purposes.

▶ **Theorem 14** (cf. [28, Sec. 1]). *If R has the* CLM *property, then*
1) *$R[\partial;\sigma]$ has a pseudo-division, and*
2) *$R[\partial;\sigma]$ also has the* CLM *property.*
The most important instances of skew polynomials are the *first* and *second Weyl algebras*:

$$W_1 = \mathbb{Q}[n,k][\partial_1;\sigma_1] \quad \text{and} \quad W_2 = W_1[\partial_2;\sigma_2] = \mathbb{Q}[n,k][\partial_1;\sigma_1][\partial_2;\sigma_2], \tag{4}$$

where $\mathbb{Q}[n,k]$ is the ring of bivariate polynomials, and the shifts satisfy $\sigma_1(p(n,k)) := p(n+1,k)$ and $\sigma_2\left(\sum_i p_i(n,k)\partial_1^i\right) := \sum_i p_i(n,k+1)\partial_1^i$. Skew polynomials in W_2 act on bidimensional sequences $f : \mathbb{Q}^{\mathbb{N}^2}$ by interpreting ∂_1 and ∂_2 as the two shifts. A linrec system of equations (2) can thus be interpreted as a system of linear equations with variables $f_1, \ldots, f_m$ and coefficients in W_2.

▶ **Example 15.** Continuing our running Example 12, we obtain the following linear system of equations with W_2 coefficients:

$$\begin{array}{rrrrr}
\partial_1\partial_2 \cdot G_p & & & & = 0, \\
-(1+(k+1)\partial_2)\cdot G_p & +(\partial_1\partial_2 - k\partial_2 - 1)\cdot G_q & & & = 0, \\
-(1+(k+1)\partial_2)\cdot G_p & & +(\partial_1\partial_2 - \partial_2)\cdot G_r & & = 0, \\
& -\partial_2 \cdot G_q & -(1+k\partial_2)\cdot G_r & +\partial_1\partial_2\cdot G_s & = 0,
\end{array}$$

$$(\partial_1\partial_2 - (k+1)\partial_2 - 1)\cdot S_1 = 0,$$
$$G_s - S_1 + G = 0.$$

Since $W_0 = \mathbb{N}[n,k]$ is commutative, it obviously has the CLM property. By two applications of Theorem 14, we have (see [2, Appendix D.1] for CLM examples):

▶ **Corollary 16.** *The two Weyl algebras W_1 and W_2 have the* CLM *property.*

A (linear) *cancelling relation* (CR) for a bidimensional sequence $f : \mathbb{Q}^{\mathbb{N}^2}$ is a linear equation of the form

$$p_{i^*,j^*}(n,k)\cdot\partial_1^{i^*}\partial_2^{j^*} f = \sum_{(i,j)<_{\text{lex}}(i^*,j^*)} p_{i,j}(n,k)\cdot\partial_1^i\partial_2^j f, \tag{CR-2}$$

where $p_{i^*,j^*}(n,k), p_{i,j}(n,k) \in \mathbb{Q}[n,k]$ are bivariate polynomial coefficients and $<_{\text{lex}}$ is the lexicographic ordering. Cancelling relations for a one-dimensional sequence $g : \mathbb{Q}^{\mathbb{N}}$ are defined analogously (we use the second variable k as the index for convenience):

$$q_{j^*}(k)\cdot\partial_2^{j^*} g = \sum_{0\le j<j^*} q_j(k)\cdot\partial_2^j g. \tag{CR-1}$$

We use cancelling relations as certificates of zeroness for f when the $p_{i,j}$'s are univariate. We do not need to construct any cancelling relation, just knowing that some exists with the required bounds suffices.

▶ **Lemma 17.** *The zeroness problem for a bidimensional linrec sequence $f : \mathbb{Q}^{\mathbb{N}^2}$ of order $\le m$ and univariate polynomial coefficients in $\mathbb{Q}[k]$ admitting some cancelling relation* (CR-2) *with leading coefficient $p_{i^*,j^*}(k) \in \mathbb{Q}[k]$ of degree $\le e$ and height $\le h$ s.t. each of the one-dimensional sections $f(M,k) \in \mathbb{Q}^{\mathbb{N}}$ for $1 \le M \le i^*$ also admits some cancelling relation* (CR-1) *of ∂_2-degree $\le d$ with leading polynomial coefficients of degrees $\le e$ and height $\le h$ is decidable in deterministic time $\tilde{O}(p(m,i^*,j^*,d,e,h))$ for some polynomial p.*

Elimination already yields decidability with elementary complexity for the zeroness problem and thus for the universality/equivalence/inclusion problems of unambiguous register automata without guessing.

▶ **Theorem 18.** *The zeroness problem for linrec sequences with univariate polynomial coefficients from $\mathbb{Q}[k]$ (or from $\mathbb{Q}[n]$) is decidable.*

▶ **Example 19.** Continuing our running Example 15, we subsequently eliminate G_p, G_s, G_r, G_q, S finally obtaining (cf. [2, Example 34 in Appendix D.2] for details)

$$\begin{aligned} G(n+4,k+4) = \quad & (k+3)\cdot G(n+3,k+4) + G(n+3,k+3) + \\ & -(k+2)\cdot G(n+2,k+4) - G(n+2,k+3). \end{aligned} \tag{5}$$

As expected, all coefficients are polynomials in $\mathbb{Q}[k]$ and in particular they do not involve the variable n. Moreover, we note that the relation above is *monic*, in the sense that the lexicographically leading term $G(n+4,k+4)$ has coefficient 1 (cf. Section 7). (Cf. [2, Example 35] for elimination in a two-register automaton and [2, Example 36] for a one-register automaton accepting all words of length ≥ 2.)

We omit a precise complexity analysis of elimination because better bounds can be obtained by resorting to linear non-commutative algebra, which is the topic of the next section.

6 Complexity of the zeroness problem

In this section we present an EXPTIME algorithm to solve the zeroness problem and we apply this result to register automata. We compute the *Hermite normal form* (HNF) of the matrix with skew polynomial coefficients associated to (2) in order to do elimination in a more efficient way. The complexity bounds provided by Giesbrecht and Kim [19] on the computation of the HNF lead to the following bounds for cancelling relations; cf. [2, Appendix E] for further details and full proofs.

▶ **Lemma 20.** *A linrec sequence $f \in \mathbb{Q}^{\mathbb{N}^2}$ of order $\leq m$, degree $\leq d$, and height $\leq h$ admits a cancelling relation* (CR-2) *with the orders i^*, j^* and the degree of p_{i^*,j^*} polynomially bounded, and with height $|p_{i^*,j^*}|_\infty$ exponentially bounded. Similarly, its one-dimensional sections $f(0,k),\dots,f(i^*,k) \in \mathbb{Q}^{\mathbb{N}}$ also admit cancelling relations* (CR-1) *of polynomially bounded orders and degree, and exponentially bounded height.*

This allows us to prove below the EXPTIME upper-bound for zeroness of Theorem 1, and the 2-EXPTIME algorithm for inclusion of Theorem 2.

Proof of Theorem 1. Thanks to the bounds from Lemma 20, i^*, j^* are polynomially bounded; we can find a polynomial bound d on the ∂_2-degrees of the cancelling relations $R_0, \dots, R_{i^*}$ for the sections $f(0,k),\dots,f(i^*,k)$, respectively; we can find a polynomial bound e on the degrees of $p_{i^*,j^*}(k)$ and the leading polynomial coefficients of the R_i's; and an exponential bound h on $|p_{i^*,j^*}|_\infty$ and the heights of the leading polynomial coefficients of the R_i's. We thus obtain an EXPTIME algorithm by Lemma 17. ◀

This yields the announced upper-bounds for the inclusion problem for register automata.

Proof of Theorem 2. For the universality problem $L(B) = (\Sigma \times \mathbb{A})^*$, let d be the number of registers and ℓ the number of control locations of B. By Lemma 13, the universality problem reduces in PSPACE to zeroness of a linrec system with polynomial coefficients in $\mathbb{Q}[k]$

containing $O(\ell \cdot 2^{d \cdot \log d})$ variables $G_{p,[\bar{a}]}$ and the same number of equations. By Theorem 1, we get a 2-EXPTIME algorithm. When the numbers of registers d is fixed, we get an EXPTIME algorithm. For the inclusion problem $L(A) \subseteq L(B)$, we first orbitise A into an equivalent orbitised register automaton without guessing A'. A close inspection of the two constructions leading to C in the proof of Lemma 8 reveal that transitions in C are either transitions from A' (and thus already orbitised), or pairs of a transition in B together with a transition in A', the second of which is already orbitised. It follows that orbitising C incurs in an exponential blow-up w.r.t. the number of registers of B, but only polynomial w.r.t. the number of registers of A' (and thus of A), since the A'-part in C is already orbitised. Consequently, we can write (in PSPACE) a system of linrec equations for the universality problem of C of size exponential in the number of registers of A and of B. By reasoning as in the first part of the proof, we obtain a EXPTIME algorithm for the universality problem of C, and thus a 2-EXPTIME algorithm for the original inclusion problem $L(A) \subseteq L(B)$. If both the number of registers of A and of B is fixed, we get an EXPTIME algorithm. The equivalence problem $L(A) = L(B)$ with both automata A, B unambiguous reduces to two inclusion problems. ◀

7 Further remarks and conclusions

We say that $P = \sum_{i,j} p_{i,j}(n,k) \cdot \partial_1^i \partial_2^j$ is *monic* if $p_{i^*,j^*} = 1$ where (i^*, j^*) is the lexicographically largest pair (i,j) s.t. $p_{i,j} \neq 0$. The cancelling relation (CR-2) in our examples (5) and [2, (10), (11), (15)] happens to be monic in this sense.

▶ **Conjecture 21** (Monicity conjecture)**.** *There always exists a* monic *cancelling relation* (CR-2) *for linrec systems obtained from automata equations in Figure 2, and similarly for their sections* (CR-1)*.*

Conjecture 21 has important algorithmic consequences. The exponential complexity in Theorem 1 comes from the exponential growth of the rational number coefficients (heights) in the HNF. This is due to the use of Lemma 17, whose complexity depends on the maximal root of the leading polynomial $p_{i^*,j^*}(n,k)$ from (CR-2). If Conjecture 21 holds, then $p_{i^*,j^*}(n,k) = 1$, Lemma 17 would yield a PTIME algorithm for zeroness, and consequently all complexities in Theorem 2, would drop by one exponential. This provides ample motivation to investigate the monicity conjecture.

In order to obtain the lower EXPTIME complexity for $L(A) \subseteq L(B)$ in Theorem 2 we have to fix the number of registers in *both* automata A and B. The EXPSPACE upper bound of Mottet and Quaas [25] holds already when only the number of registers of B is fixed, while we only obtain a 2-EXPTIME upper bound in this case. It is left for future work whether the counting approach can yield better bounds without fixing the number of registers of A.

The fact that the automata are non-guessing is crucial in each of the cases **I**, **II**, and **III** of the equations in Figure 2 in order to correctly count the number of orbits of runs. For automata with guessing from the fact that the current input a is stored in a register we cannot deduce that a actually appeared previously in the input word w, and thus our current parametrisation in terms of length and width does not lead to a recursive characterisation.

Finally, it is also left for further work to extend the counting approach to other data domains such as total order atoms, random graph atoms, etc..., and, more generally, to arbitrary homogeneous and ω-categorical atoms under suitable computability assumptions (cf. [13]), and to other models of computation such as register pushdown automata [10, 26].

References

1 Christel Baier, Stefan Kiefer, Joachim Klein, Sascha Klüppelholz, David Müller, and James Worrell. Markov Chains and Unambiguous Büchi Automata. In Swarat Chaudhuri and Azadeh Farzan, editors, *Proc. of CAV'16*, pages 23–42, Cham, 2016. Springer International Publishing.

2 Corentin Barloy and Lorenzo Clemente. Bidimensional linear recursive sequences and universality of unambiguous register automata. *arXiv e-prints*, January 2021. `arXiv:2101.01033`.

3 Mikołaj Bojańczyk. Slightly infinite sets, 2019. URL: `https://www.mimuw.edu.pl/~bojan/paper/atom-book`.

4 Alin Bostan, Arnaud Carayol, Florent Koechlin, and Cyril Nicaud. Weakly-Unambiguous Parikh Automata and Their Link to Holonomic Series. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *Proc. of ICALP'20*, volume 168 of *LIPIcs*, pages 114:1–114:16, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

5 Nicolas Bousquet and Christof Löding. Equivalence and inclusion problem for strongly unambiguous büchi automata. In Adrian-Horia Dediu, Henning Fernau, and Carlos Martín-Vide, editors, *Proc. of LATA'10*, pages 118–129, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

6 Mireille Bousquet-Mélou. Algebraic generating functions in enumerative combinatorics and context-free languages. In Volker Diekert and Bruno Durand, editors, *Proc. of STACS'05*, pages 18–35, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

7 Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Unambiguous constrained automata. In Hsu-Chun Yen and Oscar H. Ibarra, editors, *Proc. of DLT'12*, volume 7410 of *LNCS*, pages 239–250. Springer Berlin Heidelberg, 2012.

8 Michaël Cadilhac, Filip Mazowiecki, Charles Paperman, Michał Pilipczuk, and Géraud Sénizergues. On Polynomial Recursive Sequences. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *Proc. of ICALP'20*, volume 168 of *LIPIcs*, pages 117:1–117:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

9 Peter J. Cameron. *Notes on Counting: An Introduction to Enumerative Combinatorics*. Australian Mathematical Society Lecture Series. Cambridge University Press, 1 edition, 2017.

10 Edward Y. C. Cheng and Michael Kaminski. Context-free languages over infinite alphabets. *Acta Inf.*, 35(3):245–267, 1998.

11 N. Chomsky and M. P. Schützenberger. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, volume 35 of *Studies in Logic and the Foundations of Mathematics*, pages 118–161. Elsevier, 1963.

12 Lorenzo Clemente. On the complexity of the universality and inclusion problems for unambiguous context-free grammars. In Laurent Fribourg and Matthias Heizmann, editors, Proceedings 8th International Workshop on *Verification and Program Transformation* and 7th Workshop on *Horn Clauses for Verification and Synthesis,* Dublin, Ireland, 25-26th April 2020, volume 320 of *EPTCS*, pages 29–43. Open Publishing Association, 2020. `doi:10.4204/EPTCS.320.2`.

13 Lorenzo Clemente and Slawomir Lasota. Reachability analysis of first-order definable pushdown systems. In Stephan Kreutzer, editor, *Proc. of CSL'15*, volume 41 of *LIPIcs*, pages 244–259, Dagstuhl, 2015.

14 Thomas Colcombet. Forms of Determinism for Automata (Invited Talk). In Christoph Dürr and Thomas Wilke, editors, *Proc. of STACS'12*, volume 14 of *LIPIcs*, pages 1–23, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

15 Thomas Colcombet. Unambiguity in automata theory. In Jeffrey Shallit and Alexander Okhotin, editors, *Descriptional Complexity of Formal Systems*, pages 3–18, Cham, 2015. Springer International Publishing.

16 Wojciech Czerwiński, Diego Figueira, and Piotr Hofman. Universality Problem for Unambiguous VASS. In Igor Konnov and Laura Kovács, editors, *Proc. of CONCUR'20*, volume 171 of *LIPIcs*, pages 36:1–36:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

17 Laure Daviaud, Marcin Jurdzinski, Ranko Lazic, Filip Mazowiecki, Guillermo A. Pérez, and James Worrell. When is Containment Decidable for Probabilistic Automatal. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *Proc. of ICALP'18*, volume 107 of *LIPIcs*, pages 121:1–121:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

18 Stéphane Demri and Ranko Lazić. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Logic*, 10(3):16:1–16:30, April 2009.

19 Mark Giesbrecht and Myung Sub Kim. Computing the Hermite form of a matrix of Ore polynomials. *Journal of Algebra*, 376:341–362, 2013.

20 John Hopcroft, Rajeev Motwani, and Jeffrey Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 2000.

21 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.

22 Michael Kaminski and Daniel Zeitlin. Finite-memory automata with non-deterministic reassignment. *International Journal of Foundations of Computer Science*, 21(05):741–760, 2010.

23 Werner Kuich. On the multiplicity equivalence problem for context-free grammars. In *Proceedings of the Colloquium in Honor of Arto Salomaa on Results and Trends in Theoretical Computer Science*, pages 232—-250, Berlin, Heidelberg, 1994. Springer-Verlag.

24 Dugald Macpherson. A survey of homogeneous structures. *Discrete Math.*, 311(15):1599–1634, August 2011.

25 Antoine Mottet and Karin Quaas. The containment problem for unambiguous register automata and unambiguous timed automata. *Theory of Computing Systems*, 2020. `doi:10.1007/s00224-020-09997-2`.

26 A.S. Murawski, S.J. Ramsay, and N. Tzevelekos. Reachability in pushdown register automata. *Journal of Computer and System Sciences*, 87:58–83, 2017.

27 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Logic*, 5(3):403—-435, July 2004.

28 Oystein Ore. Linear equations in non-commutative fields. *Annals of Mathematics*, 32(3):463–477, 1931. URL: `http://www.jstor.org/stable/1968245`.

29 Oystein Ore. Theory of non-commutative polynomials. *Annals of Mathematics*, 34(3):480–508, 1933. URL: `http://www.jstor.org/stable/1968173`.

30 Mikhail Raskin. A Superpolynomial Lower Bound for the Size of Non-Deterministic Complement of an Unambiguous Automaton. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *Proc. of ICALP'18*, volume 107 of *LIPIcs*, pages 138:1–138:11, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

31 Arto Salomaa and Marti Soittola. *Automata-theoretic aspects of formal power series.* Texts and Monographs in Computer Science. Springer, 1978.

32 James Schmerl. A decidable $\aleph_0$-categorical theory with a non-recursive Ryll-Nardzewski function. *Fundamenta Mathematicae*, 98(2):121–125, 1978.

33 Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In Zoltán Ésik, editor, *Computer Science Logic*, volume 4207 of *LNCS*, pages 41–57. Springer Berlin Heidelberg, 2006.

34 Géraud Sénizergues. The equivalence problem for deterministic pushdown automata is decidable. In Pierpaolo Degano, Roberto Gorrieri, and Alberto Marchetti-Spaccamela, editors, *Proc. of ICALP'97*, pages 671–681, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

35 Richard P. Stanley. Differentiably finite power series. *European Journal of Combinatorics*, 1(2):175–188, 1980.

36 Richard P. Stanley. *Enumerative Combinatorics.* The Wadsworth & Brooks/Cole Mathematics Series 1. Springer, 1 edition, 1986.

37 R. Stearns and H. Hunt. On the equivalence and containment problems for unambiguous regular expressions, grammars, and automata. In *Proc. of SFCS'81*, pages 74–81, Washington, DC, USA, 1981. IEEE Computer Society. doi:10.1109/SFCS.1981.29.

38 L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time (preliminary report). In *Proc. of STOC'73*, pages 1–9, New York, NY, USA, 1973. ACM.

39 Tzeng Wen-Guey. On path equivalence of nondeterministic finite automata. *Information Processing Letters*, 58(1):43–46, 1996.

Tight Approximation Guarantees for Concave Coverage Problems

Siddharth Barman ✉
Indian Institute of Science, Bangalore, India

Omar Fawzi ✉
Univ. Lyon, ENS Lyon, UCBL, CNRS, Inria, LIP, F-69342, Lyon Cedex 07, France

Paul Fermé ✉
Univ. Lyon, ENS Lyon, UCBL, CNRS, Inria, LIP, F-69342, Lyon Cedex 07, France

Abstract

In the maximum coverage problem, we are given subsets $T_1, \dots, T_m$ of a universe $[n]$ along with an integer k and the objective is to find a subset $S \subseteq [m]$ of size k that maximizes $C(S) := \left|\bigcup_{i\in S} T_i\right|$. It is a classic result that the greedy algorithm for this problem achieves an optimal approximation ratio of $1 - e^{-1}$.

In this work we consider a generalization of this problem wherein an element a can contribute by an amount that depends on the number of times it is covered. Given a concave, nondecreasing function φ, we define $C^{\varphi}(S) := \sum_{a\in[n]} w_a \varphi(|S|_a)$, where $|S|_a = |\{i \in S : a \in T_i\}|$. The standard maximum coverage problem corresponds to taking $\varphi(j) = \min\{j, 1\}$. For any such φ, we provide an efficient algorithm that achieves an approximation ratio equal to the *Poisson concavity ratio* of φ, defined by $\alpha_{\varphi} := \min_{x\in\mathbb{N}^*} \frac{\mathbb{E}[\varphi(\mathrm{Poi}(x))]}{\varphi(\mathbb{E}[\mathrm{Poi}(x)])}$. Complementing this approximation guarantee, we establish a matching NP-hardness result when φ grows in a sublinear way.

As special cases, we improve the result of [4] about maximum multi-coverage, that was based on the unique games conjecture, and we recover the result of [11] on multi-winner approval-based voting for geometrically dominant rules. Our result goes beyond these special cases and we illustrate it with applications to distributed resource allocation problems, welfare maximization problems and approval-based voting for general rules.

2012 ACM Subject Classification Theory of computation → Packing and covering problems; Theory of computation → Rounding techniques; Theory of computation → Algorithmic game theory

Keywords and phrases Approximation Algorithms, Coverage Problems, Concave Function

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.9

Related Version *Full Version*: `https://arxiv.org/abs/2010.00970` [3]

Funding This research is supported by the French ANR project ANR-18-CE47-0011 (ACOM). SB gratefully acknowledges the support of a Ramanujan Fellowship (SERB - SB/S2/RJN-128/2015) and a Pratiksha Trust Young Investigator Award.

1 Introduction

Coverage functions are central objects of study in combinatorial optimization. Problems related to optimizing such functions arise in multiple fields, such as operations research [10], machine learning [14], algorithmic game theory [12], and information theory [2]. The most basic covering problem is the *maximum coverage* one. In this problem, we are given subsets $T_1, \dots, T_m$ of a universe $[n]$, along with a positive integer k, and the objective is to find a size-k subset $S \subseteq [m]$ that maximizes the coverage function $C(S) := \left|\bigcup_{i\in S} T_i\right|$. A fundamental result in the field of approximation algorithms establishes that an approximation ratio of $1 - e^{-1}$ can be achieved for this problem in polynomial-time [15] and, in fact, this approximation guarantee is tight, under the assumption that P ≠ NP [13].

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 9; pp. 9:1–9:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Note that in the maximum coverage problem, an element $a \in [n]$ is counted at most once in the objective, even if a appears in several selected sets. However, if we think of elements $a \in [n]$ as goods or resources, there are many settings wherein the utility indeed increases with the number of copies of a that get accumulated. Motivated, in part, by such settings, we consider a generalization of the maximum coverage problem where an element a can contribute by an amount that depends on the number of times it is covered.

Given a function $\varphi : \mathbb{N} \to \mathbb{R}_+$, an integer $k \in \mathbb{N}$, a universe of elements $[n]$, positive weights w_a for each $a \in [n]$, and subsets $T_1, \dots, T_m \subseteq [n]$, the φ-MaxCoverage problem entails maximizing $C^\varphi(S) := \sum_{a\in[n]} w_a \varphi(|S|_a)$ over subsets $S \subseteq [m]$ of cardinality k; here $|S|_a = |\{i \in S : a \in T_i\}|$.

This work focuses on functions φ that are nondecreasing and concave (i.e., $\varphi(i+2) - \varphi(i+1) \le \varphi(i+1) - \varphi(i)$ for $i \in \mathbb{N}$). We will also assume that the function φ is normalized in the sense that $\varphi(0) = 0$ and $\varphi(1) = 1$.[1] Our approximation guarantees are in terms of the *Poisson concavity ratio* of φ, which we define as follows

$$\alpha_\varphi := \inf_{x\in\mathbb{N}^*} \frac{\mathbb{E}[\varphi(\mathrm{Poi}(x))]}{\varphi(\mathbb{E}[\mathrm{Poi}(x)])} = \inf_{x\in\mathbb{N}^*} \frac{\mathbb{E}[\varphi(\mathrm{Poi}(x))]}{\varphi(x)} . \tag{1}$$

Here $\mathrm{Poi}(x)$ denotes a Poisson-distributed random variable with parameter x. We will write $\alpha_\varphi(x) := \frac{\mathbb{E}[\varphi(\mathrm{Poi}(x))]}{\varphi(x)}$, with $\alpha_\varphi(0) = 1$. One can show that $\alpha_\varphi = \min_{x\in\mathbb{N}^*} \alpha_\varphi(x) = \inf_{x\in\mathbb{R}_+} \alpha_\varphi(x)$.[2] We refer to the full version [3] for more details on the proof of this statement.

Our main result is that φ-MaxCoverage admits an efficient α_φ-approximation algorithm, when φ is normalized nondecreasing concave, and this approximation guarantee is tight when φ grows sublinearly. Formally,

▶ **Theorem 1.** For any normalized nondecreasing concave function φ, there exists a polynomial-time α_φ-approximation algorithm for the φ-MaxCoverage problem. Furthermore, for $\varphi(n) = o(n)$, it is NP-hard to approximate the φ-MaxCoverage problem within a factor better than $\alpha_\varphi + \varepsilon$, for any constant $\varepsilon > 0$.

Before detailing the proof of the theorem, we provide a few remarks and connections to related work.

1.1 Applications and related work

We can directly reduce the standard maximum coverage problem to φ-MaxCoverage by setting $\varphi(j) = \min\{j, 1\}$. In this case $\alpha_\varphi = 1 - e^{-1}$. One can also encapsulate, within our framework, the ℓ-MultiCoverage problem studied in [4] by instantiating $\varphi(j) = \min\{j, \ell\}$. In this setting, we recover the approximation ratio $\alpha_\varphi = 1 - \frac{\ell^\ell e^{-\ell}}{\ell!}$ by a simple calculation, which matches the approximation guarantee obtained in [4]. Note that the hardness result in [4] was based on the Unique Games Conjecture, whereas the current work proves that this guarantee is tight under P $\neq$ NP.

Another application of φ-MaxCoverage is in the context of multiwinner elections that entail selecting k (out of m) candidates with the objective of maximizing the cumulative utility of n voters; here, the utility of each voter $a \in [n]$ increases as more and more approved (by a) candidates get selected. One can reduce multiwinner elections to a coverage problem

[1] One can always replace a generic φ to a normalized one without changing the optimal solutions through a simple affine transformation.

[2] We require φ to be defined for nonnegative integers and will extend it over $\mathbb{R}_+$ by considering its piecewise linear extension.

by considering subset $T_i \subseteq [n]$ as the set of voters that approve of candidate $i \in [m]$ and $\varphi(j)$ as the utility that an agent achieves from j approved selections.[3] Addressing multiwinner elections in this standard utilitarian model, Dudycz et al. [11] obtain tight approximation guarantees for some well-studied classes of utilities. Specifically, the result in [11] applies to the classic *proportional approval voting rule*, which assigns a utility of $\sum_{i=1}^{j} \frac{1}{i}$ for j approved selections. This voting rule corresponds to the coverage problem with $\varphi(j) = \sum_{i=1}^{j} \frac{1}{i}$. Section 4.1 shows that Theorem 1 holds for all the settings considered in [11] and, in fact, applies more generally. In particular, the voting version of ℓ-MultiCoverage (studied in [21]) can be addressed by Theorem 1, but not by the result in [11]. Such a separation also arises when one truncates the proportional approval voting rule to, say, ℓ candidates, i.e., upon setting $\varphi(j) = \sum_{i=1}^{\min\{j,\ell\}} \frac{1}{i}$. Given that multiwinner elections model multiple real-world settings (e.g., committee selection [21] and parliamentary proceedings [6]), instantiations of φ-MaxCoverage in such social-choice contexts substantiate the applicability of our algorithmic result.

Coverage functions arise in numerous resource-allocation settings, such as sensor allocation [16], job scheduling, and plant location [10]. The goal, broadly, in such setups is to select k subsets of resources (out of m pre-specified ones) such that the *welfare* generated by the selected resources is maximized–each resource's contribution to the welfare increases with the number of times it is selected. This problem can be cast as φ-MaxCoverage by setting n to be the number of resources, $\{T_i\}_{i\in[m]}$ as the given collection of subsets, and $\varphi(j)$ to be the welfare contribution of a resource when it is covered j times.[4] Here, we mention a specific allocation problem to highlight the relevance of studying φ beyond the standard coverage and ℓ-coverage formulations (see Section 4.3 for details): in the Vehicle-Target Assignment problem [17, 19] the resources are n targets and covering a target j times contributes $\varphi^p(j) = \frac{1-(1-p)^j}{p}$ to the welfare; here, $p \in (0, 1)$ is a given parameter. Interestingly, we find that for this problem, the approximation ratio α_φ we obtain can outperform the *Price of Anarchy* (PoA), which corresponds to the approximation ratio of any algorithm where the agents selfishly maximize their utilities (see Section 4.3 for further discussion of this point). This is to be contrasted with the resource allocation problem with $\varphi(j) = \min\{j, \ell\}$ for which it was shown in [8] that the Price of Anarchy matches with α_φ.

Theorem 1 gives us a tight approximation bound of α_φ for all the above-mentioned applications of φ-MaxCoverage. The values of α_φ for these instantiations are listed in Table 1.

It is relevant to compare the approximation guarantee, α_φ, obtained in the current work with the approximation ratio based on the notion of curvature of submodular functions. Note that if φ is nondecreasing and concave, then C^φ is submodular. One can show, via a direct calculation, that for such a submodular C^φ the curvature (as defined in [9]) is given by $c = 1 - (\varphi(m) - \varphi(m-1))$ for instances with at most m cover sets. Therefore, the algorithm of Sviridenko et al. [22] provides an approximation ratio of $1 - ce^{-1}$ for the φ-MaxCoverage problem. We note that the Poisson concavity ratio α_φ is always greater than or equal to this curvature-dependent ratio (see full version [3]). Specifically, for p-Vehicle-Target Assignment, it is strictly better for all $p \notin \{0, 1\}$ and for ℓ-MultiCoverage, it is strictly better for all $\ell \geq 2$ as remarked in [4]. Therefore, for the setting at hand, the current work improves the approximation guarantee obtained in [22].

[3] Indeed, for a subset of candidates $S \subseteq [m]$, the utility of a voter $a \in [n]$ is equal to $\varphi(|S|_a)$, with $|S|_a = |\{i \in S : a \in T_i\}|$.

[4] Formally, to capture specific welfare-maximization problems in their entirety we have to a consider φ-MaxCoverage with a matroid constraint, and not just bound the number of selected subsets by k. Details pertaining to matroid constraints and the reduction appear in Section 2.2 and 4.2, respectively.

Table 1 Tight approximation ratios for particular choices of φ in the φ-MaxCoverage problem. See full version [3] for derivations of these values.

φ-MaxCoverage	$\varphi(j)$	α_φ
MaxCoverage	$\min\{j, 1\}$	$1 - e^{-1}$
ℓ-MultiCoverage	$\min\{j, \ell\}$	$1 - \frac{\ell^\ell e^{-\ell}}{\ell!}$
Proportional Approval Voting	$\sum_{i=1}^{j} \frac{1}{i}$	$\alpha_\varphi(1) \simeq 0.7965\ldots$
PAV capped at 3	$\sum_{i=1}^{\min\{j,3\}} \frac{1}{i}$	$\alpha_\varphi(1) \simeq 0.7910\ldots$
p-Vehicle-Target Assignment	$\frac{1-(1-p)^j}{p}$	$\frac{1-e^{-p}}{p}$
0.1-Vehicle-Target Assignment	$\frac{1-(1-0.1)^j}{0.1}$	$\frac{1-e^{-0.1}}{0.1} \simeq 0.9516\ldots$
0.1-VTA capped at 5	$\frac{1-(1-0.1)^{\min\{j,5\}}}{0.1}$	$\alpha_\varphi(5) \simeq 0.8470\ldots$

1.2 Remarks on the Poisson concavity ratio α_φ

By Jensen's inequality along with the nonnegativity and concavity of φ, we have that $\alpha_\varphi \in [0, 1]$. We show that α_φ can be computed numerically up to any precision $\varepsilon > 0$, in time that is polynomial in $\frac{1}{\varepsilon}$. In fact, one can show that $\alpha_\varphi(x) \geq 1 - \varepsilon$ for all $x \geq N_\varepsilon := \lceil \left(\frac{6}{\varepsilon}\right)^4 \rceil$ (see full version [3]). Thus, we can iterate over all $x \in \{1, 2, \ldots, N_\varepsilon\}$ and find $\min_{x \in [N_\varepsilon]} \alpha_\varphi(x)$ up to ε precision (under reasonable assumptions on φ). This gives us a method to overall compute α_φ, up to an absolute error of 2ε: if $\alpha_\varphi \leq 1 - \varepsilon$, then computing $\min_{x \in [N_\varepsilon]} \alpha_\varphi(x)$ (up to ε precision) suffices. Otherwise, if $\alpha_\varphi \geq 1 - \varepsilon$, then $\alpha_\varphi(1) \leq 1$ provides the desired bound. Furthermore, we note that even if we consider $\alpha_\varphi(x)$ over all $x \in \mathbb{R}_+$, an infimum (i.e., the value of α_φ) is achieved at an integer.

1.3 Proof techniques and organization

In Section 2, we present our approximation algorithm for the φ-MaxCoverage. The algorithm is an application of *pipage rounding*, a technique introduced in [1], on a linear programming relaxation of φ-MaxCoverage. We show that the multilinear extension F^φ of C^φ is efficiently computable and thus, we can compute an integer solution x^{int} from the optimal fractional one x^* satisfying $C^\varphi(x^{\text{int}}) \geq F^\varphi(x^*)$. Using the notion of *convex order* between distributions, we show that $F^\varphi(x^*) \geq \sum_{a \in [n]} w_a \mathbb{E}[\varphi(\text{Poi}(|x^*|_a))]$, where $|x|_a = \sum_{i \in [m]: a \in T_i} x_i$. Comparing this to the value $\sum_{a \in [n]} w_a \varphi(|x^*|_a)$ taken by the linear program, we get a ratio given by the *Poisson concavity ratio* α_φ. The concavity of φ is crucial at several steps of the proof: it guarantees that the natural relaxation can be written as a linear program, it is used to relate between sums of Bernouilli random variables and a Poisson random variable via the convex order, as well as for the fact that we can restrict the infimum in the definition of α_φ to integer values of x. The generalization to matroid constraints follows in a standard way and is presented in Section 2.2.

In Section 3, we present the hardness result for φ-MaxCoverage. For this, we define a generalization of the partitioning gadget of Feige [13], extending also [4]. Roughly speaking, for an integer $x_\varphi \in \mathbb{N}$, it is a collection of x_φ-covers of the set $[n]$ (an x-cover is a collection of subsets such that each element $a \in [n]$ is covered x times, or in other words, its φ-coverage is $\varphi(x)n$) that are incompatible in the sense that if we take an element from each one of these x_φ-covers, then the φ-coverage is bounded approximately by $\mathbb{E}[\varphi(\text{Poi}(x_\varphi))]n$. Then, we construct an instance of φ-MaxCoverage from an instance of the NP-hard problem *Label Cover* (as in [11]) using such a gadget with $x_\varphi \in \text{argmin}_{x \in \mathbb{N}^*} \alpha_\varphi(x)$. Having set up the partitioning gadget, the analysis of the reduction can be obtained by carefully generalizing the reductions of [4] and [11].

In Section 4, we present different domains of application of our result.

2 Approximation Algorithm for φ-MaxCoverage

Fix a function $\varphi : \mathbb{N} \to \mathbb{R}_+$ that is normalized, nondecreasing and concave. The φ-MaxCoverage problem is defined as follows. The input to the problem is given by positive integers n, m, t and m subsets $T_1, \ldots, T_m$ of the set $[n]$ (described as characteristic vectors), the weights $w_a \in \mathbb{Q}_+^*$ for $a \in [n]$ (described as a couple of bitstring of length t), as well as an integer $k \in \{1, \ldots, m\}$. The output is a subset $S \subseteq [m]$ of size exactly k that maximizes $C^\varphi(S) = \sum_{a\in[n]} w_a \varphi(|S|_a)$, where $|S|_a = |\{i \in S : a \in T_i\}|$.

Note that the input to this problem can be specified using $n(m+2t) + O(\log nmt)$ bits. To reduce the number of parameters, we will assume that t is polynomial in n and m, so that a polynomial time algorithm for this problem means an algorithm that runs in time polynomial in n and m. The counting function φ is fixed and does not depend on the instance of the problem, but for a given instance the problem only depends on the values $\varphi(0), \varphi(1), \ldots, \varphi(m)$. We assume that we have black box access to φ and to ensure that all the algorithms run in polynomial time, we assume that $\varphi(j)$ can be described with a number of bits that is polynomial in j and that this description can be computed in polynomial time.

We now describe the approximation algorithm for φ-MaxCoverage that we analyze. As described above, we follow the standard relax and round strategy, as in [4]. First, we define a natural convex relaxation.

▶ **Definition 2.1** (Relaxed program).

$$\begin{aligned}
\text{maximize} \quad & \sum_{a\in[n]} w_a c_a \\
\text{subject to} \quad & c_a \le \varphi(|x|_a), \forall a \in [n], \ \text{with } |x|_a := \sum_{i\in[m]:a\in T_i} x_i \\
& 0 \le x_i \le 1, \forall i \in [m] \\
& \sum_{i=1}^{m} x_i = k \ .
\end{aligned} \tag{2}$$

As previously mentioned, φ is defined on $\mathbb{R}_+$ by extending it in a piecewise linear fashion on non-integral points. As such, the constraint $c_a \le \varphi(|x|_a)$ is equivalent to m linear constraints. In fact, we can define φ_j to be the linear function $\varphi_j(t) = (\varphi(j) - \varphi(j-1))t - (j-1)\varphi(j) + j\varphi(j-1)$ for $j \in [m]$. Since φ is concave, we have that for all $t \in [0, m]$, $\varphi(t) = \min_{j\in[m]} \varphi_j(t)$. As such, the constraint $c_a \le \varphi(|x|_a)$ is equivalent to $c_a \le \varphi_j(|x|_a)$ for all $j \in [m]$ and so the program from Definition 2.1 is a linear program. Overall there are $n + m$ variables and $(n+1)m + 1$ linear constraints, and by assumptions all the coefficients can be described using a number of bits that is polynomial in n and m. Hence an optimal solution of this linear program can be found in polynomial time.

Also observe that the program from Definition 2.1 is a relaxation of the φ-MaxCoverage problem. To see this, given a set S of size k, consider the characteristic vector $x \in \{0,1\}^m$ defined by $x_i = 1$ if and only if $i \in S$. Then for all $a \in [n]$, we can set $c_a = \varphi(|x|_a) = \varphi(|S|_a)$, and we get an objective value of $\sum_{a\in[n]} w_a \varphi(|S|_a)$ which is exactly $C^\varphi(S)$. When solving the program from Definition 2.1, we get an optimal $x^* \in [0,1]^m$ which is in general not integral. Next, we describe a method to round it to an integral vector $x^{\text{int}} \in \{0,1\}^m$.

2.1 Rounding

For a submodular function $f : \{0,1\}^m \to \mathbb{R}$, one can use pipage rounding [1, 23, 7] to transform, in polynomial time, any fractional solution $x \in [0,1]^m$ satisfying $\sum_{i=1}^m x_i = k$ into an integral vector $x^{\text{int}} \in \{0,1\}^m$ such that $\sum_{i=1}^m x_i^{\text{int}} = k$ and $F(x^{\text{int}}) \ge F(x)$, where F

corresponds to the multilinear extension of f, provided that $F(x)$ is computable in polynomial time for a given x; see e.g., [23, Lemma 3.4]. The multilinear extension $F : [0,1]^m \to \mathbb{R}$ of f is defined by $F(x_1, \dots, x_m) := \mathbb{E}[f(X_1, \dots, X_m)]$, where X_i are independent random variables with $X_i \sim \operatorname{Ber}(x_i)$, i.e., $X_i \in \{0,1\}$ with $\mathbb{P}(X_i = 1) = x_i$. Note that $F(x) = f(x)$ for an integral vector $x \in \{0,1\}^m$.

We apply this strategy to C^φ, which is submodular, and the solution x^* of the LP relaxation from Definition 2.1. Note that overall the algorithm is polynomial time, since here $F(x)$ is computable in polynomial time for a given x:

▶ **Proposition 2.2** ([3]). Let $F(x) := \mathbb{E}_{X\sim x}[C^\varphi(X)]$ for $x \in \{0,1\}^m$. We have an explicit formula for F:

$$F(x) = \sum_{a=1}^{n}\sum_{k=0}^{m}\Big[\frac{1}{m+1}\sum_{\ell=0}^{m}\omega_{m+1}^{-\ell k}\prod_{j\in[m]:a\in T_j}(1+(\omega_{m+1}^{\ell}-1)x_j)\Big]\varphi(k) \text{ with } \omega_{m+1} := e^{\frac{2i\pi}{m+1}}$$

Thus, F is computable in polynomial time in n and m.

We now analyze the value returned by the algorithm. Using the property of pipage rounding, with the notation $X = (X_1, \dots, X_m)$ and $\operatorname{Ber}(x) = (\operatorname{Ber}(x_1), \dots, \operatorname{Ber}(x_m))$, we get

$$C^\varphi(x^{\text{int}}) = \mathbb{E}_{X\sim\operatorname{Ber}(x^{\text{int}})}[C^\varphi(X)] \geq \mathbb{E}_{X\sim\operatorname{Ber}(x^*)}[C^\varphi(X)] \ .$$

Then it suffices to relate $\mathbb{E}_{X\sim\operatorname{Ber}(x^*)}[C^\varphi(X)]$ to the optimal value of the LP relaxation 2.1, which can only be larger than the optimal value of the φ-MaxCoverage problem.

▶ **Theorem 2.** Let x, c be a feasible solution of the program from Definition 2.1 and $X \sim \operatorname{Ber}(x)$. Recalling the definition of α_φ and $\alpha_\varphi(j)$ from (1), we have

$$\mathbb{E}_{X\sim\operatorname{Ber}(x)}[C^\varphi(X)] \geq \left(\min_{j\in[m]}\alpha_\varphi(j)\right)\sum_{a\in[n]} w_a c_a$$

In particular, this implies that the described polynomial time algorithm has an approximation ratio of α_φ:

$$C^\varphi(x^{\text{int}}) \geq \alpha_\varphi \sum_{a\in[n]} w_a c_a^* \geq \alpha_\varphi \max_{S\subseteq[m]:|S|=k} C^\varphi(S) \ .$$

In order to prove this theorem, we need the following lemma:

▶ **Lemma 2.3.** For φ concave, and $p \in [0,1]^m$, we have:

$$\mathbb{E}\Big[\varphi\Big(\sum_{i=1}^{m}\operatorname{Ber}(p_i)\Big)\Big] \geq \mathbb{E}\Big[\varphi\Big(\operatorname{Poi}\Big(\sum_{i=1}^{m}p_i\Big)\Big)\Big]$$

Proof. The notion of *convex order* discussed in [20] allows us to prove this result. We say that $X \leq_{\text{cx}} Y \iff \mathbb{E}[f(X)] \leq \mathbb{E}[f(Y)]$ for any convex f. Thanks to Lemma 2.3 of [4], we have that for $p \in [0,1]$:

$$\operatorname{Ber}(p) \leq_{\text{cx}} \operatorname{Poi}(p)$$

Since this order is preserved through convolution (Theorem 3.A.12 of [20]), and the fact that $\sum_{i=1}^m \mathrm{Poi}(p_i) \sim \mathrm{Poi}\left(\sum_{i=1}^m p_i\right)$, we have:

$$\sum_{i=1}^m \mathrm{Ber}(p_i) \leq_{\mathrm{cx}} \mathrm{Poi}\left(\sum_{i=1}^m p_i\right)$$

Applying this result to $-\varphi$, which is convex, concludes the proof. ◀

We will also use the following property on $\alpha_\varphi(x)$:

▶ **Proposition 2.4** ([3]). For all $x \in \mathbb{R}_+$, we have $\alpha_\varphi(x) \geq \min\{\alpha_\varphi(\lfloor x \rfloor), \alpha_\varphi(\lceil x \rceil)\}$; here, $\alpha_\varphi(0) := \lim_{x\to 0} \alpha_\varphi(x) = 1$.

Proof of Theorem 2. By linearity of expectation and the fact that the weights w_a are positive, it is sufficient to show that for all $a \in [n]$:

$$\mathbb{E}[C_a^\varphi(X)] \geq \left(\min_{j\in[m]} \alpha_\varphi(j)\right) c_a \ ,$$

where $C_a^\varphi(S) := \varphi(|S|_a)$. Note that $|X|_a = \sum_{i\in[m]:a\in T_i} X_i$, and thus:

$$\begin{aligned}
\mathbb{E}[C_a^\varphi(X)] = \ & \mathbb{E}\Big[\varphi\Big(\sum_{i\in[m]:a\in T_i} X_i\Big)\Big] = \mathbb{E}\Big[\varphi\Big(\sum_{i\in[m]:a\in T_i} \mathrm{Ber}(x_i)\Big)\Big] \\
\geq \ & \mathbb{E}\Big[\varphi\Big(\mathrm{Poi}\Big(\sum_{i\in[m]:a\in T_i} x_i\Big)\Big)\Big] \text{ thanks to Lemma 2.3} \\
= \ & \mathbb{E}[\varphi(\mathrm{Poi}(|x|_a))] \geq \min\{\alpha_\varphi(\lfloor |x|_a \rfloor), \alpha_\varphi(\lceil |x|_a \rceil)\}\varphi(|x|_a) \quad \text{by Proposition 2.4} \\
\geq \ & \left(\min_{j\in[m]} \alpha_\varphi(j)\right) \varphi(|x|_a) \geq \left(\min_{j\in[m]} \alpha_\varphi(j)\right) c_a \ .
\end{aligned} \tag{3}$$

◀

2.2 Generalization to Matroid Constraints

Instead of taking a cardinality constraint k on the size of the subset S, we look now at general matroid constraints on S. Specifically, as input, instead of k, we take a matroid $\mathcal{M}$ defined on $[m]$ and given by a set of linear constraints describing its base polytope $B(\mathcal{M})$. The output is a set $S \in \mathcal{M}$ that maximizes $C^\varphi(S)$. Note that the cardinality constraint considered above is the special case where $\mathcal{M}$ is the uniform matroid of all subsets of size at most k and the base polytope $B(\mathcal{M}) = \{x \in [0,1]^m : \sum_{i=1}^m x_i = k\}$.

We first note that in the order to establish Theorem 2, the cardinality constraint $\sum_{i=1}^m x_i = k$ is not used. Thus, since the pipage rounding strategy applies to matroid constraints $\mathcal{M}$ (see [23, Lemma 3.4]), the strategy and the analysis of its efficiency generalize immediately when applied to the following linear program:

▶ **Definition 2.5** (Relaxed program for matroid constraints).

$$\begin{aligned}
\text{maximize} \quad & \sum_{a\in[n]} w_a c_a \\
\text{subject to} \quad & c_a \leq \varphi(|x|_a), \forall a \in [n] \\
& 0 \leq x_i \leq 1, \forall i \in [m] \\
& x \in B(\mathcal{M}) \quad \text{the base polytope of } \mathcal{M} \ .
\end{aligned} \tag{4}$$

▶ **Theorem 3.** Let x, c a feasible solution of the program from Definition 2.5 and $X \sim \mathrm{Ber}(x)$. Then:

$$\mathbb{E}_{X\sim \mathrm{Ber}(x)}[C^{\varphi}(X)] \geq \left(\min_{j\in[m]} \alpha_{\varphi}(j)\right) \sum_{a\in[n]} w_a c_a \ .$$

In particular, this implies that the described polynomial time algorithm has an approximation ratio of α_{φ}:

$$C^{\varphi}(x^{\mathrm{int}}) \geq \alpha_{\varphi} \sum_{a\in[n]} w_a c^*_a \geq \alpha_{\varphi} \max_{S\in\mathcal{M}} C^{\varphi}(S) \ .$$

3 Hardness of Approximation for φ-MaxCoverage

In this section, we establish an inapproximability bound for the φ-MaxCoverage problem with weights 1 under cardinality constraints. Throughout this section we use Γ to denote the universe of elements and, hence, an instance of the φ-MaxCoverage problem consists of Γ, along with a collection of subsets $\mathcal{F} = \{F_i \subseteq \Gamma\}_{i=1}^{m}$ and an integer k. Recall that the objective of this problem is to find a size-k subset $S \subseteq [m]$ that maximizes $C^{\varphi}(S) = \sum_{a\in\Gamma} \varphi(|S|_a)$.

We establish the following theorem in this section:

▶ **Theorem 4.** It is NP-hard to approximate the φ-MaxCoverage problem for $\varphi(n) = o(n)$ within a factor greater that $\alpha_{\varphi} + \varepsilon$ for any $\varepsilon > 0$.

Our reduction is based on a problem called h-AryLabelCover, which is equivalent to the more standard GapLabelCover problem.

▶ **Definition 3.1** (h-AryLabelCover)**.** An instance $\mathcal{G} = (V, E, [L], [R], \{\pi_{e,v}\}_{e\in E, v\in e})$ of h-AryLabelCover is characterized by an h-uniform regular hypergraph (V, E) and bijection constraints $\pi_{e,v} : [L] \to [R]$. Here, each h-uniform hyperedge represents a h-ary constraint. Additionally, for any labeling $\sigma : V \to [L]$, we have the following notions of strongly and weakly satisfied constraints:

- An edge $e = (v_1, \ldots, v_h) \in E$ is *strongly satisfied* by σ if:

$$\forall x, y \in [h], \pi_{e,v_x}(\sigma(v_x)) = \pi_{e,v_y}(\sigma(v_y))$$

- An edge $e = (v_1, \ldots, v_h) \in E$ is *weakly satisfied* by σ if:

$$\exists x \neq y \in [h], \pi_{e,v_x}(\sigma(v_x)) = \pi_{e,v_y}(\sigma(v_y))$$

▶ **Proposition 3.2** (δ, h-AryGapLabelCover - [3])**.** For any fixed integer $h \geq 2$ and fixed $\delta > 0$, there exists an R_0 such that for any integer $R \geq R_0$, it is NP-hard for instances $\mathcal{G} = (V, E, [L], [R], \{\pi_{e,v}\}_{e\in E, v\in e})$ of h-AryLabelCover with right alphabet $[R]$ to distinguish between:

- **YES:** There exists a labeling σ that *strongly satisfies* all the edges.
- **NO:** No labeling *weakly satisfies* more than δ fraction of the edges.

3.1 Partitioning System

The key ingredient to prove Theorem 4 is a constant size combinatorial object called partitioning system, generalizing the work of Feige [13] and [4]. For any set $[n]$, $\mathcal{Q} \subseteq 2^{[n]}$, we overload the definition $C^{\varphi}(\mathcal{Q}) := \sum_{a\in[n]} \varphi(|\mathcal{Q}|_a)$ with $|\mathcal{Q}|_a := |\{P \in \mathcal{Q} : a \in P\}|$ and $C^{\varphi}_a(\mathcal{Q}) := \varphi(|\mathcal{Q}|_a)$. Let us take $x_{\varphi} \in \operatorname{argmin}_{x\in\mathbb{N}^*} \alpha_{\varphi}(x)$, thus $\alpha_{\varphi} = \alpha_{\varphi}(x_{\varphi})$.

We say that $\mathcal{Q}$ is an *x-cover* of $x \in \mathbb{N}$ if every element of $[n]$ is covered x times, so $C^{\varphi}(\mathcal{Q}) = n\varphi(x)$.

▶ **Definition 3.3.** An $([n], h, R, \varphi, \eta)$-*partitioning system* consists of R collections of subsets of $[n]$, $\mathcal{P}_1, \ldots, \mathcal{P}_R \subseteq 2^{[n]}$, that satisfy $\frac{x_\varphi n}{h} \in \mathbb{N}$, $x_\varphi \geq h$ and:

1. For every $i \in [R], \mathcal{P}_i$ is a collection of h subsets $P_{i,1}, \ldots, P_{i,h} \subseteq [n]$ each of size $\frac{x_\varphi n}{h}$ which is an x_φ-cover.
2. For any $T \subseteq [R]$ and $\mathcal{Q} = \{P_{i,j(i)} : i \in T\}$ for some function $j : T \to [h]$, we have $\left|C^\varphi(\mathcal{Q}) - \psi^\varphi_{|T|,h} n\right| \leq \eta n$ where:

$$\psi^\varphi_{k,h} := \mathbb{E}\left[\varphi\left(\operatorname{Bin}\left(k, \frac{x_\varphi}{h}\right)\right)\right] . \tag{5}$$

▶ Remark. In particular, for any $\mathcal{Q} = \{Q_1, \ldots, Q_k\}$ with Q_i of size $\frac{x_\varphi n}{h}$, we have that $C^\varphi(\mathcal{Q}) \leq n\varphi(k\frac{x_\varphi}{h})$. Indeed $C^\varphi(\mathcal{Q}) = \sum_{a\in[n]} \varphi(|\mathcal{Q}|_a)$ with $\sum_{a\in[n]} |\mathcal{Q}|_a = \sum_{i\in[k]} |Q_i| = k \cdot \frac{x_\varphi n}{h}$. By concavity of φ and Jensen's inequality, this function is maximized when all $|\mathcal{Q}|_a$ are equals, where we get $n\varphi(k\frac{x_\varphi}{h})$.

▶ **Proposition 3.4** ([3]). For $R, h \in \mathbb{N}$ with $h \geq x_\varphi$, $\eta \in (0, 1)$, $n \geq \eta^{-2} R\varphi(R)^2 \log(20(h + 1))$ such that $\frac{x_\varphi n}{h} \in \mathbb{N}$, there exists an $([n], h, R, \varphi, \eta)$-partitioning system, which can be found in time $\exp(Rn \log(n))\cdot\operatorname{poly}(h)$.

3.2 The Reduction

Proof of Theorem 4. Let $\varepsilon > 0$. Without loss of generality, we can assume that $\varepsilon < 1$. We show that it is NP-hard to reach an approximation greater than $\alpha_\varphi + \varepsilon$ for the φ-MaxCoverage problem, via a reduction from δ, h-AryGapLabelCover. Define:

- $\eta = \frac{\varphi(x_\varphi)}{4x_\varphi}\varepsilon$, so $0 < \eta \leq \varepsilon < 1$,
- $h \geq x_\varphi$ such that $\left|\psi^\varphi_{h,h} - \alpha_\varphi\varphi(x_\varphi)\right| \leq \eta$ (see (5) for the definition of ψ^φ); one can show that such a choice exists by bounding the total variation between Bernouilli and Poisson laws, together with the fact that $\varphi(x) = o(x)$ (see full version [3]),
- θ such that for all $x \geq \theta$, $\frac{\varphi(x)}{x} \leq \eta$, which exists since $\varphi(x) = o(x)$,
- $\xi = \frac{x_\varphi}{\theta}$,
- $\delta = \frac{\eta}{2}\frac{\xi^3}{h^2}$,
- $R \geq h$ large enough for Proposition 3.2 to hold.

Given an instance $\mathcal{G} = (V, E, [L], [R], \Sigma, \{\pi_{e,v}\}_{e\in E, v\in e})$ of δ, h-AryGapLabelCover, we construct an instance $(\Gamma, \mathcal{F}, k)$ of the φ-MaxCoverage problem with:

- n a large enough integer to have an $([n], h, R, \varphi, \eta)$-partitioning system (Proposition 3.4),
- $\Gamma = [n] \times E$,
- $k = |V|$,
- Consider a $([n], h, R, \varphi, \eta)$-partitioning system, and call $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_R\}$ the corresponding set of collections. Define sets $T^{e,v_j}_\beta = P_{\pi_{e,v_j}(\beta),j} \times \{e\}$ for $e = (v_1, \ldots, v_h) \in E, j \in [h], \beta \in [L]$. Then, choose as cover sets $F^v_\beta := \bigsqcup_{e\in E: v\in e} T^{e,v}_\beta$ and take $\mathcal{F} := \{F^v_\beta, v \in V, \beta \in [L]\}$.

We will now prove that if we are in a YES instance, we have that there exists $\mathcal{T}$ of size k such that $C^\varphi(\mathcal{T}) \geq \varphi(x_\varphi)|\Gamma|$ (completeness). Moreover, if we are in a NO instance, then we have that for all $\mathcal{T}$ of size $k = |V|$, $C^\varphi(\mathcal{T}) \leq (\alpha_\varphi + \varepsilon)\varphi(x_\varphi)|\Gamma|$ (soundness). Establishing these two properties will conclude the proof.

In order to achieve this, let us define $C^{\varphi,e} := \sum_{a\in[n]\times\{e\}} C^\varphi_a$. In particular, $C^\varphi = \sum_{a\in\Gamma} C^\varphi_a = \sum_{e\in E} C^{\varphi,e}$. For $\mathcal{T} \subseteq \mathcal{F}$, we define the relevant part of $\mathcal{T}$ on e by:

$$\mathcal{T}_e := \{T^{e,v}_\beta : v \in e, \beta \in [L], F^v_\beta \in \mathcal{T}\} = \{F^v_\beta \cap ([n] \times \{e\}), F^v_\beta \in \mathcal{T}\}.$$

Note that $C^{\varphi,e}(\mathcal{T}) = C^{\varphi,e}(\mathcal{T}_e)$, and in particular $C^\varphi(\mathcal{T}) = \sum_{e\in E} C^{\varphi,e}(\mathcal{T}_e)$.

3.3 Completeness

Suppose the given h-AryLabelCover instance $\mathcal{G}$ is a YES instance. Then, there exists a labeling $\sigma : V \mapsto [L]$ which strongly satisfies all edges. Consider the collection of $|V|$ subsets $\mathcal{T} := \{F^v_{\sigma(v)} : v \in V\}$. Fix $e = (v_1, \dots, v_h) \in E$. Since e is strongly satisfied by σ, there exists $r \in [R]$ such that $\pi_{e,v_i}(\sigma(v_i)) = r$ for all $i \in [h]$. Thus, $\mathcal{T}_e = \{T^{e,v_i}_{\sigma(v_i)}\}_{i\in[h]} = \{P_{r,i} \times \{e\}\}_{i\in[h]}$ is an x_φ-cover of $[n] \times \{e\}$, and so $C^{\varphi,e}(\mathcal{T}_e) = n\varphi(x_\varphi)$. Thus $C^\varphi(\mathcal{T}) = \sum_{e\in E} C^{\varphi,e}(\mathcal{T}_e) = |E|\varphi(x_\varphi)n = \varphi(x_\varphi)|\Gamma|$.

3.4 Soundness

Suppose the given h-AryLabelCover instance $\mathcal{G}$ is a NO instance. Let us prove the contrapositive of the soundness: we suppose that there exists $\mathcal{T}$ of size $k = |V|$ such that $C^\varphi(\mathcal{T}) > (\alpha_\varphi + \varepsilon)\varphi(x_\varphi)|\Gamma|$. Let us show that there exists a labelling σ that weakly satisfies a strictly larger fraction of the edges than δ.

For every vertex $v \in V$, we define $L(v) := \{\beta \in [L] : F^v_\beta \in \mathcal{T}\}$ to be the candidate set of labels that can be associated with the vertex v. We extend this definition to hyperedges $e = (v_1, \dots, v_h)$ where we define $L(e) := \bigcup_{i\in[h]} L(v_i)$ to be the *multiset* of all labels associated with the edge. Note that $|\mathcal{T}_e| = |L(e)|$.

We say that $e = (v_1, \dots, v_h) \in E$ is *consistent* if and only if $\exists x \neq y \in [h], \pi_{e,v_x}(L(v_x)) \cap \pi_{e,v_y}(L(v_y)) \neq \emptyset$. We then decompose E in three parts:

- B is the set of edges $e \in E$ with $|L(e)| \geq \frac{h}{\xi}$.
- N is the set of consistent edges $e \in E$ with $|L(e)| < \frac{h}{\xi}$.
- $I = E - (B \cup N)$ is the set of inconsistent edges $e \in E$ with $|L(e)| < \frac{h}{\xi}$.

We want to show that the contribution of N is not too small, which we will use to construct a labelling weakly satisfying enough edges. This comes from the following lemmas:

▶ **Lemma 3.5.** $\sum_{e\in E} |L(e)| = |E|h$

Proof. Recall that our h-uniform hypergraph is regular; call d its regular degree. In particular, we have that $d|V| = |E|h$. Note also that $\sum_{v\in V} |L(v)| = |\mathcal{T}| = |V|$. Thus:

$$\sum_{e\in E} |L(e)| = \sum_{e\in E} \sum_{v\in V : v\in e} |L(v)| = \sum_{v\in V} \sum_{e\in E : v\in e} |L(v)| = d \sum_{v\in V} |L(v)| = d|V| = |E|h\ . \tag{6}$$

◀

Next, we bound the contribution of B:

▶ **Lemma 3.6.** $\sum_{e\in B} C^{\varphi,e}(\mathcal{T}_e) \leq \frac{\varepsilon}{4}\varphi(x_\varphi)|\Gamma|$.

Proof. We have:

$$\begin{aligned}
\sum_{e\in B} C^{\varphi,e}(\mathcal{T}_e) \leq & \sum_{e\in B} n\varphi\Big(|L(e)|\frac{x_\varphi}{h}\Big) \quad \text{by the remark on Definition 3.3 and } |\mathcal{T}_e| = |L(e)| \\
\leq & \ |B| \cdot n\varphi\Big(\frac{\sum_{e\in B} |L(e)|}{|B|}\frac{x_\varphi}{h}\Big) \quad \text{by Jensen's inequality on concave } \varphi \\
\leq & \ |B| \cdot n\varphi\Big(\frac{|E|h}{|B|}\frac{x_\varphi}{h}\Big) \quad \text{since } \varphi \text{ nondecreasing and } \sum_{e\in B} |L(e)| \leq |E|h \\
= & \ \frac{\varphi\big(\frac{|E|x_\varphi}{|B|}\big)}{\frac{|E|x_\varphi}{|B|}} x_\varphi|\Gamma|\ .
\end{aligned}$$

(7)

We have seen that $\sum_{e\in B}|L(e)| \le |E|h$, but $\sum_{e\in B}|L(e)| \ge |B|\frac{h}{\xi}$ by definition of B, so we have that $\frac{|B|}{|E|} \le \xi$. Thus $\frac{|E|x_\varphi}{|B|} \ge \frac{x_\varphi}{\xi} = \theta$. By definition of θ, we get that $\sum_{e\in B} C^{\varphi,e}(\mathcal{T}_e) \le \eta x_\varphi|\Gamma| = \frac{\varepsilon}{4}\varphi(x_\varphi)|\Gamma|$. ◀

In order to bound the contribution of I, we will prove a property on inconsistent edges:

▶ **Proposition 3.7.** Let $e = (v_1, \dots, v_h) \in E$ be an inconsistent hyperedge with respect to $\mathcal{T}$. Then we have that $\left|C^{\varphi,e}(\mathcal{T}_e) - \psi^{\varphi}_{|L(e)|,h}n\right| \le \eta n$.

Proof. Since e is inconsistent, $\forall x \neq y \in [h], \pi_{e,v_x}(L(v_x)) \cap \pi_{e,v_y}(L(v_y)) = \emptyset$. Therefore, for every $i \in [R]$, there is at most one $v \in e$ such that $i \in \pi_{e,v}(L(v))$, i.e., $\mathcal{T}_e$ intersects with $\mathcal{P}_i \times \{e\}$ in at most one subset. This gives us a subset $T \subseteq [R]$ and a function $j : T \to [h]$ such that $\mathcal{T}_e = \{P_{i,j(i)} \times \{e\} : i \in T\}$. As a consequence, $|T| = |\mathcal{T}_e| = |L(e)|$ and by the second condition of the partitioning system, we get the expected result. ◀

Now, we can bound the contribution of I:

▶ **Lemma 3.8.** $\sum_{e\in I} C^{\varphi,e}(\mathcal{T}_e) \le (\alpha_\varphi + \frac{\varepsilon}{2})\varphi(x_\varphi)|\Gamma|$.

Proof. Thanks to Proposition 3.7, we have:

$$\sum_{e\in I} C^{\varphi,e}(\mathcal{T}_e) \le \sum_{e\in I}(\psi^{\varphi}_{|L(e)|,h} + \eta)n \le \sum_{e\in E}(\psi^{\varphi}_{|L(e)|,h} + \eta)n \ , \tag{8}$$

since $I \subseteq E$ and $\psi^{\varphi}_{|L(e)|,h} \ge 0$. But $\sum_{e\in E}|L(e)| = |E|h$ by Lemma 3.5, and one can show that $x \mapsto \psi^{\varphi}_{x,h}$ is concave (see full version [3]), so we can use Jensen's inequality to get $\sum_{e\in E}\psi^{\varphi}_{|L(e)|,h} \le |E|\psi^{\varphi}_{\frac{\sum_{e\in E}|L(e)|}{|E|},h} = |E|\psi^{\varphi}_{h,h}$ and thus:

$$\sum_{e\in I} C^{\varphi,e}(\mathcal{T}_e) \le (\psi^{\varphi}_{h,h} + \eta)n|E| \le (\alpha_\varphi\varphi(x_\varphi) + 2\eta)|\Gamma| \ , \tag{9}$$

by definition of h. This implies that the total contribution of inconsistent edges I is at most $\sum_{e\in I} C^{\varphi,e}(\mathcal{T}_e) \le (\alpha_\varphi\varphi(x_\varphi) + 2\eta)|\Gamma| \le (\alpha_\varphi + \frac{\varepsilon}{2})\varphi(x_\varphi)|\Gamma|$ by definition of η. ◀

▶ **Lemma 3.9.** $\sum_{e\in N} C^{\varphi,e}(\mathcal{T}_e) > \frac{\varepsilon}{4}\varphi(x_\varphi)|\Gamma|$ and thus $\frac{|N|}{|E|} \ge \xi\eta$.

Proof. Since we have supposed that $\sum_{e\in E} C^{\varphi,e}(\mathcal{T}_e) = C^{\varphi}(\mathcal{T}) > (\alpha_\varphi + \varepsilon)\varphi(x_\varphi)|\Gamma|$, and with the help of Lemmas 3.6 and 3.8, we have that the contribution of N is:

$$\sum_{e\in N} C^{\varphi,e}(\mathcal{T}_e) > \frac{\varepsilon}{4}\varphi(x_\varphi)|\Gamma| \ .$$

However, we have that for $e \in N$ that $C^{\varphi,e}(\mathcal{T}_e) \le n\varphi\left(|\mathcal{T}_e|\frac{x_\varphi}{h}\right) = n\varphi\left(|L(e)|\frac{x_\varphi}{h}\right) \le n\varphi\left(\frac{x_\varphi}{\xi}\right) \le \frac{nx_\varphi}{\xi}$ thanks to the remark on Definition 3.3 and the bound $|L(e)| < \frac{h}{\xi}$. This implies that:

$$\frac{|N|}{|E|} \ge \frac{\xi}{x_\varphi}\frac{\varepsilon\varphi(x_\varphi)}{4} = \xi\eta \ .$$ ◀

From this, we construct a randomized labeling $\sigma : V \mapsto [L]$ as follows: for $v \in V$, if $L(v) \neq \emptyset$, set $\sigma(v)$ uniformly from $L(v)$, otherwise set it arbitrarily. We claim that in expectation, this labeling must weakly satisfy δ fraction of the hyperedges.

To see this, fix any $e = (v_1, \dots, v_h) \in N$. Thus $\exists x \neq y \in [h], \pi_{e,v_x}(L(v_x)) \cap \pi_{e,v_y}(L(v_y)) \neq \emptyset$. Furthermore $|L(v_x)|, |L(v_y)| \leq \frac{h}{\xi}$. Thus, we have that $\pi_{e,v_x}(L(v_x)) = \pi_{e,v_y}(L(v_y))$ with probability at least $\frac{1}{|L(v_x)||L(v_y)|} \geq \left(\frac{\xi}{h}\right)^2$.

Therefore:

$$\begin{aligned} & \mathbb{E}_\sigma \mathbb{E}_{e\sim E}[\sigma \text{ weakly satisfies } e] \\ \geq \; & \xi\eta \mathbb{E}_\sigma \mathbb{E}_{e\sim E}[\sigma \text{ weakly satisfies } e | e \in N] \quad \text{by Lemma 3.9} \\ > \; & \frac{\eta}{2}\frac{\xi^3}{h^2} = \delta \, . \end{aligned} \tag{10}$$

In particular there exists some labeling σ such that $\mathbb{E}_{e\sim E}[\sigma \text{ weakly satisfies } e] > \delta$, and thus the soundness is also proved. ◀

4 Applications

This section show that instantiations of φ-MaxCoverage encapsulate and generalize multiple problems from fields such as computational social choice [5] and algorithmic game theory [18].

4.1 Multiwinner Elections

As mentioned previously, multiwinner elections (with a utilitarian model for the voters) entail selection of k (out of m) candidates that maximize the utility across n voters. Here, the utility of each voter $a \in [n]$ increases with the number of approved (by a) selections. The work of Dudycz et al. [11] study the computational complexity of such elections and, in particular, address classic voting rules in which – for a specified sequence of nonnegative weights $(w_1, w_2, \dots)$ – voter a's utility is equal to $\sum_{i=1}^{j} w_i$, when she approves of j candidates among the selected ones. One can view this election exercise as a coverage problem by considering subset $T_i \subseteq [n]$ as the set of voters that approve of candidate $i \in [m]$ and $\varphi(j) = \sum_{i=1}^{j} w_i$. Indeed, for a subset of candidates $S \subseteq [m]$, the utility of a voter $a \in [n]$ is equal to $\varphi(|S|_a)$, with $|S|_a = |\{i \in S : a \in T_i\}|$.

Dudycz et al. [11] show that if the weights satisfy $w_1 \geq w_2 \geq \dots$ (i.e., bear a diminishing returns property) along with geometric dominance ($w_i \cdot w_{i+2} \geq w_{i+1}^2$ for all $i \in \mathbb{N}^*$) and $\lim_{i\to\infty} w_i = 0$, then a tight approximation guarantee can be obtained for the election problem at hand. Note that the diminishing returns property implies that $\varphi(j) = \sum_{i=1}^{j} w_i$ is concave and $\lim_{i\to\infty} w_i = 0$ ensures that φ is sublinear. Furthermore, one can show that:

▶ **Proposition 4.1** ([3]). If $w_i := \varphi(i) - \varphi(i-1)$ is *geometrically dominant*, ie. $\forall i \in \mathbb{N}^*$, $\frac{w_i}{w_{i+1}} \geq \frac{w_{i+1}}{w_{i+2}}$, then $\alpha_\varphi = \alpha_\varphi(1)$.

Hence, Theorem 1, together with Proposition 4.1, can be invoked to recover the result in [11] where we get $\alpha_\varphi = \alpha_\varphi(1)$. In fact, Theorem 1 does not require geometric dominance among the weights and, hence, applies to a broader class of voting rules. For instance, the geometric dominance property does not hold if one considers the voting weights induced by ℓ-MultiCoverage, i.e., $w_i = 1$, for $1 \leq i \leq \ell$, and $w_j = 0$ for $j > \ell$. However, using Theorem 1, we get that for this voting rule we can approximate the optimal utility within a factor of $\alpha_\varphi = 1 - \frac{\ell^\ell e^{-\ell}}{\ell!}$. Another example of such a separation arises if one truncates the proportional approval voting. The standard proportional approval voting corresponds to $w_i = \frac{1}{i}$, for all $i \in \mathbb{N}$ (equivalently, $\varphi(j) = \sum_{i=1}^{j} \frac{1}{i}$) and falls within the purview of [11]. While the truncated version with $\varphi(j) = \sum_{i=1}^{\min\{j,\ell\}} \frac{1}{i}$, for a given threshold ℓ, does not satisfy geometric dominance, Theorem 1 continues to hold and provide a tight approximation ratio that can be computed numerically (see Table 1 for examples):

▶ **Proposition 4.2** ([3]). If $\forall x \geq \ell, \varphi(x) = \varphi(\ell) > 0$, then $\alpha_\varphi(x)$ is nondecreasing from ℓ to $+\infty$ and $\alpha_\varphi(x) = \frac{\varphi(\ell) - e^{-x} \sum_{k=0}^{\ell-1} (\varphi(\ell) - \varphi(k)) \frac{x^k}{k!}}{\varphi(x)}$. In particular, $\alpha_\varphi = \min_{x \in [\ell]} \alpha_\varphi(x)$, and the argmin can be computed numerically.

4.2 Resource Allocation in Multiagent Systems

A significant body of prior work in algorithmic game theory has addressed game-theoretic aspects of maximizing welfare among multiple (strategic) agents; see, e.g., [19]. Complementing such results, this section shows that the optimization problem underlying multiple welfare-maximization games can be expressed in terms of φ-MaxCoverage.

Specifically, consider a setting with n resources, k agents, and a (counting) function $\varphi : \mathbb{N} \mapsto \mathbb{R}_+$. Every agent i is endowed with a collection of resource subsets $\mathcal{A}_i = \{T_1^i, \ldots, T_{m_i}^i\} \subseteq 2^{[n]}$ (i.e., each $T_j^i \subseteq [n]$). The objective is to select a subset $A_i \in \mathcal{A}_i$, for all $i \in [k]$, so as to maximize $W^\varphi(A_1, A_2, \ldots, A_k) \coloneqq \sum_{a \in [n]} w_a \; \varphi(|A|_a)$. Here, $w_a \in \mathbb{R}_+$ is a weight associated with $a \in [n]$ and $|A|_a \coloneqq |\{i \in [k] : a \in A_i\}|$. We will refer to this problem as the φ-Resource Allocation problem.

While φ-Resource Allocation does not directly reduce to φ-MaxCoverage, the next theorem shows that it corresponds to maximizing φ-coverage functions subject to a matroid constraint. Hence, invoking our result from Section 2.2, we obtain a tight α_φ-approximation for φ-Resource Allocation:

▶ **Theorem 5** ([3]). For any normalized nondecreasing concave function φ, there exists a polynomial-time α_φ-approximation algorithm for φ-Resource Allocation. Furthermore, for $\varphi(n) = o(n)$, it is NP-hard to approximate φ-Resource Allocation within a factor better than $\alpha_\varphi + \varepsilon$, for any constant $\varepsilon > 0$.

4.3 Vehicle-Target Assignment

Vehicle-Target Assignment [17, 19] is another problem which highlights the applicability of coverage problems, with a concave φ. In particular, Vehicle-Target Assignment can be directly expressed as φ-Resource Allocation: the $[n]$ resources correspond to targets, the agents correspond to vehicles $i \in [k]$, each with a collection of covering choices $\mathcal{A}_i \subseteq 2^{[n]}$, and $\varphi^p(j) = \frac{1-(1-p)^j}{p}$, for a given parameter $p \in (0,1)$. As limit cases, we define $\varphi^0(j) \coloneqq \lim_{p \to 0} \varphi^p(j) = j$ and $\varphi^1(j) \coloneqq 1$. Since $\varphi^p(j)$ is concave, by a simple calculation and Theorem 5, we obtain a novel tight approximation ratio of $\alpha_{\varphi^p} = \frac{1-e^{-p}}{p}$ for this problem. Also, one can look at the capped version of this problem, $\varphi_\ell^p(j) \coloneqq \varphi^p(\min\{j, \ell\})$. In particular, we recover the ℓ-MultiCoverage function when $p = 0$. In Figure 1, we have plotted several cases of the tight approximations $\alpha_{\varphi_\ell^p}$ in function of ℓ for several values of ℓ:

Paccagnan and Marden [19] study the game-theoretic aspects of Vehicle-target assignment. A key goal in [19] is to bound the welfare loss incurred due to strategic selection by the k vehicles, i.e., the selection of each $A_i \in \mathcal{A}_i$ by a self-interested vehicle/agent $i \in [k]$. The loss is quantified in terms of the *Price of Anarchy* (PoA). Formally, this performance metric is defined as ratio between the welfare of the worst-possible equilibria and

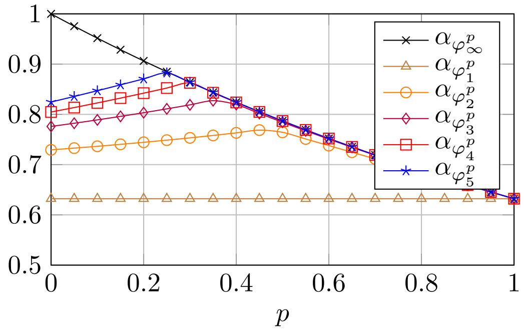


Figure 1 Tight approximation ratios $\alpha_{\varphi^p_\ell}$, where ℓ is the rank of the capped version of the p-Vehicle-Target Assignment problem. When $p = 0$, we recover the ℓ-coverage problem.

the optimal welfare. Paccagnan and Marden [19] show that, for computationally tractable equilibrium concepts (in particular, for coarse correlated equilibria), tight price of anarchy bounds can be obtained via linear programs.

Note that our hardness result (Theorem 1) provides upper bounds on PoA of tractable equilibrium concepts–this follows from the observation that computing an equilibrium provides a specific method for finding a coverage solution. In [8] and in the particular case of the ℓ-MultiCoverage problem, it is shown that this in fact an equality, i.e., PoA $= \alpha_\varphi$ if $\varphi(j) = \min\{j, \ell\}$ for all values of ℓ. However, numerically comparing the approximation ratio for Vehicle-Target Assignment, $\alpha_{\varphi^p} = \frac{1-e^{-p}}{p}$, with the optimal PoA bound, we note that α_{φ^p} can in fact be strictly greater than the PoA guarantee; see Figure 2.

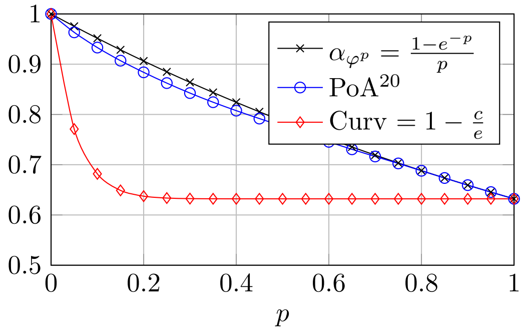


Figure 2 Comparison between the PoA and α_φ for the Vehicle-Target Assignment problem. Using the linear program found in [19], we were able to compute the blue curve PoA^{20}, the *Price of Anarchy* of this problem for $m = 20$ players. Since the PoA only decreases when the number of players grows, this means that PoA $< \alpha_\varphi$ in that case. As a comparison, the red curve Curv depicts the general approximation ratio (see [22]) obtained for submodular function with curvature c, with $c = 1 - \varphi^p(m) + \varphi^p(m-1)$ here.

4.4 Welfare Maximization for φ-Coverage

Maximizing (social) welfare by partitioning items among agents is a key problem in algorithmic game theory; see, e.g., the extensive work on combinatorial auctions [18]. The goal here is to partition t items among a set of k agents such that the sum of values achieved by the agents – referred to as the social welfare – is maximized. That is, one needs to partition $[t]$ into k pairwise disjoint subsets $A_1, A_2, \ldots, A_k$ with the objective of maximizing $\sum_{i=1}^{k} v_i(A_i)$. Here, $v_i(S)$ denotes the valuation that agent i has for a subset of items $S \subseteq [t]$.

When each agent's valuation v_i is submodular, a tight $(1 - e^{-1})$-approximation ratio is known for social welfare maximization [23]. This section shows that improved approximation guarantees can be achieved if, in particular, the agents' valuations are φ-coverage functions. Towards a stylized application of such valuations, consider a setting in which each "item" $b \in [t]$ represents a bundle (subset) of goods $T_b \subseteq [n]$ and the value of an agent increases with the number of copies of any good $a \in [n]$ that get accumulated. Indeed, if each agent's value for j copies of a good is $\varphi(j)$, then we have a φ-coverage function and the overall optimization problem is find a k-partition, $A_1, A_2, \ldots, A_k$, of $[t]$ that maximizes $\sum_{i=1}^{k} \left(\sum_{a \in [n]} \varphi\left(|A_i|_a \right) \right)$, where $|A_i|_a \coloneqq \{b \in A_i : a \in T_b\}$.

In the current setup, one can obtain an α_φ approximation ratio for social-welfare maximization by reducing this problem to φ-coverage with a matroid constraint, and applying the result from Section 2.2. Specifically, we can consider a partition matroid over the universe $[t] \times [k]$: for a bundle/item $b \in [t]$ and an agent $i \in [k]$, the element (b, i) in the universe represents that bundle b is assigned to agent i, i.e., $b \in A_i$. The partition-matroid constraint is imposed to ensure that each bundle b is assigned to at most one agent. Furthermore, we can create k copies of the underlying set of goods $[n]$ and set $T_{(b,i)} \coloneqq \{(a, i) : a \in T_b\}$ to map the φ-coverage over the universe to the social-welfare objective. This, overall, gives us the desired α_φ approximation guarantee.

Conclusion

We have introduced the φ-MaxCoverage problem where having c copies of element a gives a value $\varphi(c)$. We have shown that when φ is normalized, nondecreasing and concave, we can obtain an approximation guarantee given by the *Poisson concavity ratio* $\alpha_\varphi \coloneqq \min_{x \in \mathbb{N}^*} \frac{\mathbb{E}[\varphi(\mathrm{Poi}(x))]}{\varphi(\mathbb{E}[\mathrm{Poi}(x)])}$ and we showed it is tight for sublinear functions φ. The Poisson concavity ratio strictly beats the bound one gets when using the notion of curvature submodular functions, except in very special cases such as MaxCoverage where the two bounds are equal.

An interesting open question is whether there exists combinatorial algorithms that achieve this approximation ratio. As mentioned in [4], for the ℓ-MultiCoverage with $\ell \geq 2$, which is the special case where $\varphi(x) = \min\{x, \ell\}$, the simple greedy algorithm only gives a $1 - e^{-1}$ approximation ratio, which is strictly less than the ratio $\alpha_\varphi = 1 - \frac{\ell^\ell e^{-\ell}}{\ell!}$ in that case. Also, for any geometrically dominant vector $w = (\varphi(i+1) - \varphi(i))_{i \in \mathbb{N}}$ which is not p-geometric, such as Proportional Approval Voting, the greedy algorithm achieves an approximation ratio which is strictly less than α_φ (see Theorem 18 of [11]).

Another open question is whether the hardness result remains true even when $\varphi(n) \neq o(n)$. A good example is given by $\varphi(0) = 0$ and $\varphi(1 + t) = 1 + (1 - c)t$ with $c \in (0, 1)$. We know that the problem is hard for $c = 1$ but easy for $c = 0$. One can show that the approximation ratio achieved by our algorithm is $\alpha_\varphi = 1 - \frac{c}{e}$ in that case (which is the same approximation ratio obtained from the curvature in [22]), but the tightness of this approximation ratio remains open.

References

1 Alexander A. Ageev and Maxim Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *J. Comb. Optim.*, 8(3):307–328, 2004. doi:10.1023/B:JOCO.0000038913.96607.c2.

2 Siddharth Barman and Omar Fawzi. Algorithmic aspects of optimal channel coding. *IEEE Trans. Inf. Theory*, 64(2):1038–1045, 2018. doi:10.1109/TIT.2017.2696963.

3 Siddharth Barman, Omar Fawzi, and Paul Fermé. Tight approximation guarantees for concave coverage problems. *CoRR*, abs/2010.00970, 2020. arXiv:2010.00970.

4 Siddharth Barman, Omar Fawzi, Suprovat Ghoshal, and Emirhan Gürpinar. Tight approximation bounds for maximum multi-coverage. In Daniel Bienstock and Giacomo Zambelli, editors, *Integer Programming and Combinatorial Optimization - 21st International Conference, IPCO 2020, London, UK, June 8-10, 2020, Proceedings*, volume 12125 of *Lecture Notes in Computer Science*, pages 66–77. Springer, 2020. doi:10.1007/978-3-030-45771-6_6.

5 Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Procaccia, editors. *Handbook of Computational Social Choice.* Cambridge University Press, 2016. doi:10.1017/CBO9781107446984.

6 Markus Brill, Jean-François Laslier, and Piotr Skowron. Multiwinner approval rules as apportionment methods. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 414–420. AAAI Press, 2017. doi:10.1177/0951629818775518.

7 Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011. doi:10.1137/080733991.

8 Rahul Chandan, Dario Paccagnan, and Jason R. Marden. Optimal mechanisms for distributed resource-allocation. *CoRR*, abs/1911.07823, 2019. arXiv:1911.07823.

9 Michele Conforti and Gérard Cornuéjols. Submodular set functions, matroids and the greedy algorithm: Tight worst-case bounds and some generalizations of the Rado-Edmonds theorem. *Discret. Appl. Math.*, 7(3):251–274, 1984. doi:10.1016/0166-218X(84)90003-9.

10 Gérard Cornuéjols, Marshall L Fisher, and George L Nemhauser. Exceptional paper—location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management science*, 23(8):789–810, 1977. doi:10.1287/mnsc.23.8.789.

11 Szymon Dudycz, Pasin Manurangsi, Jan Marcinkowski, and Krzysztof Sornat. Tight approximation for proportional approval voting. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 276–282. ijcai.org, 2020. doi:10.24963/ijcai.2020/39.

12 Shaddin Dughmi and Jan Vondrák. Limitations of randomized mechanisms for combinatorial auctions. *Games Econ. Behav.*, 92:370–400, 2015. doi:10.1016/j.geb.2014.01.007.

13 Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

14 Vitaly Feldman and Pravesh Kothari. Learning coverage functions and private release of marginals. In Maria-Florina Balcan, Vitaly Feldman, and Csaba Szepesvári, editors, *Proceedings of The 27th Conference on Learning Theory, COLT 2014, Barcelona, Spain, June 13-15, 2014*, volume 35 of *JMLR Workshop and Conference Proceedings*, pages 679–702. JMLR.org, 2014. URL: http://proceedings.mlr.press/v35/feldman14a.html.

15 Dorit S. Hochbaum. Approximation algorithms for NP-hard problems. *SIGACT News*, 28(2):40–52, 1997. doi:10.1145/261342.571216.

16 Jason R. Marden and Adam Wierman. Distributed welfare games with applications to sensor coverage. In *Proceedings of the 47th IEEE Conference on Decision and Control, CDC 2008, December 9-11, 2008, Cancún, Mexico*, pages 1708–1713. IEEE, 2008. doi:10.1109/CDC.2008.4738800.

17 Robert A Murphey. Target-based weapon target assignment problems. In *Nonlinear assignment problems*, pages 39–53. Springer, 2000. doi:10.1007/978-1-4757-3155-2_3.

18 Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani, editors. *Algorithmic Game Theory.* Cambridge University Press, 2007. doi:10.1017/CBO9780511800481.

19 Dario Paccagnan and Jason R Marden. Utility design for distributed resource allocation–part II: Applications to submodular, covering, and supermodular problems. *CoRR*, abs/1807.01343, 2018. arXiv:1807.01343.

20 Moshe Shaked and J George Shanthikumar. *Stochastic orders*. Springer Science & Business Media, 2007. doi:10.1007/978-0-387-34675-5.

21 Piotr Skowron, Piotr Faliszewski, and Jérôme Lang. Finding a collective set of items: From proportional multirepresentation to group recommendation. *Artif. Intell.*, 241:191–216, 2016. doi:10.1016/j.artint.2016.09.003.

22 Maxim Sviridenko, Jan Vondrák, and Justin Ward. Optimal approximation for submodular and supermodular optimization with bounded curvature. *Math. Oper. Res.*, 42(4):1197–1218, 2017. doi:10.1287/moor.2016.0842.

23 Jan Vondrák. Submodularity in Combinatorial Optimization. *Univerzita Karlova, Matematicko-Fyzikální Fakulta*, 2007. URL: https://dspace.cuni.cz/bitstream/handle/20.500.11956/13738/140038775.pdf.

Symmetric Promise Constraint Satisfaction Problems: Beyond the Boolean Case

Libor Barto
Department of Algebra, Faculty of Mathematics and Physics,
Charles University, Prague, Czech Republic

Diego Battistelli
Department of Algebra, Faculty of Mathematics and Physics,
Charles University, Prague, Czech Republic

Kevin M. Berg
Department of Algebra, Faculty of Mathematics and Physics,
Charles University, Prague, Czech Republic

Abstract

The Promise Constraint Satisfaction Problem (PCSP) is a recently introduced vast generalization of the Constraint Satisfaction Problem (CSP). We investigate the computational complexity of a class of PCSPs beyond the most studied cases – approximation variants of satisfiability and graph coloring problems. We give an almost complete classification for the class of PCSPs of the form: given a 3-uniform hypergraph that has an admissible 2-coloring, find an admissible 3-coloring, where admissibility is given by a ternary symmetric relation. The only PCSP of this sort whose complexity is left open in this work is a natural hypergraph coloring problem, where admissibility is given by the relation "if two colors are equal, then the remaining one is higher."

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

Keywords and phrases constraint satisfaction problems, promise constraint satisfaction, Boolean PCSP, polymorphism

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.10

Funding All three authors have received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No 771005).

1 Introduction

The Constraint Satisfaction Problem (CSP) over a finite relational structure $\mathbf{A}$ (also called a *template*), denoted CSP($\mathbf{A}$), can be defined as a homomorphism problem with a fixed target structure. In the decision version of the problem, an instance is a finite relational structure $\mathbf{X}$ (of the same signature as $\mathbf{A}$) and the problem is to decide whether there exists a homomorphism (i.e., a relation-preserving map) from $\mathbf{X}$ to $\mathbf{A}$. In the search version of the problem, we are required to find such a homomorphism whenever it exists. Many computational problems, including various versions of logical satisfiability, graph coloring, and systems of equations can be represented in this form, see the survey [4]. For example, the CSP over $\mathbf{K}_3$ (the clique on 3 vertices) is the 3-coloring problem, the CSP over the two-element structure with all the ternary relations is the 3-SAT problem, the CSP over the two-element structure **1in3** with a single ternary relation $\{(0,0,1),(0,1,0),(1,0,0)\}$ is the positive 1-in-3-SAT, and the CSP over the two-element structure **NAE** with a single ternary relation $\{0,1\}^3 \setminus \{(0,0,0),(1,1,1)\}$ is the positive not-all-equal-3-SAT.

An early general complexity classification result was Schaeffer's dichotomy theorem for Boolean (i.e., two-element) templates [19]: each such CSP is in P or is NP-complete. Another influential result was a dichotomy theorem by Hell and Nešetřil [15] for CSPs over (undirected) graphs. Motivated in part by these two theorems, Feder and Vardi formulated

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 10; pp. 10:1–10:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

their dichotomy conjecture stating that the P versus NP-complete dichotomy remains true for CSPs over arbitrary finite structures. This conjecture inspired a very active research program in the last 20 years, culminating in a celebrated positive resolution independently obtained by Bulatov [11] and Zhuk [21]. Their proofs rely heavily on a general theory of CSPs developed in [16, 10] (the so-called *algebraic approach*) whose central concept is a *polymorphism*, a multivariate function preserving the relations in the template. An NP-hardness criterion in terms of polymorphisms has been isolated in [10] and was conjectured to be the only source of hardness – all the templates that do not satisfy this criterion should be in P. This is what Bulatov and Zhuk confirmed in their work.

This paper studies a recently introduced [1, 6] vast generalization of the fixed-template CSP, the *Promise CSP* (PCSP). A promise template is a pair $(\mathbf{A}, \mathbf{B})$ of finite relational structures such that $\mathbf{A} \to \mathbf{B}$, i.e., there exists a homomorphism from $\mathbf{A}$ to $\mathbf{B}$. The PCSP over $(\mathbf{A}, \mathbf{B})$ is then (in the search version) the following problem: given a relational structure $\mathbf{X}$ such that $\mathbf{X} \to \mathbf{A}$ (this is the promise), find a homomorphism $\mathbf{X} \to \mathbf{B}$ (which is guaranteed to exist since $\mathbf{A} \to \mathbf{B}$). This framework generalizes that of CSP (where $\mathbf{A} = \mathbf{B}$) and also includes many important problems in approximation, e.g., if $\mathbf{A} = \mathbf{K}_k$ and $\mathbf{B} = \mathbf{K}_l$, $k \leq l$, then $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ is the problem of finding an l-coloring of a k-colorable graph, a problem whose complexity is open after more than 40 years of research. On the other hand, the basics of the algebraic approach to CSPs from [5] can be generalized to PCSPs [12, 3], and this discovery gives us hope that a full complexity classification might be possible.

Motivated by this ambitious goal, a line of research focuses on studying restricted classes of templates, the two main directions being graph templates and Boolean templates, mimicking the two "base cases" in the CSP. For the graph templates, a complexity conjecture has been formulated by Brakensiek and Guruswami in [6] stating that all templates that are not in P for simple reasons ($\mathbf{A}$ or $\mathbf{B}$ has a loop or is bipartite) are NP-hard. Boolean templates form a rich source of examples and are the context where the PCSP framework was introduced [1, 6]. Boolean PCSPs are interesting both from the NP-hardness perspective and, unlike the graph templates, from the algorithmic perspective: a generalization of [3] in [9] brought the strongest NP-hardness criterion we currently have, which we will also employ in this paper, and the most general polynomial algorithm [7, 8] also came from investigating Boolean PCSPs. The strongest classification result obtained so far in this direction is the dichotomy theorem over Boolean *symmetric* templates, i.e., templates whose relations are all invariant under permutations of coordinates [6, 14].

Note that both undirected graphs and symmetric Boolean templates use symmetric relations – undirected graphs consist of a single binary relation over a domain of an arbitrary size, symmetric Boolean templates consist of arbitrarily many symmetric relations of arbitrary arities over a two-element domain. The focus of this paper is somewhere in between these two extremes: we study templates $(\mathbf{A}, \mathbf{B})$ where $\mathbf{A}$ consists of a single symmetric ternary relation over a two-element set. Our motivation was that the classification of more concrete templates of PCSPs is needed for improving the general theory, such as finding hardness and tractability criteria. The class we study is amenable for case analysis and, on the other hand, already includes important problems both from a hardness and an algorithmic perspective (e.g., k-coloring a 2-colorable 3-uniform hypergraph or finding a positive not-all-equal solution to a satisfiable positive 1-in-3-sat instance).

Let $(\mathbf{A}, \mathbf{B})$ be a PCSP template such that $\mathbf{A}$ has domain $\{0, 1\}$ and a single symmetric ternary relation $R \subseteq \{0, 1\}^3$, and let $\mathbf{B}$ consist of a single relation $R \subseteq B^3$ on a domain B. By taking into account the main result of [13] and ruling out some trivial cases (see Section 4 for details), we are left with the case where $\mathbf{A} = \mathbf{1in3}$ and R is also symmetric.

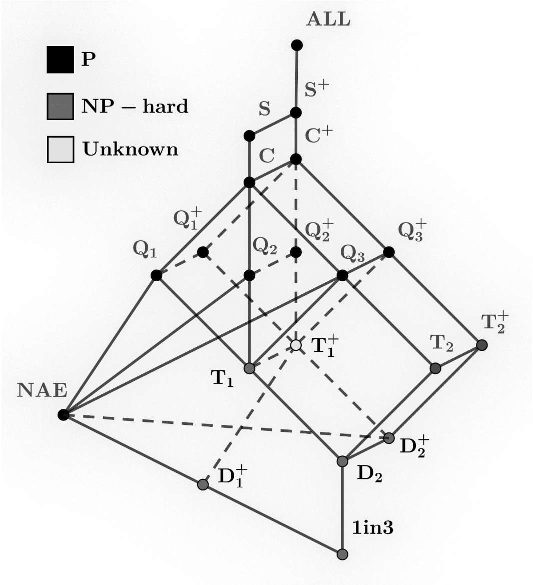


Figure 1 The Ordering of Homomorphism Classes of Symmetric Structures.

Note that in this situation PCSP$(\mathbf{A},\mathbf{B})$ has a hypergraph coloring interpretation: given a 3-uniform hypergraph that is **1in3**-colorable (that is, each vertex can be assigned a color from $\{0,1\}$ so that there is exactly one 1 appearing in each hyperedge), find a **B**-coloring (that is, a coloring by B such that the three colors appearing in each hyperedge are from R).[1]

1.1 Three-element domain

The first non-Boolean domain size, $|B| = 3$, already turns out to be interesting. Figure 1 shows all the templates $\mathbf{B}$ ordered by the relation $\mathbf{B} \leq \mathbf{B}'$ if $\mathbf{B} \to \mathbf{B}'$ (if $\mathbf{B} \leq \mathbf{B}' \leq \mathbf{B}$, only one representative is drawn; we also remark that lines are drawn solid or dashed only to improve clarity in the picture). As $\mathbf{B} \leq \mathbf{B}'$ implies that PCSP$(\mathbf{1in3},\mathbf{B}')$ reduces to PCSP$(\mathbf{1in3},\mathbf{B})$, the higher the template is, the "easier" the corresponding PCSP is. This fact and the notation of the templates are detailed in Sections 2 and 4.

We were able to classify all but one case:

▶ **Theorem 1.** *Let* $(\mathbf{1in3},\mathbf{B})$ *be a PCSP template, where* $\mathbf{B}$ *has domain-size three.*

- *If* $\mathbf{NAE} \to \mathbf{B}$ *or* $\mathbf{T}_2 \to \mathbf{B}$*, then* PCSP$(\mathbf{1in3},\mathbf{B})$ *is in P.*
- *If* $\mathbf{B} \to \mathbf{T}_1$ *or* $\mathbf{B} \to \mathbf{D}_1^+$ *or* $\mathbf{B} \to \mathbf{D}_2^+$*, then* PCSP$(\mathbf{1in3},\mathbf{B})$ *is NP-hard.*

Even though $|B| = 3$ is a small domain size, many interesting phenomena already show up, and we believe that the collection of templates is a valuable source of examples for further exploration. We now emphasize some of the phenomena and open questions.

[1] We commit a slight imprecision here, since the relation of the instance can contain entries with repeated coordinates, and thus not all instances correspond to 3-uniform hypergraphs. The difference is insignificant for our results.

Tractability of $\mathbf{NAE}$ and $\mathbf{T}_2$ can be obtained by using the sufficient condition from [7, 8] – the existence of block-symmetric polymorphisms of arbitarily large block-sizes. However, $\mathbf{T}_2$ is "simpler" in that one can use a finite-template CSP to solve it in polynomial-time, while for $\mathbf{NAE}$ no such finite template exists [2]. Among the templates in Figure 1, where is the borderline for such a "finite tractability"?

Two features of our NP-hardness results are worth noting. First, they are obtained by applying the currently strongest NP-hardness condition from [9]. It seems (but we do not prove it here) that weaker conditions, which were sufficient for NP-hardness of symmetric Boolean CSPs [6, 14] and NP-hardness of approximate hypergraph coloring [13] (cf. [3]), are not sufficient here, namely for $\mathbf{D}_2^+$. The only other such templates we are aware of are those from [9]. Second, NP-hardness of $\mathbf{D}_1^+$ has, similarly to [13], topological sources since it employs the high chromatic number of Kneser's graphs [18] – a result of Lovász that started topological combinatorics. However, the application of Kneser's graphs is more direct in our situation, and this may help in further improving the topological methods in PCSPs. In particular, it would be desirable to find a common generalization of the proof in [13] and in the paper [17], which employs algebraic topology in a seemingly different way.

The unresolved case, $\mathbf{T}_1^+$, in fact corresponds to a natural hypergraph coloring problem that appears to be new: given a **1in3**-colorable 3-uniform hypergraph, find a 3-coloring such that, in each hyperedge, if two colors are equal, then the third one is *higher* (as opposed to "different" for the standard hypergraph coloring). We conjecture that this problem, as well as the natural generalization to larger domains, is NP-complete. If true, there is a unique source of hardness for our templates.

1.2 Larger domains

For a 4-element B, the preceding conjecture would resolve all the cases with the exception of the interval between $\check{\mathbf{C}}$ and $\check{\mathbf{C}}^+$, where $\check{\mathbf{C}}$ is given by the relation containing the tuples $(0,0,1)$, $(1,1,2)$, $(2,2,3)$, $(3,3,0)$ and their permutations, and $\check{\mathbf{C}}^+$ is given by the same relation with all the "rainbow" tuples (i,j,k) such that $|\{i,j,k\}| = 3$.[2]

We are able to show that the bottom of the interval corresponds to an NP-hard PCSP, and the top one gives a template that does not satisfy the sufficient condition for tractability from [7, 8].

▶ **Theorem 2.** $\mathrm{PCSP}(\mathbf{1in3}, \check{\mathbf{C}})$ *is NP-hard. The template* $(\mathbf{1in3}, \check{\mathbf{C}}^+)$ *does not have a block symmetric polymorphism with two blocks of sizes 23 and 24.*

The theorem suggests that even for $|B| = 4$, essentially the only tractable cases are $\mathbf{NAE}$ and $\mathbf{T}_2$. Is this the case on arbitrary domains?

2 Basic definitions

Throughout this paper, we adopt the convention that $[n] = \{1, 2, \dots, n\}$.

A *relational structure* (of a finite signature) is a tuple $\mathbf{A} = (A; R_1, \dots, R_n)$, where A is a set called the *domain* of $\mathbf{A}$, and each R_i is a relation of arity $\mathrm{ar}_i \geq 1$, that is, a nonempty subset of A^{ar_i}. A relational structure is *symmetric* if each relation in it is invariant under any permutation of coordinates. Two relational structures $\mathbf{A} = (A; R_1, \dots, R_n)$ and $\mathbf{B} = (B; R'_1, \dots, R'_{n'})$ *have the same signature* if $n = n'$ and each R_i has the same arity as R'_i.

[2] The notation is derived from the Czech word for a square – čtverec.

In this situation, a mapping $f : A \to B$ is a homomorphism from $\mathbf{A}$ to $\mathbf{B}$, written $f : \mathbf{A} \to \mathbf{B}$, if it preserves the relations, that is, for each i and each tuple $\mathbf{a} \in R_i$, we have $f(\mathbf{a}) \in R'_i$, where f is applied to $\mathbf{a}$ component-wise. The fact that there exists a homomorphism from $\mathbf{A}$ to $\mathbf{B}$ is denoted by $\mathbf{A} \to \mathbf{B}$.

▶ **Definition 3.** *A* PCSP template *is a pair of finite relational structures with the same signature,* $\mathbf{A}$*,* $\mathbf{B}$ *such that* $\mathbf{A} \to \mathbf{B}$*. We denote the PCSP template of* $\mathbf{A}$ *and* $\mathbf{B}$ *by* $(\mathbf{A}, \mathbf{B})$*.*

For a given PCSP template, it is possible to define both a decision problem and a search problem variant of the PCSP.

▶ **Definition 4.** *Let* $(\mathbf{A}, \mathbf{B})$ *be a PCSP template. The* decision version of PCSP$(\mathbf{A}, \mathbf{B})$ *is, given an input structure* $\mathbf{I}$ *with the same signature as* $\mathbf{A}$ *and* $\mathbf{B}$*, to output yes if* $\mathbf{I} \to \mathbf{A}$ *and no if* $\mathbf{I} \not\to \mathbf{B}$*.*

The search version of PCSP$(\mathbf{A}, \mathbf{B})$ *is, given an input structure* $\mathbf{I}$ *with the same signature as* $\mathbf{A}$ *and* $\mathbf{B}$ *mapping homomorphically to* $\mathbf{A}$*, to find a homomorphism* $h : \mathbf{I} \to \mathbf{B}$*.*

It is not hard to see that the decision version of PCSP$(\mathbf{A}, \mathbf{B})$ can be reduced to its search version. The tractability results in this paper apply to the search version (and hence also to the decision version), while NP-hardness results apply to the decision version (and hence also to the search version).

The following concept captures the situation when one PCSP can be reduced to another one by the trivial reduction, that is, the reduction that does not change the instance.

▶ **Definition 5.** *Let* $(\mathbf{A}, \mathbf{B})$ *and* $(\mathbf{A}', \mathbf{B}')$ *be PCSP templates of the same signature. We say that* $(\mathbf{A}', \mathbf{B}')$ *is a* homomorphic relaxation *of* $(\mathbf{A}, \mathbf{B})$ *if* $\mathbf{A}' \to \mathbf{A}$ *and* $\mathbf{B} \to \mathbf{B}'$*.*

Observe that, indeed, the trivial reduction from PCSP$(\mathbf{A}', \mathbf{B}')$ to PCSP$(\mathbf{A}, \mathbf{B})$ is correct if and only if $(\mathbf{A}', \mathbf{B}')$ is a homomorphic relaxation of $(\mathbf{A}, \mathbf{B})$.

A crucial notion for the algebraic approach to PCSP is a *polymorphism*. A polymorphism of a template is simply a homomorphism from a Cartesian power of the first structure to the second one. This can be spelled out as follows.

▶ **Definition 6.** *Let* $(\mathbf{A}, \mathbf{B})$ *be a PCSP template. A mapping* $f : A^n \to B$ *(where* n *is a positive integer) is a* polymorphism of arity n *if, for each pair of corresponding relations* R_i *and* R'_i *in the signatures of* $\mathbf{A}$ *and* $\mathbf{B}$*, respectively, and any* $(r_{1,1}, r_{2,1}, \dots, r_{n,1}), \dots, (r_{1,\mathrm{ar}_i}, r_{2,\mathrm{ar}_i}, \dots, r_{n,\mathrm{ar}_i})$ *with* $(r_{j,1}, r_{j,2}, \dots, r_{j,\mathrm{ar}_i}) \in R_i$ *for all* $j \in [n]$*, we have* $(f(r_{1,1}, r_{2,1}, \dots, r_{n,1}), \dots, f(r_{1,\mathrm{ar}_i}, r_{2,\mathrm{ar}_i}, \dots, r_{n,\mathrm{ar}_i})) \in R'_i$*.*

Another core concept in the algebraic approach is a *minor*.

▶ **Definition 7.** *Let* $f : A^n \to B$*,* $\alpha : [n] \to [m]$ *be mappings. A* minor *of* f *given by* α *is the mapping* $f^\alpha : A^m \to B$ *defined by*

$$f^\alpha(a_1, \dots, a_m) = f(a_{\alpha(1)}, \dots, a_{\alpha(n)})$$

for every $a_1, \dots, a_m \in A$*. A function* $g : A^m \to B$ *is a minor of* f *if* $g = f^\alpha$ *for some* α*.*

The significance of polymorphisms stems from the fact that the computational complexity of PCSP$(\mathbf{A}, \mathbf{B})$ depends only on the set of all polymorphisms of the template $(\mathbf{A}, \mathbf{B})$ [6, 12, 3]. This set is a *minion*, i.e., it is closed under taking minors.

Table 1 Diagrams of Symmetric Structures.

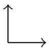
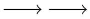
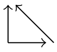
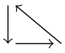
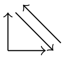
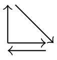
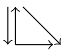
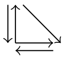
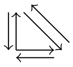
Diagram						
Structure $\mathbf{B}$	**1in3**	**NAE**	$\mathbf{D_1}$	$\mathbf{D_2}$	$\mathbf{T_1}$	$\mathbf{T_2}$

Diagram						
Structure $\mathbf{B}$	$\mathbf{Q_1}$	$\mathbf{Q_2}$	$\mathbf{Q_3}$	**C**	**S**	

3 Templates

In this section we introduce the notation for the templates considered in the paper, and provide several easy observations about these templates.

We consider symmetric relational structures with a single ternary relation. To each such structure $\mathbf{B} = (B; R)$ we associate *its digraph* by taking B as the vertex set and including the arc $b \to b'$ if and only if $(b, b, b') \in R$. By $\mathbf{B}^+$ we denote the structure obtained from $\mathbf{B}$ by adding to R all the tuples (b, b', b'') with $|\{b, b', b''\}| = 3$. Note that this is the "largest" structure with the same associated digraph as $\mathbf{B}$. Also observe that over a three-element domain, i.e., $|B| = 3$, there are exactly two structures with the same associated digraph. The notational convention for 3-element structures in Figure 1 is given in Table 1.[3] The names in the table refer to the smaller of the two structures with the same digraph, e.g., the relation of $\mathbf{D}_2$ consists of all permutations of the triples $(0, 0, 1)$, $(1, 1, 2)$, while $\mathbf{D}_2^+$ also contains $(0, 1, 2)$ and its permutations. Of course, the structure depends on the concrete choice of vertices, but the choice is irrelevant for our purposes.

It is a simple exercise to verify that the ordering in Figure 1 is correct, and we do not give the details here. Let us just observe that $\mathbf{T}_1^+$ is the only case not covered by Theorem 1. Indeed, the digraph associated to a three-element $\mathbf{B}$ either contains a directed cycle, or is acyclic. In the former case, depending on the length of the cycle we have $\mathbf{NAE} \to \mathbf{B}$ (length 1 or 2) or $\mathbf{T}_2 \to \mathbf{B}$ (length 1 or 3). In the latter case, the digraph can be extended to a linear order, so $\mathbf{B} \to \mathbf{T}_1^+$. If $\mathbf{B} \neq \mathbf{T}_1^+$, then $\mathbf{B}$ has a homomorphism to a symmetric substructure of $\mathbf{T}_1^+$ with one of the four triples, $(0, 0, 1)$, $(0, 0, 2)$, $(1, 1, 2)$, $(0, 1, 2)$, missing. By omitting the second, the third, or the fourth tuple we get the structures $\mathbf{D}_2^+$, $\mathbf{D}_1^+$, $\mathbf{T}_1$ from the second item of Theorem 1, respectively. By omitting the first tuple, we get a structure $\mathbf{B}$ such that $\mathbf{B} \to \mathbf{1in3} \to \mathbf{B}$, and so this structure sits at the bottom of Figure 1.

The unresolved case structure $\mathbf{T}_1^+$ has a simple description. A tuple (b_1, b_2, b_3) is in its relation if and only if the following condition is satisfied: if two of b_1, b_2, b_3 are equal to b, then the remaining one must be strictly greater than b in the linear order $0 < 1 < 2$. We denote the structure obtained by the same definition on a k-element domain ordered $0 < 1 < 2 < \dots < k-1$ by $\mathbf{LO}_k$, e.g., $\mathbf{LO}_2 = \mathbf{1in3}$ and $\mathbf{LO}_3 = \mathbf{T}_1^+$.

For the case $|B| = 4$, a similar case analysis shows that the only structures $\mathbf{B}$ with $\mathbf{NAE}, \mathbf{T}_2 \not\to \mathbf{B}$ and $\mathbf{B} \not\to \mathbf{LO}_4$ are the structures whose associated digraph is the directed cycle of length 4 – the structures in the interval between $\check{\mathbf{C}}$ and $\check{\mathbf{C}}^+$ alluded to in the introduction.

[3] The notation is derived from the number of edges of the associated digraph in Italian.

Finally, we denote by $\mathbf{NAE}_k$ the structure with a k-element domain and the ternary non-all-equal relation, e.g., $\mathbf{NAE}_2 = \mathbf{NAE}$.

4 Tractability and hardness

In this section we deal with the simple cases, as well as the cases that are resolved by known results. We also provide the hardness criterion that we will employ for the more complex cases. Throughout this section we consider a PCSP template $(\mathbf{A}, \mathbf{B})$ such that $\mathbf{A}$ is a relational structure with the two-element domain $\{0, 1\}$ and a single symmetric ternary relation, and $\mathbf{B} = (B; R)$, where $B = \{0, 1, \dots\}$ and $R \subseteq B^3$.

4.1 Symmetrization

First, observe that R can be assumed symmetric without loss of generality. We first note that there is a trivial reduction from $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ to $\mathrm{PCSP}(\mathbf{A}, \mathbf{B}_{\mathrm{sym}})$, where $\mathbf{B}_{\mathrm{sym}} = (B, R_{\mathrm{sym}})$ is the *lower symmetrization* of $\mathbf{B}$, i.e., R_{sym} consists of all the tuples whose every permutation is in R – in particular, R_{sym} is symmetric. In the other direction, given an instance $\mathbf{X}$ of $\mathrm{PCSP}(\mathbf{A}, \mathbf{B}_{\mathrm{sym}})$ such that $\mathbf{X} \to \mathbf{A}$, we also have $\mathbf{X}^{\mathrm{sym}} \to \mathbf{A}$, where $\mathbf{X}^{\mathrm{sym}} = (X, S^{\mathrm{sym}})$ is the *symmetrization* of $\mathbf{X} = (X, S)$, i.e., S^{sym} contains all the permutations of the tuples in S. On the other hand, $\mathbf{X}^{\mathrm{sym}} \to \mathbf{B}$ implies $\mathbf{X} \to \mathbf{B}_{\mathrm{sym}}$, and it follows that $\mathbf{X} \to \mathbf{X}^{\mathrm{sym}}$ is a correct reduction from $\mathrm{PCSP}(\mathbf{A}, \mathbf{B}_{\mathrm{sym}})$ to $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$. These PCSPs are therefore equivalent.

For the remainder of this section we assume that $\mathbf{B}$ is a symmetric structure.

4.2 Tractability

If R (the relation in $\mathbf{B}$) or the relation in $\mathbf{A}$ contains a constant tuple, $(\mathbf{A}, \mathbf{B})$ is a homomorphic relaxation of the "trivial" template whose two structures have a one-element domain. In particular, $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ is in P.

If $\mathbf{A} = \mathbf{1in3}$ and $\mathbf{NAE} \to \mathbf{B}$, then $(\mathbf{A}, \mathbf{B})$ is a homomorphic relaxation of $(\mathbf{1in3}, \mathbf{NAE})$. The PCSP over the latter template is in P by [6], and therefore so is the PCSP over the former.

The remaining tractable case in Theorem 1 is $\mathbf{A} = \mathbf{1in3}$ and $\mathbf{T}_2 \to \mathbf{B}$. These templates are homomorphic relaxations of $(\mathbf{T}_2, \mathbf{T}_2)$. But the PCSP over $(\mathbf{T}_2, \mathbf{T}_2)$, i.e., the CSP over $\mathbf{T}_2$, is in P because the relation of $\mathbf{T}_2$ can be described as $\{(x, y, z) \in \{0, 1, 2\}^3 : x + y + z = 1 \pmod 3\}$, and so $\mathrm{CSP}(\mathbf{T}_2)$ can be efficiently solved by solving a system of linear equations over the three-element field.

These tractability results can be also derived from a recent theorem that we now state. We require a definition. A mapping $f : A^n \to B$ is *block-symmetric of width k* if there exists a partition of the coordinates of f into blocks $X_1 \cup \dots \cup X_l = [n]$ of size at least k such that f is permutation-invariant within each coordinate block X_i.

▶ **Theorem 8** ([7, 8]). *The following are equivalent for every PCSP template* $(\mathbf{A}, \mathbf{B})$.

- $(\mathbf{A}, \mathbf{B})$ *has block-symmetric polymorphisms of arbitrarily high width.*
- *For every* $k \in \mathbb{N}$, $(\mathbf{A}, \mathbf{B})$ *has a block-symmetric polymorphism of arity* $2k + 1$ *with two blocks of size* k *and* $k + 1$.

If these equivalent conditions are satisfied, then $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ *is in P.*

In fact, this theorem is strong enough to prove the tractability of all the currently known tractable Boolean PCSPs. In Appendix B we use this fact to provide evidence for the NP-hardness of $\mathrm{PCSP}(\mathbf{1in3}, \check{\mathbf{C}}^{+})$: we prove that the template does not have a block-symmetric polymorphism with two blocks of sizes 23 and 24, as claimed in the second part of Theorem 2.

4.3 Hardness

If R does not contain a constant tuple and $\mathbf{A} = \mathbf{NAE}$, then $(\mathbf{NAE}, \mathbf{NAE}_{|B|})$ is a homomorphic relaxation of $(\mathbf{A}, \mathbf{B})$, and $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ is therefore NP-hard by the following theorem.

▶ **Theorem 9** ([13]). $\mathrm{PCSP}(\mathbf{NAE}_k, \mathbf{NAE}_l)$ *is NP-hard for every* $2 \leq k \leq l$.

The hard cases in Theorems 1, 2 – that is, $\mathbf{A} = \mathbf{1in3}$, and $\mathbf{B} = \mathbf{D}_1^+$, $\mathbf{D}_2^+$, $\mathbf{T}_1$, or $\check{\mathbf{C}}$ – are dealt with in Sections 5, 6, and 7, and Appendix A, respectively. All of these results employ an NP-hardness criterion that we now state. We will require an additional piece of notation. A *chain of minors* is a sequence of the form $(f_0, \alpha_{0,1}, f_1, \alpha_{1,2}, \ldots, \alpha_{l-1,l}, f_l)$ where $f_0, \ldots, f_l : A^{n_i} \to B$, $\alpha_{i-1,i} : [n_{i-1}] \to [n_i]$, and $f_{i-1}^{\alpha_{i-1,i}} = f_i$ for every $i \in [l]$. We will then write $\alpha_{i,j} : [n_i] \to [n_j]$ for the composition of $\alpha_{i,i+1}, \alpha_{i+1,i+2}, \ldots, \alpha_{j-1,j}$. Note that $f_i^{\alpha_{i,j}} = f_j$.

▶ **Theorem 10** (Corollary 4.2. in [9]). *Let* $(\mathbf{A}, \mathbf{B})$ *be a PCSP template. Suppose there are constants* $k, l \in \mathbb{N}$ *and an assignment of a set of at most* k *coordinates* $\mathrm{sel}(f) \subseteq [\mathrm{ar}(f)]$ *to every polymorphism* f *of* $(\mathbf{A}, \mathbf{B})$ *such that for every chain of minors* $(f_0, \alpha_{0,1}, \ldots, f_l)$ *with each* f_i *a polymorphism of* $(\mathbf{A}, \mathbf{B})$*, there are* $0 \leq i < j \leq l$ *such that* $\alpha_{i,j}(\mathrm{sel}(f_i)) \cap \mathrm{sel}(f_j) \neq \emptyset$ *(or, equivalently,* $\mathrm{sel}(f_i) \cap \alpha_{i,j}^{-1}(\mathrm{sel}(f_j)) \neq \emptyset$*). Then* $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ *is NP-hard.*

The special case $l = 1$ of this theorem is sufficient to prove the NP-hardness of all NP-hard symmetric Boolean PCSPs. For Theorem 9, $l = 1$ is not sufficient; however, it can be derived using a still weaker version of Theorem 10.[4] Theorem 10 in its full power was first used in [9] to prove the NP-hardness of certain symmetric non-Boolean PCSPs.

4.4 0-sets, 1-sets, ...

We conclude this section by introducing the notion of an i-set, which will be used extensively throughout the rest of the paper.

Given a mapping $f : \{0,1\}^n \to B$ (usually an n-ary polymorphism of $(\mathbf{1in3}, \mathbf{B})$) and a subset of coordinates $X \subseteq [n]$, we write $f(X)$ for the value $f(a_1, \ldots, a_n)$ where $a_i = 1$ if $i \in X$ and $a_i = 0$ else. We say that X is a *0-set* if $f(X) = 0$. 1-sets, 2-sets, etc. are defined similarly. The function f will be always clear from the context.

Observe that $f : \{0,1\}^n \to B$ is a polymorphism of $(\mathbf{1in3}, \mathbf{B})$ if and only if, for every partition of the coordinates of f into three blocks $X \cup Y \cup Z = [n]$, we have $(f(X), f(Y), f(Z)) \in R$. The forward implication of this observation will be applied many times in the proofs, and we simply say, e.g., "by compatibility of $f(X)$, $f(Y)$ and $f(Z)$," "by compatibility applied to X and Y, ...," or "by compatibility, ... " in such situations.

For example, it is common in our templates for the relation R to have no tuple of the form $(2, 2, *)$. Therefore, if X and Y are both known to be 2-sets, we would argue that, by compatibility, it must be the case that X and Y are not disjoint. In such cases, we would say, e.g., "there are no disjoint 2-sets of f."

One useful feature of i-sets is that they are closed under preimages within a chain of minors – that is, if $(f_0, \alpha_{0,1}, f_1, \alpha_{1,2}, \ldots, \alpha_{l-1,l}, f_l)$ is a chain of minors for $(\mathbf{1in3}, \mathbf{B})$ and X is an i-set for some polymorphism f_{j_1} in the chain with $0 \leq j_1 \leq l$, then for any j_2 with $0 \leq j_2 < j_1$, $\alpha_{j_2,j_1}^{-1}(X)$ is an i-set of f_{j_2}.

[4] In fact, the proof in [3] (Theorem 5.23) is based on [13] and applies a version which uses a super-constant k (it is enough that, e.g., k is bounded by a polynomial in the logarithm of the arity of f). Wrochna [20] has shown that this is not necessary. We also remark that Theorem 10 can also be strengthened to super-constant values of k.

5 $\mathbf{D}_1^+$

In this section we prove the NP-hardness of $\mathrm{PCSP}(\mathbf{1in3}, \mathbf{D}_1^+)$, where $\mathbf{D}_1^+ = (\{0,1,2\}, R)$ and R consists of all the permutations of the tuples $(0,0,1)$, $(0,0,2)$, and $(0,1,2)$.

Before applying Theorem 10, we first derive several properties of polymorphisms of the template. Let us fix any polymorphism $f : \{0,1\}^n \to \{0,1,2\}$ of $(\mathbf{1in3}, \mathbf{D}_1^+)$

▶ **Lemma 11.** *There are no two disjoint 1-sets nor 2-sets.*

Proof. If X and Y are i-sets for the same $i \in \{1,2\}$, then $(f(X), f(Y), f([n] \setminus (X \cup Y)) \in R$ by compatibility. But R does not contain any tuple of the form $(1,1,*)$ or $(2,2,*)$, a contradiction. ◀

The next lemma uses the high chromatic number of Kneser graphs. Recall that the Kneser graph with parameters n, m, denoted $\mathbf{KG}_{n,m}$, is the graph whose vertices are the m-element subsets of $[n]$, and where two vertices are adjacent if and only if the two corresponding sets are disjoint.

▶ **Theorem 12** (Lovász [18]). *For $n \geq 2m$, there is no coloring of $\mathbf{KG}_{n,m}$ by strictly less than $n - 2m + 2$ colors.*

▶ **Lemma 13.** *f has a 1-set or a 2-set of size at most 3.*

Proof. We first assume that $n \geq 2$ and set $m = \lfloor (n-2)/2 \rfloor$. Since $n - 2m + 2 \geq 4$, Theorem 12 implies that the mapping $X \mapsto f(X)$ cannot be a valid coloring of $\mathbf{KG}_{n,m}$. Therefore, there are two disjoint sets X and Y of size m such that $f(X) = f(Y)$. By compatibility applied to X and Y, the set $Z = [n] \setminus (X \cup Y)$ is a 1-set or 2-set. Its size is at most $n - 2m \leq 3$.

In the case $n = 1$, $\{1\}$ is itself a 1-set or a 2-set by compatibility applied to $\emptyset$ and $\emptyset$. ◀

We are ready to prove the NP-hardness of our template.

▶ **Theorem 14.** $\mathrm{PCSP}(\mathbf{1in3}, \mathbf{D}_1^+)$ *is NP-hard*

Proof. We apply Theorem 10 with $k = 3$ and $l = 2$. For a polymorphism f of the template, we define $\mathrm{sel}(f)$ as a 1-set or a 2-set of size at most 3 – such a set exists by Lemma 13 (if both a small 1-set and a small 2-set exist, we select arbitrarily).

Let $(f_0, \alpha_{0,1}, f_1, \alpha_{1,2}, f_2)$ be a chain of minors consisting of polymorphisms. By the pigeonhole principle, there exists $0 \leq i < j \leq 2$ such that $\mathrm{sel}(f_i)$ and $\mathrm{sel}(f_j)$ is an m-set for the same $m \in \{1,2\}$. Then $\alpha_{i,j}^{-1}(\mathrm{sel}(f_j))$ is an m-set as well and then $\mathrm{sel}(f_i) \cap \alpha_{i,j}^{-1}(\mathrm{sel}(f_j)) \neq \emptyset$ by Lemma 11, as required. ◀

6 $\mathbf{D}_2^+$

In this section we prove the NP-hardness of $\mathrm{PCSP}(\mathbf{1in3}, \mathbf{D}_2^+)$, where $\mathbf{D}_2^+ = (\{0,1,2\}, R)$ and R consists of all the permutations of the tuples $(0,0,1)$, $(1,1,2)$, and $(0,1,2)$.

Let $f : \{0,1\}^n \to \{0,1,2\}$ be a polymorphism of $(\mathbf{1in3}, \mathbf{D}_2^+)$. We start with a lemma that concerns unions of i-sets.

▶ **Lemma 15.** *Let X and Y be disjoint subsets of $[n]$.*

(a) *If $f(\emptyset) = 0$, $f(X) = 0$, and $f(Y) \in \{0,2\}$, then $f(X \cup Y) \in \{0,2\}$.*
(b) *If $f(\emptyset) = 0$, $f(X) = 1$, and $f(Y) \in \{0,1\}$, then $f(X \cup Y) = 1$.*
(c) *If $f(\emptyset) = 1$, $f(X) = f(Y) = 1$, then $f(X \cup Y) \in \{0,1\}$.*
(d) *If $f(\emptyset) = 1$, $f(X) = f(Y) = 0$, then $f(X \cup Y) = 2$.*

Proof. For the first item, by compatibility applied to X and Y, the complement $Z = [n] \setminus (X \cup Y)$ is a 1-set. Therefore, by compatibility applied to $\emptyset$ and Z, $X \cup Y$ is a 0-set or a 2-set. The proof for the remaining items is similar. ◀

The following lemma will be useful in the case that $\emptyset$ is a 0-set. Note that in this case $[n]$ is a 1-set by compatibility applied to $\emptyset$ and $\emptyset$.

▶ **Lemma 16.** *Assume $f(\emptyset) = 0$ and that f has no singleton 2-set. Then f has a singleton 1-set and does not have any two disjoint 1-sets.*

Proof. If every singleton is a 0-set, then by adding to $\emptyset$ singletons, one by one, and using item (a) of Lemma 15, we get that $[n]$ is a 0-set or a 2-set. But by the preceding observation, $[n]$ is a 1-set. This contradiction shows that there exists a singleton 1-set.

For the second part of the claim, suppose X and Y are disjoint 1-sets. By adding to Y singletons and using item (b) of Lemma 15, we obtain that $[n] \setminus X$ is a 1-set, a contradiction to compatibility applied to $\emptyset$ and X. ◀

We now consider the case that $\emptyset$ is a 1-set. Observe that $[n]$ is a 2-set in this case.

▶ **Lemma 17.** *Assume $f(\emptyset) = 1$ and that, for some $j \geq 2$, all at most j-element subsets of $[n]$ are 1-sets. Then $j < n$ and all $(j+1)$-element subsets of $[n]$ are 1-sets.*

Proof. Clearly $j < n$ as $[n]$ is a 2-set. Assume, for a contradiction, that for some j-element X and $y \notin X$, the set $Y := X \cup \{y\}$ is not a 1-set. Since X and $\{y\}$ are 1-sets, then Y is a 0-set by item (c) of Lemma 15.

We prove by induction on i that every set Z of size i disjoint with Y is a 1-set. The base case of the induction may be, e.g., $i = 0$ (or $i = j$). For the induction step, consider an $(i+1)$-element Z disjoint from Y and write $Z = Z' \cup \{z\}$ where $|Z'| = i$. By the induction hypothesis Z' is a 1-set. The set $\{y, z\}$ is a 1-set as well by assumption (note that $j \geq 2$). Therefore $Z' \cup \{y, z\} = Z \cup \{y\}$ is a 0-set or 1-set by item (c) of Lemma 15. By compatibility applied to X and $Z \cup \{y\}$, the complement $W = [n] \setminus (X \cup Z \cup \{y\}) = [n] \setminus (Y \cup Z)$ is a 0-set or 2-set. But then, by compatibility applied to Y (a 0-set) and W, Z is a 1-set, as required.

We have proved that $[n] \setminus Y$ is a 1-set, a contradiction to compatibility applied to $\emptyset$ and Y. ◀

▶ **Lemma 18.** *If $f(\emptyset) = 1$, then there exists a 0-set or a 2-set of size at most 2.*

Proof. In the opposite case, every set of coordinates of size at most 2 is a 1-set. It would then follow from Lemma 17 that $[n]$ is a 1-set, a contradiction. ◀

Equipped with these lemmata, we can now proceed to our main argument for this section.

▶ **Theorem 19.** $\mathrm{PCSP}(\mathbf{1in3}, \mathbf{D}_2^+)$ *is NP-hard.*

Proof. We apply Theorem 10 with $k = 2$ and $l = 5$. We assign to a polymorphism its *type* and define $\mathrm{sel}(f)$ as follows.

- Type 1: f has a 2-set X of size at most 2. In this case we set $\mathrm{sel}(f) = X$.
- Type 2: f has no 2-set of size at most 2, $f(\emptyset) = 0$, and $\{x\}$ is a 1-set for some $x \in [n]$. We set $\mathrm{sel}(f) = \{x\}$.
- Type 3: f has no 2-set of size at most 2, $f(\emptyset) = 1$, and f has a 0-set X of size at most 2. We set $\mathrm{sel}(f) = X$.

Note that $\emptyset$ cannot be a 2-set. The first part of Lemma 16 and Lemma 18 then guarantee that every polymorphism is of one of the three types.

Let $(f_0, \alpha_{0,1}, \dots, f_l)$ be a chain of minors consisting of polymorphisms. Note that $f_i(\emptyset)$ does not depend on i, therefore types 2 and 3 do not simultaneously occur in the chain. If, for some $i < j$, both f_i and f_j have type 1, then $\mathrm{sel}(f_i)$ and $\alpha_{i,j}^{-1}(\mathrm{sel}(f_j))$ are both 2-sets, so they have a nonempty intersection. Similarly, if two polymorphisms in this chain have type 2, then we obtain a nonempty intersection by the second part of Lemma 16.

Otherwise, since $l = 5$, the chain contains four polymorphisms f_{i_1}, f_{i_2}, f_{i_3}, f_{i_4} of type 3 (where $i_1 < i_2 < i_3 < i_4$). Let $X_1 = \mathrm{sel}(f_{i_1})$ and $X_j = \alpha_{i_1,i_j}^{-1}(\mathrm{sel}(f_{i_j}))$ for $j = 2, 3, 4$. These four sets are 0-sets (as preimages of 0-sets). If they are pairwise disjoint, then $X_1 \cup X_2$ and $X_3 \cup X_4$ are disjoint sets, which are 2-sets by item (d) in Lemma 15, a contradiction. Therefore, two of these sets, say X_j and $X_{j'}$, have a nonempty intersection. But then $Y := \mathrm{sel}(f_{i_j})$ and $Z := \alpha_{i_j,i_{j'}}^{-1}(\mathrm{sel}(f_{i'_j}))$ also have a nonempty intersection as $X_j = \alpha_{i_1,i_j}^{-1}(Y)$ and $X_{j'} = \alpha_{i_1,i_j}^{-1}(Z)$. [5] ◀

7 $\mathbf{T_1}$

In this section we prove the NP-hardness of PCSP($\mathbf{1in3}, \mathbf{T}_1$), where $\mathbf{T}_1 = (\{0,1,2\}, R)$ and R consists of all the permutations of the tuples $(0,0,1)$, $(0,0,2)$, and $(1,1,2)$.

Let $f : \{0,1\}^n \to \{0,1,2\}$ be a polymorphism of $(\mathbf{1in3}, \mathbf{T}_1)$. Note that, as in the previous cases, f cannot have two disjoint 2-sets. In particular, $\emptyset$ is a 0-set or a 1-set. The following simple lemma will be useful in both cases.

▶ **Lemma 20.** *If $Z \subseteq X \cup Y$, and X and Y are 1-sets, then Z is not a 2-set.*

Proof. By compatibility applied to X and Y, the complement $[n] \setminus (X \cup Y)$ is a 2-set. Since it is disjoint with Z, Z cannot be a 2-set. ◀

For the case $f(\emptyset) = 0$ we introduce some notation. We define $r : \{0,1,2\} \to \{0,1\}$ by $0 \mapsto 0$ and $1, 2 \mapsto 1$, and set

$$E(f) = \{x \in [n] : r(f(\{x\})) = 1\}, \text{ and } I(f) = \{x \in [n] : r(f(\{x\})) = 0\} = [n] \setminus E(f).$$

▶ **Lemma 21.** *The size of $E(f)$ is odd and, for any set of coordinates $X \subseteq [n]$, we have $r(f(X)) = |X \cap E(f)| \mod 2$.*

Proof. By the "union argument" as in the proof of Lemma 15, we get that for any two disjoint Y and Z, $r(f(Y \cup Z)) = r(f(Y)) + r(f(Z))$, where the addition is modulo 2. It then follows (by adding to $\emptyset$ singletons from X) that $r(f(X)) = \sum_{x \in X} r(f(\{x\})) = |X \cap E(f)| \mod 2$.

In particular, $r(f([n])) = |E(f)| \mod 2$. But $[n]$ cannot be a 0-set (by compatibility applied to $\emptyset$ and $\emptyset$), so $|E(f)|$ is odd. ◀

▶ **Lemma 22.** *Suppose that $f(\emptyset) = 0$ and f does not have any 2-sets of size 2. If X is a 1-set such that $E(f) \setminus X$ is nonempty, then $X \cup I(f)$ is a 1-set.*

[5] The last part of the argument applies Theorem 10 in a similar way as in [9], see their "smug sets" Corollary 4.2.

Proof. It is enough to show that $X \cup \{y\}$ is a 1-set for any $y \in I(f)$, as the claim then follows by induction. By Lemma 21, $X \cup \{y\}$ is a 1-set or 2-set, so it is enough to exclude the latter option. Take $z \in E(f) \setminus X$. By Lemma 21, $\{y, z\}$ is a 1-set or a 2-set, therefore it is a 1-set by the assumption. But then $X \cup \{y\}$ is not a 2-set by Lemma 20. ◀

▶ **Lemma 23.** *Assume that $f(\emptyset) = 0$ and f does not have any singleton 2-set. If $X \subseteq E(f)$ is a 1-set, then, for any $Y \subseteq E(f)$ with $|Y| = |X|$, Y is a 1-set.*

Proof. We will show that $Z := (X \setminus \{x\}) \cup \{y\}$ is a 1-set for any $x \in X$ and $y \in E(f)$. The claim will then follow by induction since any set Y can be obtained from X by a sequence of such "swaps".

By Lemma 21 and the non-existence of singleton 2-sets, $|X|$ is odd, Z is a 1-set or 2-set, and $\{y\}$ is a 1-set (it cannot be a 2-set by the assumption of the lemma). Since $Z \subseteq X \cup \{y\}$, then Z is a 1-set by Lemma 20. ◀

▶ **Lemma 24.** *If $f(\emptyset) = 0$ and f does not have any 2-sets of size at most 2, then $|E(f)| \leq 5$.*

Proof. We first observe that any $X \subseteq E(f)$ of odd size $|X| \leq |E(f)|/2$ is a 1-set. Indeed, otherwise we can find Y disjoint from X of the same size. By Lemma 21 and Lemma 23, both X and X' are 2-sets, a contradiction.

By Lemma 21, the size $i := |E(f)|$ is odd. If $i > 5$, then $E(f)$ can be written as a disjoint union $E(f) = X \cup Y \cup Z$ of sets that have odd sizes smaller than $|E(f)|/2$. By the previous paragraph, all of these sets are 1-sets. But then, by Lemma 22, $Z \cup I(f) = [n] \setminus (X \cup Y)$ is a 1-set as well, a contradiction to compatibility applied to X and Y. ◀

We now consider the case that $\emptyset$ is a 1-set. Observe that $[n]$ is a 2-set in this case.

▶ **Lemma 25.** *If $f(\emptyset) = 1$, then f has a 2-set of size at most 2.*

Proof. Assume, for a contradiction, that there are no 2-sets of size at most 2.

Union arguments in the case $f(\emptyset) = 1$ give us $r(f(Y \cup Z)) = r(f(X)) + r(f(Y)) + 1 \pmod 2$ and, as in Lemma 21, we obtain that $X = \{x \in [n] : f(\{x\}) = 0\}$ has an odd size and that, using additionally the "no two-element 2-sets" assumption, every two-element subset of X is a 1-set.

Note that if Y and Z are disjoint 1-sets, then the union argument gives us a sharper result – $Y \cup Z$ is a 1-set. It follows that $X \setminus \{x\}$, where $x \in X$ is an arbitrary element, is a 1-set (as it is a disjoint union of 2-element subsets of X) and $[n] \setminus X$ is a 1-set (as it is a disjoint union of singletons outside X, which are 1-sets by the "no singleton 2-set" assumption). Now compatibility applied to $X \setminus \{x\}$ and $[n] \setminus X$ gives us that $\{x\}$ is a 2-set, a contradiction. ◀

▶ **Theorem 26.** $\mathrm{PCSP}(\mathbf{1in3}, \mathbf{T}_1)$ *is NP-hard.*

Proof. We apply Theorem 10 with $k = 5$ and $l = 2$. We assign to a polymorphism its *type* and define $\mathrm{sel}(f)$ as follows.

- Type 1: f has a 2-set X of size at most 2. In this case we set $\mathrm{sel}(f) = X$.
- Type 2: f has no 2-set of size at most 2. In this case we set $\mathrm{sel}(f) = E(f)$.

Note that $\mathrm{sel}(f)$ in type 1 is nonempty. Type 2 only occurs in the case that $f(\emptyset) = 0$ (by Lemma 25), and then $E(f)$ has size at most 5 by Lemma 24.

Let $(f_0, \alpha_{0,1}, f_1, \alpha_{0,2}, f_2)$ be a chain of minors consisting of polymorphisms. If both f_i and f_j (where $i < j$) have type 1, then $\mathrm{sel}(f_i)$ and $\alpha_{i,j}^{-1}(\mathrm{sel}(f_j))$ are both 2-sets, so they have a nonempty intersection. Otherwise, since $l = 2$, the chain contains 2 polymorphisms

f_i, f_j of type 2 (where $i < j$). We have $f_i(\emptyset) = f_j(\emptyset) = 0$ and $\alpha_{i,j}^{-1}(\mathrm{sel}(f_j))$ is a 1-set of f_i. By Lemma 21, this 1-set has an odd-sized intersection with $E(f_i) = \mathrm{sel}(f_i)$, in particular $\mathrm{sel}(f_i) \cap \alpha_{i,j}^{-1}(\mathrm{sel}(f_j)) \neq \emptyset$. ◀

8 Conclusion

The investigation of PCSPs over the templates $(\mathbf{A}, \mathbf{B})$, with $\mathbf{A}$ a Boolean structure consisting of a single ternary symmetric relation, boils down to $\mathrm{PCSP}(\mathbf{LO}_2, \mathbf{B})$ where $\mathbf{B}$ is symmetric. We have classified the computational complexity for all such three-element structures $\mathbf{B}$ with the exception of $\mathbf{B} = \mathbf{LO}_3$. The remaining case, and its generalization to larger domains, is a natural computational problem – recall the interpretation as a version of hypergraph coloring from the introduction. We conjecture that all of them are NP-hard.

▶ **Conjecture 27.** *For every* $2 \leq k < l$, $\mathrm{PCSP}(\mathbf{LO}_k, \mathbf{LO}_l)$ *is NP-hard.*

A possible intermediate step to resolving the smallest unknown case, $\mathrm{PCSP}(\mathbf{LO}_2,\ \mathbf{LO}_3)$, is to replace $\mathbf{LO}_2$ by a 3-element structure in the interval between $\mathbf{1in3}$ and $\mathbf{LO}_3$.

For four-element structures, the remaining cases additionally include the structures in the interval between $\check{\mathbf{C}}$ and $\check{\mathbf{C}}^{+}$. We proved NP-hardness for $\check{\mathbf{C}}$ and provided evidence suggesting that $\check{\mathbf{C}}^{+}$ also gives rise to an NP-hard PCSP:

▶ **Conjecture 28.** $\mathrm{PCSP}(\mathbf{1in3}, \check{\mathbf{C}}^{+})$ *is NP-hard.*

Negative resolution of this conjecture would also be valuable – it would require a polynomial-time algorithm that has not yet been used for PCSPs.

Observe that a homomorphism from a 3-uniform hypergraph to $\check{\mathbf{C}}^{+}$ also has a nice interpretation, as a coloring by 4-colors such that if two vertices of an hyperedge receive the same color, the last vertex must receive a color which is one higher (mod 4). Other templates admit a natural interpretation and generalizations as well, e.g., $\mathbf{B} = \mathbf{D}_2$.

Finally, it seems possible that $(\mathbf{1in3}, \mathbf{NAE})$ and $(\mathbf{1in3}, \mathbf{T}_2)$ are essentially the only tractable templates for arbitrary domain sizes. We do not feel that we have enough evidence supporting such a conjecture, so we refrain from phrasing it.

References

1 Per Austrin, Venkatesan Guruswami, and Johan Håstad. $(2+\epsilon)$-Sat is NP-hard. *SIAM J. Comput.*, 46(5):1554–1573, 2017. doi:10.1137/15M1006507.

2 L. Barto. Promises make finite (constraint satisfaction) problems infinitary. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–8, 2019.

3 Libor Barto, Jakub Bulín, Andrei Krokhin, and Jakub Opršal. Algebraic approach to promise constraint satisfaction, 2019. arXiv:1811.00970.

4 Libor Barto, Andrei Krokhin, and Ross Willard. Polymorphisms, and how to use them. In Andrei Krokhin and Stanislav Živný, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 1–44. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2017. doi:10.4230/DFU.Vol7.15301.1.

5 Libor Barto, Jakub Opršal, and Michael Pinsker. The wonderland of reflections. *Israel Journal of Mathematics*, 223(1):363–398, February 2018. doi:10.1007/s11856-017-1621-9.

6 Joshua Brakensiek and Venkatesan Guruswami. Promise constraint satisfaction: Structure theory and a symmetric boolean dichotomy. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'18, pages 1782–1801, Philadelphia, PA, USA, 2018. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=3174304.3175422.

7 Joshua Brakensiek and Venkatesan Guruswami. Symmetric polymorphisms and efficient decidability of promise CSPs. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '20, page 297–304, USA, 2020. Society for Industrial and Applied Mathematics.

8 Joshua Brakensiek, Venkatesan Guruswami, Marcin Wrochna, and Stanislav Živný. The power of the combined basic linear programming and affine relaxation for promise constraint satisfaction problems. *SIAM Journal on Computing*, 49(6):1232–1248, 2020. doi:10.1137/20M1312745.

9 Alex Brandts, Marcin Wrochna, and Stanislav Živný. The Complexity of Promise SAT on Non-Boolean Domains. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:13, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.17.

10 Andrei Bulatov, Peter Jeavons, and Andrei Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.*, 34(3):720–742, March 2005. doi:10.1137/S0097539700376676.

11 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 319–330, October 2017. doi:10.1109/FOCS.2017.37.

12 Jakub Bulín, Andrei Krokhin, and Jakub Opršal. Algebraic approach to promise constraint satisfaction. In *Proceedings of the 51st Annual ACM SIGACT Symposium on the Theory of Computing (STOC '19)*, New York, NY, USA, 2019. ACM. doi:10.1145/3313276.3316300.

13 Irit Dinur, Oded Regev, and Clifford Smyth. The hardness of 3-uniform hypergraph coloring. *Combinatorica*, 25(5):519–535, September 2005. doi:10.1007/s00493-005-0032-4.

14 Miron Ficak, Marcin Kozik, Miroslav Olšák, and Szymon Stankiewicz. Dichotomy for Symmetric Boolean PCSPs. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 57:1–57:12, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2019.57.

15 Pavol Hell and Jaroslav Nešetřil. On the complexity of H-coloring. *J. Combin. Theory Ser. B*, 48(1):92–110, 1990.

16 Peter Jeavons. On the algebraic structure of combinatorial problems. *Theor. Comput. Sci.*, 200(1-2):185–204, 1998.

17 A. Krokhin and J. Opršal. The complexity of 3-colouring H-colourable graphs. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1227–1239, 2019. doi:10.1109/FOCS.2019.00076.

18 Lászlo Lovász. Kneser's conjecture, chromatic number, and homotopy. *J. Combin. Theory Ser. A*, 25(3):319–324, 1978. doi:10.1016/0097-3165(78)90022-5.

19 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 216–226, New York, NY, USA, 1978. ACM. doi:10.1145/800133.804350.

20 Marcin Wrochna. personal communication.

21 Dmitriy Zhuk. A proof of the CSP dichotomy conjecture. *J. ACM*, 67(5), August 2020. doi:10.1145/3402029.

A $\check{\mathbf{C}}$

In this appendix we begin the proof of Theorem 2 by verifying the first statement: the NP-hardness of PCSP($\mathbf{1in3}, \check{\mathbf{C}}$), where $\check{\mathbf{C}} = (\{0,1,2,3\}, R)$ and R consists of all the permutations of the tuples $(0,0,1)$, $(1,1,2)$, $(2,2,3)$, and $(0,3,3)$.

Before applying Theorem 10, we begin by deriving several properties of polymorphisms of the template. Let $f : \{0,1\}^n \to \{0,1,2,3\}$ be a polymorphism of $(\mathbf{1in3}, \check{\mathbf{C}})$.

▶ **Lemma 29.** *Suppose $f(\emptyset) = i$.*

(a) *f has no $(i+2 \mod 4)$-sets.*

(b) *f has no two disjoint $(i+1 \mod 4)$-sets.*

Proof. Suppose $f(\emptyset) = 0$. If $X \subseteq [n]$ were a 2-set, then by compatibility with $\emptyset$ it would be the case that $[n] \setminus X$ is compatible with a 0-set and a 2-set. There are no such sets, proving item (a) for this value. Furthermore, if $X \subseteq [n]$ and $Y \subseteq [n]$ are both disjoint 1-sets, then by compatibility with X and Y, $[n] \setminus (X \cup Y)$ is a 2-set, but there are no such sets, proving item (b) for this value.

The proof for the remaining values of i is similar. ◀

Union arguments (see the proof of Lemma 15) give us the following properties.

▶ **Lemma 30.** *Let X and Y be disjoint subsets of $[n]$.*

(a) *If $f(\emptyset) = f(X) = f(Y) = i$, then $f(X \cup Y) = i$.*

(b) *If $f(\emptyset) = i$ and $f(X) = f(Y) = i+3 \mod 4$, then $f(X \cup Y) = i+1 \mod 4$.*

Finally, we prove a lemma about small i-sets which will facilitate our main argument for this appendix.

▶ **Lemma 31.** *If $f(\emptyset) = i$, and f has no $(i+3 \mod 4)$-set with size at most 2, then there exists a singleton $(i+1 \mod 4)$-set.*

Proof. We will consider the case where $f(\emptyset) = 0$, as proofs for other values of i will be similar. Observe that in this case, $[n]$ is a 1-set by compatibility applied to $\emptyset$ and $\emptyset$. Suppose by way of contradiction that no such $y \in [n]$ exists. It must then be the case that every singleton is a 0-set. However, by adding to $\emptyset$ singletons, one by one, and using item (a) of Lemma 30, we get that $[n]$ is a 0-set, a contradiction. ◀

Equipped with these lemmata, we can now proceed to our main argument for this appendix.

▶ **Theorem 32.** $\mathrm{PCSP}(\mathbf{1in3}, \check{\mathbf{C}})$ *is NP-hard.*

Proof. We apply Theorem 10 with $k = 2$ and $l = 5$. We assign to a polymorphism with $f(\emptyset) = i$ its *type* and define $\mathrm{sel}(f)$ as follows.

- Type 1: f has a $(i+3 \mod 4)$-set X of size at most 2. In this case we set $\mathrm{sel}(f) = X$.
- Type 2: f does not have a $(i+3 \mod 4)$-set of size at most 2 and f has a singleton $(i+1 \mod 4)$-set $\{x\}$. We set $\mathrm{sel}(f) = \{x\}$.

Lemma 31 guarantees that every polymorphism is of one of the two types.

Let $(f_0, \alpha_{0,1}, \ldots, f_l)$ be a chain of minors consisting of polymorphisms and note that the value at $\emptyset$ is constant throughout the chain. For simplicity, let this value be 0. If, for some $i < j$, both f_i and f_j have type 2, then $\mathrm{sel}(f_i)$ and $\alpha_{i,j}^{-1}(\mathrm{sel}(f_j))$ are both 1-sets, so they have a nonempty intersection by item (b) of Lemma 29.

Otherwise, since $l = 5$, the chain contains four polymorphisms $f_{i_1}, f_{i_2}, f_{i_3}, f_{i_4}$ of type 1 (where $i_1 < i_2 < i_3 < i_4$). Let $X_1 = \mathrm{sel}(f_{i_1})$ and $X_j = \alpha_{i_1,i_j}^{-1}(\mathrm{sel}(f_{i_j}))$ for $j = 2, 3, 4$.

These four sets are 3-sets (as preimages of 3-sets). If they are pairwise disjoint, then $X_1 \cup X_2$ and $X_3 \cup X_4$ are disjoint sets, which are 1-sets by item (b) in Lemma 30, a contradiction with item (b) of Lemma 29.

Therefore, two of these sets, say X_j and $X_{j'}$, have a nonempty intersection. But then $Y := \mathrm{sel}(f_{i_j})$ and $Z := \alpha^{-1}_{i_j,i_{j'}}(\mathrm{sel}(f_{i'_j}))$ also have a nonempty intersection as $X_j = \alpha^{-1}_{i_1,i_j}(Y)$ and $X_{j'} = \alpha^{-1}_{i_1,i_j}(Z)$. ◀

B $\check{\mathbf{C}}^{+}$

Recall that $\check{\mathbf{C}}^{+} = (\{0,1,2,3\}, R)$, where R consists of all the permutations of the tuples $(0,0,1)$, $(1,1,2)$, $(2,2,3)$, and $(0,3,3)$, as well as all the "rainbow" tuples (i,j,k) such that $|\{i,j,k\}| = 3$. In this appendix we show that there is no block symmetric polymorphism of $(\mathbf{1in3}, \check{\mathbf{C}}^{+})$ with two blocks of sizes 23 and 24.

▶ **Lemma 33.** *If $g : \{0,1\}^{47} \to \{0,1,2,3\}$ is a block symmetric polymorphism of $(\mathbf{1in3}, \check{\mathbf{C}}^{+})$ with two blocks of sizes 23 and 24, then there exists a symmetric polymorphism $f : \{0,1\}^{23} \to \{0,1,2,3\}$ of $(\mathbf{1in3}, \check{\mathbf{C}}^{+})$.*

Proof. We will define $f : \{0,1\}^{23} \to \{0,1,2,3\}$ based on the symmetric blocks of g, which we name X_{23} and X_{24} in accordance with their sizes. For any $X \subseteq [23]$, we set $f(X) = g(Y \cup Z)$, where $Y \subseteq X_{23}$ with $|Y| = |X|$ and $Z \subseteq X_{24}$ with $|Z| = 8$. By construction, then, f is a symmetric polymorphism of $(\mathbf{1in3}, \check{\mathbf{C}}^{+})$. ◀

▶ **Theorem 34.** *There is no block symmetric polymorphism of $(\mathbf{1in3}, \check{\mathbf{C}}^{+})$ with two blocks of sizes 23 and 24.*

Proof. Suppose by way of contradiction that there is such a polymorphism, say $g : \{0,1\}^{47} \to \{0,1,2,3\}$. Let $f : \{0,1\}^{23} \to \{0,1,2,3\}$ be a symmetric polymorphism of $(\mathbf{1in3}, \check{\mathbf{C}}^{+})$, guaranteed by the previous lemma.

Since f is symmetric, we adopt the convention that $f(m)$ is the value of $f(X)$ for any $X \subseteq [n]$ with $|X| = m$. Assume now that $f(8) = 0$ – our argument will be constructed such that other choices for the value of $f(8)$ can be carried forward to likewise achieve a contradiction. By compatibility with $f(8)$ and $f(8)$, we have then that $f(7) = 1$, and similarly by compatibility with $f(7)$ and $f(7)$ it must be the case that $f(9) = 2$. Since $f(9) = 2$, by compatibility with $f(9)$ and $f(9)$ it follows that $f(5) = 3$. In turn, by compatibility with $f(5)$ and $f(5)$, we get that $f(13) = 0$. Since $f(8) = f(13) = 0$, it must then be the case by compatibility that $f(2) = 1$. Therefore, since $f(14)$ is compatible with $f(7) = 1$ and $f(2) = 1$, we get that $f(14) = 2$. Since $f(9) = f(14) = 2$, it follows in turn by compatibility that $f(0) = 3$.

Consider now the possible values of $f(6)$. If $f(6) = 0$, then $f(9) = 1$ by compatibility with $f(6)$ and $f(8)$, but it has already been shown that $f(9) = 2$, a contradiction. If $f(6) = 2$, then $f(8) = 3$ by compatibility with $f(6)$ and $f(9)$, but by our initial assumption, $f(8) = 0$, a contradiction. If $f(6) = 1$, then $f(11) = 2$ by compatibility with $f(6)$ and $f(6)$, and $f(10) = 2$ by compatibility with $f(6)$ and $f(7) = 1$. However, it must then be the case by compatibility with $f(10)$ and $f(11)$ that $f(2) = 3$, but it has already been shown that $f(2) = 1$, a contradiction. Finally, if $f(6) = 3$, then $f(11) = 0$ by compatibility with $f(6)$ and $f(6)$. Similarly, by compatibility with $f(5)$ and $f(6)$, we get that $f(12) = 0$. But $f(0) = 3$ and is compatible with $f(11) = f(12) = 0$, which is a contradiction since no permutation of $(0,0,3)$ is in R. Therefore, no value of $f(6)$ is possible, and thus no such g exists. ◀

This theorem, together with Theorem 32, completes the proof of Theorem 2.

A Characterization of Wreath Products Where Knapsack Is Decidable

Pascal Bergsträßer
Fachbereich Informatik, Technische Universität Kaiserslautern, Germany

Moses Ganardi
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Georg Zetzsche
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Abstract

The knapsack problem for groups was introduced by Miasnikov, Nikolaev, and Ushakov. It is defined for each finitely generated group G and takes as input group elements $g_1, \ldots, g_n, g \in G$ and asks whether there are $x_1, \ldots, x_n \geq 0$ with $g_1^{x_1} \cdots g_n^{x_n} = g$. We study the knapsack problem for wreath products $G \wr H$ of groups G and H.

Our main result is a characterization of those wreath products $G \wr H$ for which the knapsack problem is decidable. The characterization is in terms of decidability properties of the indiviual factors G and H. To this end, we introduce two decision problems, the *intersection knapsack problem* and its restriction, the *positive intersection knapsack problem*.

Moreover, we apply our main result to $H_3(\mathbb{Z})$, the discrete Heisenberg group, and to Baumslag-Solitar groups $\mathsf{BS}(1,q)$ for $q \geq 1$. First, we show that the knapsack problem is undecidable for $G \wr H_3(\mathbb{Z})$ for any $G \neq 1$. This implies that for $G \neq 1$ and for infinite and virtually nilpotent groups H, the knapsack problem for $G \wr H$ is decidable if and only if H is virtually abelian and solvability of systems of exponent equations is decidable for G. Second, we show that the knapsack problem is decidable for $G \wr \mathsf{BS}(1,q)$ if and only if solvability of systems of exponent equations is decidable for G.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Theory and algorithms for application domains

Keywords and phrases knapsack, wreath products, decision problems in group theory, decidability, discrete Heisenberg group, Baumslag-Solitar groups

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.11

Related Version This paper is based on the following technical report:
Full Version: https://arxiv.org/abs/2101.06132 [5]

1 Introduction

The knapsack problem. The knapsack problem is a decision problem for groups that was introduced by Myasnikov, Nikolaev, and Ushakov [27]. If G is a finitely generated group, then the knapsack problem for G, denoted $\mathsf{KP}(G)$, takes group elements $g_1, \ldots, g_n, g \in G$ as input (as words over the generators) and it asks whether there are natural numbers $x_1, \ldots, x_n \geq 0$ such that $g_1^{x_1} \cdots g_n^{x_n} = g$. Since its introduction, a significant amount of attention has been devoted to understanding for which groups the problem is decidable and what the resulting complexity is [17, 20, 10, 26, 16, 9, 21, 7]. For matrix semigroups, the knapsack problem has been studied implicitly by Bell, Halava, Harju, Karhumäki, and Potapov [3], Bell, Potapov, and Semukhin [4], and for commuting matrices by Babai, Beals, Cai, Ivanyos, and Luks [1].

There are many groups for which knapsack has been shown decidable. For example, knapsack is decidable for virtually special groups [20, Theorem 3.1], co-context-free groups [16, Theorem 8.1], hyperbolic groups [27, Theorem 6.1], the discrete Heisenberg group [16,

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 11; pp. 11:1–11:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Theorem 6.8], and Baumslag-Solitar groups $\mathsf{BS}(p,q)$ for co-prime $p,q>1$ [6, Theorem 2] and for $p=1$ [21, Theorem 4.1]. Moreover, the class of groups where knapsack is decidable is closed under free products with amalgamation [19, Theorem 14] and HNN extensions [19, Theorem 13] over finite identified subgroups. On the other hand, there are nilpotent groups for which knapsack is undecidable [16, Theorem 6.5].

Wreath products. A prominent construction in group theory and semigroup theory is the wreath product $G \wr H$ of two groups G and H. Wreath products are important algorithmically, because the Magnus embedding theorem [22, Lemma] states that for any free group F of rank r and a normal subgroup N of F, one can find $F/[N,N]$ as a subgroup of $\mathbb{Z}^r \wr (F/N)$, where $[N,N]$ is the commutator subgroup of N. This has been used by several authors to obtain algorithms for groups of the form $F/[N,N]$, and in particular free solvable groups. Examples include the word problem (folklore, see [13]), the conjugacy problem [24, 28, 13, 25], the power problem [13], and the knapsack problem [7, 10].

For groups G and H, their wreath product $G \wr H$ can be roughly described as follows. An element of $G \wr H$ consists of (i) a labeling, which maps each element of H to an element of G and (ii) an element of H, called the *cursor*. Here, the labeling has finite support, meaning all but finitely many elements of H are mapped to the identity of G. Moreover, each element of $G \wr H$ can be written as a product of elements from G and from H. Multiplying an element $g \in G$ will multiply g to the label of the current cursor position. Multiplying an element $h \in H$ will move the cursor by multiplying h.

Understanding the knapsack problem for wreath products is challenging for two reasons. First, the path that the expression $g_1^{x_1} \cdots g_n^{x_n} g^{-1}$ takes through the group H can have complicated interactions with itself: The product can place elements of G at (an *a priori* unbounded number of) positions $h \in H$ that are later revisited. At the end of the path, each position of H must carry the identity of G so as to obtain $g_1^{x_1} \cdots g_k^{x_k} g^{-1} = 1$. The second reason is that the groups G and H play rather different roles: *A priori*, for each group G the class of all H with decidable $\mathsf{KP}(G \wr H)$ could be different, resulting in a plethora of cases.

Decidability of the knapsack problem for wreath products has been studied by Ganardi, König, Lohrey, and Zetzsche [10]. They focus on the case that H is knapsack-semilinear, which means that the solution sets of equations $g_1^{x_1} \cdots g_n^{x_n} = g$ are (effectively) semilinear. A set $S \subseteq \mathbb{N}^n$ is *semilinear* if it is a finite union of *linear* sets $\{u_0+\lambda_1 u_1+\cdots+\lambda_k u_k \mid \lambda_1,\ldots,\lambda_k \in \mathbb{N}\}$ for some vectors $u_0,\ldots,u_k \in \mathbb{N}^n$. Under this assumption, they show that $\mathsf{KP}(G \wr H)$ is decidable if and only if solvability of systems of exponent equations is decidable for G [10, Theorem 5.3]. Here, an exponent equation is one of the form $g_1^{x_1} \cdots g_n^{x_n} = g$, where variables x_i are allowed to repeat. The problem of solvability of systems of exponent equations is denoted $\mathsf{ExpEq}(G)$. Moreover, it is shown there that for some number $\ell \in \mathbb{N}$, knapsack is undecidable for $G \wr (H_3(\mathbb{Z}) \times \mathbb{Z}^\ell)$, where $H_3(\mathbb{Z})$ denotes the discrete Heisenberg group and G is any non-trivial group [10, Theorem 5.2]. Since $\mathsf{KP}(H_3(\mathbb{Z}) \times \mathbb{Z}^\ell)$ is decidable for any $\ell \geq 0$ [16, Theorem 6.8], this implies that wreath products do not preserve decidability of knapsack in general. However, apart from the latter undecidability result, little is known about wreath products $G \wr H$ where H is not knapsack-semilinear. As notable examples of this, knapsack is decidable for solvable Baumslag-Solitar groups $\mathsf{BS}(1,q)$ [21, Theorem 4.1] and for the discrete Heisenberg group $H_3(\mathbb{Z})$ [16, Theorem 6.8], but it is not known for which G the knapsack problem is decidable for $G \wr H_3(\mathbb{Z})$ or for $G \wr \mathsf{BS}(1,q)$.

The only other paper which studies the knapsack problem over wreath products is [7]. It is concerned with complexity results (for knapsack-semilinear groups) whereas in this paper we are concerned with decidability results.

Contribution. Our main result is a characterization of the groups G and H for which $\mathsf{KP}(G \wr H)$ is decidable. Specifically, we introduce two problems, *intersection knapsack* $\mathsf{KP}^{\pm}(H)$ and the variant *positive intersection knapsack* $\mathsf{KP}^{+}(H)$ and show the following. Let G and H be finitely generated, with G non-trivial and H infinite. Then knapsack for $G \wr H$ is decidable if and only if $\mathsf{ExpEq}(G)$ is decidable and either (i) G is abelian and $\mathsf{KP}^{+}(H)$ is decidable or (ii) G is not abelian and $\mathsf{KP}^{\pm}(H)$ is decidable. Note that the case of finite H is not interesting: For $|H| = m$, $\mathsf{KP}(G \wr H)$ is equivalent to $\mathsf{KP}(G^m)$ (see Section 3).

Thus, our result relieves us from considering every pair (G, H) of groups and allows us to study the factors separately. It is not hard to see that decidability of $\mathsf{ExpEq}(G)$ is necessary for decidability of $\mathsf{KP}(G \wr H)$ if H is infinite. It is surprising that the only other property of G that is relevant for decidability of $\mathsf{KP}(G \wr H)$ is whether G is abelian or not. This is in contrast to the effect of other structural properties of G on the complexity of $\mathsf{KP}(G \wr \mathbb{Z})$: If $G \neq 1$ is a finite nilpotent group, then $\mathsf{KP}(G \wr \mathbb{Z})$ is NP-complete [7, Theorem 2], whereas for finite and non-solvable G, the problem $\mathsf{KP}(G \wr \mathbb{Z})$ is Σ_2^p-complete [7, Corollary 25].

Applications. We also obtain two applications. First, we deduce that $\mathsf{KP}(G \wr H_3(\mathbb{Z}))$ is undecidable for every $G \neq 1$. This implies that if $G \neq 1$ and H is virtually nilpotent and infinite, then $\mathsf{KP}(G \wr H)$ is decidable if and only if H is virtually abelian and $\mathsf{ExpEq}(G)$ is decidable. Moreover, we show that $\mathsf{KP}(G \wr \mathsf{BS}(1, q))$ is decidable if and only if $\mathsf{ExpEq}(G)$ is.

Ingredients. For the "if" direction of our main result, we reduce $\mathsf{KP}(G \wr H)$ to $\mathsf{ExpEq}(G)$ and $\mathsf{KP}^{\pm}(H)$ (respectively $\mathsf{KP}^{+}(H)$) using extensions of techniques used by Figelius, Ganardi, Lohrey, and Zetzsche [7]. Roughly speaking, the problem $\mathsf{KP}^{\pm}(H)$ takes as input an expression $h_0 g_1^{x_1} h_1 \cdots g_n^{x_n} h_n$ and looks for numbers $x_1, \ldots, x_n \geq 0$ such that the walk defined by the product $h_0 g_1^{x_1} h_1 \cdots g_n^{x_n} h_n$ meets specified constraints about self-intersections. Such a constraint can be either (i) a *loop constraint*, meaning the walk visits the same point after two specified factors or (ii) a *disjointness constraint* saying that the $(x_i + 1)$-many points visited when multiplying $g_i^{x_i}$ do not intersect the $(x_j + 1)$-many points visited while multiplying $g_j^{x_j}$.

The "only if" reductions in our main result involve substantially new ideas. The challenge is to guarantee that the constructed instances of $\mathsf{KP}(G \wr H)$ will leave an element $\neq 1$ somewhere, as soon as any constraint is violated. In particular, the loop constraints have to be checked independently of the disjointness constraints. Moreover, if several constraints are violated, the resulting elements $\neq 1$ should not cancel each other. Furthermore, this has to be achieved despite almost no information on the structure of G and H. This requires an intricate construction that uses various patterns in the Cayley graph of H for which we show that only very specific arrangements permit cancellation. To this end, we introduce the notion of *periodic complexity*, which measures how many periodic sequences are needed to cancel out a sequence of elements of a group. Roughly speaking, for the loop constraints we use patterns of high periodic complexity, whereas for the disjointness constraints we use patterns with low periodic complexity but many large gaps. This ensures that the disjointness patterns cannot cancel the loop patterns or vice versa.

2 Preliminaries

Knapsack problems. For a group G and a subset $S \subseteq G$ we write S^* for the submonoid generated by S, i.e. the set of products of elements from S. Let G be a group with a finite *(monoid) generating set* $\Sigma \subseteq G$, i.e. $G = \Sigma^*$. Such groups are called *finitely generated.* An *exponent expression* over G is an expression $E = e_1 \ldots e_n$ consisting of *atoms* e_i where each atom e_i is either a *constant* $e_i = g_i \in G$ or a *power* $e_i = g_i^{x_i}$ for some $g_i \in G$ and variable x_i. Here the group elements g_i are given as words over Σ. We write $\gamma(e_i) = g_i$ for the constant or the base of the power. Furthermore let $P_E \subseteq [1, n]$ be the set of indices of the powers in E and $Q_E = [1, n] \setminus P_E$ be the set of indices of the constants in E. If $\nu \in \mathbb{N}^X$ is a valuation of the variables X that occur in E, then for each $i \in [1, n]$, we define $\nu(e_i) = \gamma(e_i)^{\nu(x_i)}$ if $i \in P_E$; and $\nu(e_i) = e_i$ if $i \in Q_E$. Moreover, $\nu(E) := \nu(e_1) \cdots \nu(e_n)$ and the set of G-*solutions* of E as $\mathsf{sol}_G(E) := \{\nu \in \mathbb{N}^X \mid \nu(E) = 1\}$.

For a group G, the problem of *solvability of exponent equations* $\mathsf{ExpEq}(G)$ is defined as:

Given a finite list of exponent expression $E_1, \ldots, E_k$ over G.

Question Is $\bigcap_{i=1}^{k} \mathsf{sol}_G(E_i)$ non-empty?

An exponent expression is called a *knapsack expression* if all variables occur at most once. The *knapsack problem* $\mathsf{KP}(G)$ over G is defined as follows:

Given a knapsack expression E over G.

Question Is there a valuation ν such that $\nu(E) = 1$?

The definition from [27] asks whether $g_1^{x_1} \cdots g_n^{x_n} = g$ has a solution for given $g_1, \ldots, g_n, g \in G$. The two versions are inter-reducible in polynomial time [16, Proposition 7.1].

Wreath products. Let G and H be groups. Consider the direct sum $K = \bigoplus_{h \in H} G_h$, where G_h is a copy of G. We view K as the set $G^{(H)}$ of all mappings $f\colon H \to G$ such that $\mathsf{supp}(f) := \{h \in H \mid f(h) \neq 1\}$ is finite, together with pointwise multiplication as the group operation. The set $\mathsf{supp}(f) \subseteq H$ is called the *support* of f. The group H has a natural left action on $G^{(H)}$ given by ${}^{h}f(a) = f(h^{-1}a)$, where $f \in G^{(H)}$ and $h, a \in H$. The corresponding semidirect product $G^{(H)} \rtimes H$ is the (restricted) *wreath product* $G \wr H$. In other words:

- Elements of $G \wr H$ are pairs (f, h), where $h \in H$ and $f \in G^{(H)}$.
- The multiplication in $G \wr H$ is defined as follows: Let $(f_1, h_1), (f_2, h_2) \in G \wr H$. Then $(f_1, h_1)(f_2, h_2) = (f, h_1h_2)$, where $f(a) = f_1(a)f_2(h_1^{-1}a)$.

There are canonical mappings $\sigma\colon G \wr H \to H$ with $\sigma(f, h) = h$ and $\tau\colon G \wr H \to G^{(H)}$ with $\tau(f, h) = f$ for $f \in G^{(H)}$, $h \in H$. In other words: $g = (\tau(g), \sigma(g))$ for $g \in G \wr H$. Note that σ is a homomorphism whereas τ is in general not a homomorphism. Throughout this paper, the letters σ and τ will have the above meaning (the groups G, H will be always clear from the context). We also define $\mathsf{supp}(g) = \mathsf{supp}(\tau(g))$ for all $g \in G \wr H$.

The following intuition might be helpful: An element $(f, h) \in G \wr H$ can be thought of as a finite multiset of elements of $G \setminus \{1_G\}$ that are sitting at certain elements of H (the mapping f) together with the distinguished element $h \in H$, which can be thought of as a *cursor* moving in H. We can compute the product $(f_1, h_1)(f_2, h_2)$ as follows: First, we shift the finite collection of G-elements that corresponds to the mapping f_2 by h_1: If the element $g \in G \setminus \{1_G\}$ is sitting at $a \in H$ (i.e., $f_2(a) = g$), then we remove g from a and put it to the new location $h_1a \in H$. This new collection corresponds to the mapping $f_2'\colon a \mapsto f_2(h_1^{-1}a)$. After this shift, we multiply the two collections of G-elements pointwise: If $g_1 \in G$ and $g_2 \in G$ are sitting at $a \in H$ (i.e., $f_1(a) = g_1$ and $f_2'(a) = g_2$), then we put g_1g_2 into the location a. The new distinguished H-element (the new cursor position) becomes h_1h_2.

Clearly, H is a subgroup of $G \wr H$. We also regard G as a subgroup of $G \wr H$ by identifying G with the set of all $f \in G^{(H)}$ with $\mathsf{supp}(f) \subseteq \{1\}$. This copy of G together with H generates $G \wr H$. In particular, if $G = \langle \Sigma \rangle$ and $H = \langle \Gamma \rangle$ with $\Sigma \cap \Gamma = \emptyset$ then $G \wr H$ is generated by $\Sigma \cup \Gamma$. With these embeddings, GH is the set of $(f, h) \in G \wr H$ with $\mathsf{supp}(f) \subseteq \{1\}$ and $h \in H$.

Groups. Our applications will involve two well-known types of groups: the *discrete Heisenberg group* $H_3(\mathbb{Z})$, which consists of the matrices $\left(\begin{smallmatrix} 1 & a & c \\ 0 & 1 & b \\ 0 & 0 & 1 \end{smallmatrix}\right)$ with $a, b, c \in \mathbb{Z}$, and the *Baumslag-Solitar groups* [2] $\mathsf{BS}(p,q)$ for $p, q \in \mathbb{N}$, where $\mathsf{BS}(p,q) = \langle a, t \mid ta^p t^{-1} = a^q \rangle$.

A subgroup H of G is called *finite-index* if there are finitely many cosets gH. If $ab = ba$ for every $a, b \in G$, then G is *abelian*. A group has a property *virtually* if it has a finite-index subgroup H with that property. For example, a group is virtually abelian if it has a finite-index abelian subgroup. For two elements $a, b \in G$, we write $[a, b] = aba^{-1}b^{-1}$ and call this the *commutator* of a, b. If A, B are subgroups of G, then $[A, B]$ is the subgroup generated by all $[a, b]$ with $a \in A$ and $b \in B$. For $g, h \in G$, we write ${}^h g = hgh^{-1}$. In particular, if $g \in G$ and $h \in H$, then ${}^h g$ is the element $(f, 1) \in G \wr H$ with $f(h) = g$ and $f(h') = 1$ for $h' \neq h$.

3 Main results

We first introduce the new (positive) intersection knapsack problem. A solution to a knapsack expression E describes a walk in the Cayley graph that starts and ends in the group identity. Whereas the ordinary knapsack problem only asks for the expression to yield the identity, our extended version can impose constraints on how this walk intersects itself.

A *walk* over G is a nonempty sequence $\pi = (g_1, \dots, g_n)$ over G. Its support is $\mathsf{supp}(\pi) = \{g_1, \dots, g_n\}$. It is a *loop* if $g_1 = g_n$. Two walks are *disjoint* if their supports are disjoint. We define a partial concatenation on walks: If $\pi = (g_1, \dots, g_n)$ and $\rho = (h_1, \dots, h_m)$ with $g_n = h_1$ then $\pi\rho = (g_1, \dots, g_n, h_2, \dots, h_m)$. A *progression* with period $h \in G$ over G is a walk of the form $\pi = (g, gh, gh^2, \dots, gh^\ell)$ for some $g \in G$ and $\ell \geq 0$. We also call the set $\mathsf{supp}(\pi)$ a progression, whose period may not be unique. If $h \neq 1$ we also call π a *ray*.

A *factorized walk* is a walk π equipped with a *factorization* $(\pi_1, \dots, \pi_n)$, i.e. $\pi = \pi_1 \dots \pi_n$. One also defines the concatenation of factorized walks in the straightforward fashion. If $E = e_1 \dots e_n$ is an exponent expression and ν is a valuation over E we define the factorized walk $\pi_{\nu,E} = \pi_1 \dots \pi_n$ induced by ν on E where

$$\pi_i = \begin{cases} (\nu(e_1 \dots e_{i-1})\, g_i^k)_{0 \leq k \leq \nu(x_i)}, & \text{if } e_i = g_i^{x_i} \\ (\nu(e_1 \dots e_{i-1}), \nu(e_1 \dots e_{i-1})\, g_i), & \text{if } e_i = g_i. \end{cases}$$

The *intersection knapsack problem* $\mathsf{KP}^\pm(G)$ over G is defined as follows:

Given a knapsack expression E over G, a set $L \subseteq [0, n]^2$ of loop constraints, and a set $D \subseteq [1, n]^2$ of disjointness constraints.

Question Is there a valuation ν such that $\nu(E) = 1$ and the factorized walk $\pi_{\nu,E} = \pi_1 \dots \pi_n$ induced by ν on E satisfies the following conditions:

- $\pi_{i+1} \dots \pi_j$ is a loop for every $(i, j) \in L$
- π_i and π_j are disjoint for every $(i, j) \in D$.

The *positive intersection knapsack problem* $\mathsf{KP}^+(G)$ over G is the restriction of $\mathsf{KP}^\pm(G)$ to instances where $D = \emptyset$. We denote the set of solutions of a $\mathsf{KP}^\pm(G)$-instance (resp. $\mathsf{KP}^+(G)$-instance) (E, I, D) (resp. (E, I)) as $\mathsf{sol}_G(E, I, D)$ (resp. $\mathsf{sol}_G(E, I)$). Figure 1 shows an example for the intersection knapsack problem over $\mathbb{Z}^2$.

The following is our main result.

▶ **Theorem 3.1.** *Let G and H be f.g. groups such that G is non-trivial and H is infinite. Then $\mathsf{KP}(G \wr H)$ is decidable if and only if $\mathsf{ExpEq}(G)$ is decidable and either*

1. *G is abelian and $\mathsf{KP}^+(H)$ is decidable or*
2. *G is not abelian and $\mathsf{KP}^\pm(H)$ is decidable.*

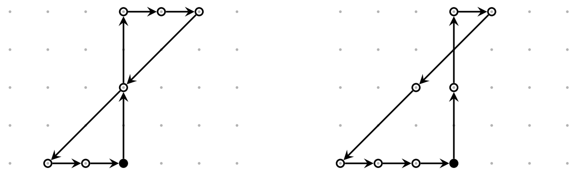


Figure 1 Consider the knapsack equation $g_1^{x_1} g_2^{x_2} g_3^{x_3} g_4^{x_4} = 1$ over $\mathbb{Z}^2$ written multiplicatively, where $g_1 = (0, 2)$, $g_2 = (1, 0)$, $g_3 = (-2, -2)$ and $g_4 = (1, 0)$ and the disjointness condition $D = \{(1, 3)\}$. The solid dot represents the origin $(0, 0)$. The knapsack equation is satisfied by $(x_1, x_2, x_3, x_4) = (2, 2, 2, 2)$ but it violates D, as illustrated on the left. On the right the solution $(x_1, x_2, x_3, x_4) = (2, 1, 2, 3)$ is depicted, which satisfies D.

Here, we assume H to be infinite, because the case of finite H is not interesting: If $|H| = m$, then $G \wr H$ has G^m as a finite-index subgroup [18, Proposition 1], meaning $\mathsf{KP}(G \wr H)$ is decidable if and only if $\mathsf{KP}(G^m)$ is [16, Theorem 7.3].

If H is knapsack-semilinear, it is easy to see that both $\mathsf{KP}^+(H)$ and $\mathsf{KP}^\pm(H)$ are decidable via an encoding in Presburger arithmetic. Hence, the main decidability result of [10], saying that for knapsack-semilinear H, $\mathsf{KP}(G \wr H)$ is decidable if and only if $\mathsf{ExpEq}(G)$ is decidable, is generalized by Theorem 3.1.

Logical version of KP$^+$ and KP$^\pm$. For our applications of Theorem 3.1, it is often convenient to use a formulation of $\mathsf{KP}^+(G)$ and $\mathsf{KP}^\pm(G)$ in terms of logics over an extended Cayley graph of G. The *Cayley graph of* G is the logical structure $\mathcal{C}(G) = (G, (\xrightarrow{g})_{g \in G})$, with domain G and with the relation $\xrightarrow{g}$ for each[1] $g \in G$, where $g_1 \xrightarrow{g} g_2$ if and only if $g_1 g = g_2$. We define the extension $\mathcal{C}^+(G) = (G, (\xrightarrow{g})_{g \in G}, (\xrightarrow{g}{}^*)_{g \in G})$ where $\xrightarrow{g}{}^*$ is the reflexive transitive closure of $\xrightarrow{g}$. Finally, we define a further extension $\mathcal{C}^\pm(G) = (G, (\xrightarrow{g})_{g \in G}, (\xrightarrow{g}{}^*)_{g \in G}, (\perp_{g,h})_{g,h \in G})$ with *disjointness relations* $\perp_{g,h}$, which are binary relations on pairs G^2: For any $g, h \in G$ and $(g_1, g_2), (h_1, h_2) \in G^2$ we have that $(g_1, g_2) \perp_{g,h} (h_1, h_2)$ if and only if for some $k, \ell \in \mathbb{N}$, we have $g_1 g^k = g_2$, $h_1 h^\ell = h_2$, and the walks $(g_1, g_1 g, \ldots, g_1 g^k)$ and $(h_1, h_1 h, \ldots, h_1 h^\ell)$ are disjoint. We denote by $\mathcal{F}^\pm$ the set of positive existential first-order formulas over $\mathcal{C}^\pm(G)$, i.e. formulas $\exists y_1 \ldots \exists y_m \varphi(y_1, \ldots, y_m)$ where $\varphi(y_1, \ldots, y_m)$ is a positive Boolean combination of atomic formulas. Then $\mathsf{SAT}^\pm(G)$ is the decision problem that asks if a closed formula in $\mathcal{F}^\pm$ holds in $\mathcal{C}^\pm(G)$. The fragment $\mathcal{F}^+$ and the problem $\mathsf{SAT}^+(G)$ are defined similarly. Clearly, $\mathsf{KP}^\pm(G)$ (resp. $\mathsf{KP}^+(G)$) reduces to $\mathsf{SAT}^\pm(G)$ (resp. $\mathsf{SAT}^+(G)$). In the full version [5], we show:

▶ **Theorem 3.2.** *For any finitely generated group G, the problem $\mathsf{SAT}^\pm(G)$ (resp. $\mathsf{SAT}^+(G)$) is decidable if and only if $\mathsf{KP}^\pm(G)$ (resp. $\mathsf{KP}^+(G)$) is decidable.*

Virtually nilpotent groups. It was shown by Ganardi, König, Lohrey, and Zetzsche that for some number $\ell \in \mathbb{N}$ and all groups $G \neq 1$, $\mathsf{KP}(G \wr (H_3(\mathbb{Z}) \times \mathbb{Z}^\ell))$ is undecidable [10, Theorem 5.2], but essentially nothing is known so far about the groups G for which the problem $\mathsf{KP}(G \wr H_3(\mathbb{Z}))$ is decidable. Using Theorem 3.1, this can be settled.

▶ **Theorem 3.3.** *For every non-trivial G, the problem $\mathsf{KP}(G \wr H_3(\mathbb{Z}))$ is undecidable.*

[1] Customarily, one only includes the edge relations $(\xrightarrow{s})_{s \in S}$ for some finite generating set S of G. We choose $S = G$ to make the presentation in the following cleaner.

This is in contrast to decidability of $\mathsf{KP}(H_3(\mathbb{Z}))$ [16, Theorem 6.8]. We show Theorem 3.3 by proving in Section 6 that $\mathsf{SAT}^+(H_3(\mathbb{Z}))$ (and thus $\mathsf{KP}^+(H_3(\mathbb{Z}))$) is undecidable.

The interest in the Heisenberg group stems from its special role inside the class of virtually nilpotent groups. This class, in turn, consists exactly of the finite extensions of groups of unitriangular integer matrices (see, for example, [14, Theorem 17.2.5]). Furthermore, a celebrated result of Gromov [12] states that the f.g. virtually nilpotent groups are precisely the f.g. groups with polynomial growth. In some sense, the discrete Heisenberg group is the smallest f.g. virtually nilpotent group that is not virtually abelian. Therefore, Theorem 3.3 implies the following characterization of all wreath products $G \wr H$ with decidable $\mathsf{KP}(G \wr H)$ where H is infinite and virtually nilpotent. See the full version [5] for details.

▶ **Corollary 3.4.** *Let G, H be f.g. non-trivial groups. If H is virtually nilpotent and infinite, then $\mathsf{KP}(G \wr H)$ is decidable if and only if H is virtually abelian and $\mathsf{ExpEq}(G)$ is decidable.*

By undecidability of $\mathsf{ExpEq}(H_3(\mathbb{Z}))$, this implies: If $G \neq 1$ and H are f.g. virtually nilpotent and H is infinite, then $\mathsf{KP}(G \wr H)$ is decidable if and only if G and H are virtually abelian.

Solvable Baumslag-Solitar groups. Our second application of Theorem 3.1 concerns wreath products $G \wr \mathsf{BS}(1,q)$. It is known that knapsack is decidable for $\mathsf{BS}(1,q)$ [21, Theorem 4.1], but again, essentially nothing is known about $\mathsf{KP}(G \wr \mathsf{BS}(1,q))$ for any G.

▶ **Theorem 3.5.** *For any f.g. group G and $q \geq 1$, the problem $\mathsf{KP}(G \wr \mathsf{BS}(1,q))$ is decidable if and only if $\mathsf{ExpEq}(G)$ is decidable.*

Extending methods from Lohrey and Zetzsche [21], we show that $\mathsf{KP}^\pm(\mathsf{BS}(1,q))$ is decidable for any $q \geq 1$ and thus obtain Theorem 3.5 in Section 6.

Magnus embedding. Another corollary concerns groups of the form $F/[N,N]$, where F is a f.g. free group and N is a normal subgroup. Recall that any f.g. group can be written as F/N, where F is an f.g. free group and N is a normal subgroup of F. Dividing by $[N,N]$ instead of N yields $F/[N,N]$, which is subject to the Magnus embedding [22, Lemma] of $F/[N,N]$ into $\mathbb{Z}^r \wr (F/N)$, where r is the rank of F. We show in the full version [5]:

▶ **Corollary 3.6.** *Let F be a finitely generated free group and N be a normal subgroup of F. If $\mathsf{KP}^+(F/N)$ is decidable, then so is $\mathsf{KP}(F/[N,N])$.*

Knapsack vs. intersection knapsack. Introducing the problems KP^+ and $\mathsf{KP}^\pm$ raises the question of whether they are substantially different from the similar problems KP and ExpEq: Is $\mathsf{KP}^+(G)$ or $\mathsf{KP}^\pm(G)$ perhaps inter-reducible with $\mathsf{KP}(G)$ or $\mathsf{ExpEq}(G)$? Our applications show that this is not the case. Since $\mathsf{KP}(H_3(\mathbb{Z}))$ is decidable [16, Theorem 6.8], but $\mathsf{KP}^+(H_3(\mathbb{Z}))$ is not, neither $\mathsf{KP}^+(G)$ nor $\mathsf{KP}^\pm(G)$ can be inter-reducible with $\mathsf{KP}(G)$ in general. Moreover, one can show[2] that $\mathsf{ExpEq}(\mathsf{BS}(1,2))$ is undecidable [11], whereas $\mathsf{KP}^\pm(\mathsf{BS}(1,q))$ is decidable for any $q \geq 1$. Hence, neither $\mathsf{KP}^+(G)$ nor $\mathsf{KP}^\pm(G)$ can be inter-reducible with $\mathsf{ExpEq}(G)$ in general. However, we leave open whether there is a f.g. group G for which $\mathsf{KP}^+(G)$ is decidable, but $\mathsf{KP}^\pm(G)$ is undecidable (see Section 7).

[2] Since there is no published proof available, we include a proof in the full version [5], with kind permission of Moses Ganardi and Markus Lohrey.

4 From wreath products to intersection knapsack

In this section, we prove the "if" direction of Theorem 3.1 by deciding $\mathsf{KP}(G \wr H)$ using $\mathsf{ExpEq}(G)$ and either $\mathsf{KP}^{\pm}(H)$ or $\mathsf{KP}^{+}(H)$ (depending on whether G is abelian).

Normalization. We fix a wreath product $G \wr H$ with G and H finitely generated groups. Note that we may assume that $\mathsf{KP}(H)$ is decidable. In our reduction, we will augment the $\mathsf{KP}(G \wr H)$-instance with positive intersection constraints regarding the cursor in H. This results in instances of the *hybrid intersection knapsack problem* $\mathsf{HKP}^{\pm}(G \wr H)$ over $G \wr H$: It is defined as $\mathsf{KP}^{\pm}(G \wr H)$ but the loop and disjointness constraints consider the σ-image of elements. Let us make this more precise. If $E = \alpha_1 \cdots \alpha_n$ is a knapsack expression over $G \wr H$, then we define for all $i \in [1, n]$ and $\nu \in \mathbb{N}^X$ the set

$$\mathsf{supp}_E^{\nu}(i) := \{\sigma(\nu(\alpha_1 \cdots \alpha_{i-1})\gamma(\alpha_i)^k) \mid 0 \le k \le \nu(x_i) - 1\}$$

if $i \in P_E$ and

$$\mathsf{supp}_E^{\nu}(i) := \{\sigma(\nu(\alpha_1 \cdots \alpha_{i-1}))\}$$

if $i \in Q_E$. For a walk $w = (w_1, \ldots, w_k)$ over $G \wr H$ we write $\sigma(w) := (\sigma(w_1), \ldots, \sigma(w_k))$. Then the *hybrid intersection knapsack problem* $\mathsf{HKP}^{\pm}(G \wr H)$ over $G \wr H$ is defined as follows:

Given a knapsack expression E over G, a set $L \subseteq [0, n]^2$ of loop constraints, and a set $D \subseteq [1, n]^2$ of disjointness constraints.

Question Is there a valuation $\nu \in \mathbb{N}^X$ with factorized walk $\pi_{\nu,E} = \pi_1 \ldots \pi_n$ induced by ν on E such that the following conditions are fulfilled:

- $\nu(E) = 1$
- $\sigma(\pi_{i+1} \ldots \pi_j)$ is a loop for all $(i, j) \in L$
- $\mathsf{supp}_E^{\nu}(i) \cap \mathsf{supp}_E^{\nu}(j) = \emptyset$ for all $(i, j) \in D$.

Its *positive* version $\mathsf{HKP}^{+}(G \wr H)$ is again defined by having no disjointness constraints. The set $\mathsf{sol}_{G \wr H}$ is defined accordingly. Note that to simplify the constructions in the proofs, the disjointness constraints in an $\mathsf{HKP}^{\pm}(G \wr H)$-instance disregard the last point of walks.

In the following, when we write a knapsack expression as $E = \alpha_1 \cdots \alpha_n \alpha_{n+1}$, we assume w.l.o.g. that α_{n+1} is a constant. Two elements $g, h \in H$ are called *commensurable* if $g^x = h^y$ for some $x, y \in \mathbb{Z} \setminus \{0\}$. It is known that if g_1, g_2 have infinite order and are not commensurable, then there is at most one solution $(x_1, x_2) \in \mathbb{Z}^2$ for the equations $g_1^{x_1} g_2^{x_2} = g$ [7, Lemma 9].

Let $E = \alpha_1 \cdots \alpha_n \alpha_{n+1}$ be a knapsack expression and write $g_i = \gamma(\alpha_i)$ for $i \in [1, n+1]$. The expression (resp. the corresponding $\mathsf{HKP}^{\pm}(G \wr H)$-instance) is *c-simplified* if for any $i, j \in P_E$ with $g_i \notin H$ and $g_j \notin H$, we have that commensurability of $\sigma(g_i)$ and $\sigma(g_j)$ implies $\sigma(g_i) = \sigma(g_j)$. We call the expression (resp. the corresponding $\mathsf{HKP}^{\pm}(G \wr H)$-instance) *normalized* if it is c-simplified and each atom α_i with $i \in [1, n]$ is of one of the following types: We either have (a) $i \in Q_E$ and $g_i \in H$ or (b) $i \in P_E$ and $\sigma(g_i) = 1$ or (c) $i \in P_E$, $g_i \in GH$ and $\sigma(g_i)$ has infinite order. Using generalizations of ideas from [10] and [7], we show:

▶ **Theorem 4.1.** *Given an instance of* $\mathsf{KP}(G \wr H)$*, one can effectively construct an equivalent finite set of normalized* $\mathsf{HKP}^{+}(G \wr H)$*-instances.*

Here, a problem instance I is *equivalent* to a set $\mathcal{I}$ of problem instances if I has a solution if and only if at least one of the instances in $\mathcal{I}$ has a solution.

Non-abelian case. Note that in a normalized knapsack expression, atoms of type (b) and (c) and the last atom α_{n+1} may place non-trivial elements of G. Our next step is to transform the input instance further so that only the atoms of type (c) can place non-trivial elements of G, which leads to the notion of stacking-freeness.

Let $E = \alpha_1 \cdots \alpha_n \alpha_{n+1}$ be a knapsack expression over $G \wr H$ and let $g_i := \gamma(\alpha_i)$ for all $i \in [1, n+1]$. We call an index $i \in [1, n+1]$ *stacking* if either $i \in P_E$ and $\sigma(g_i) = 1$, or $i = n+1$ and $g_{n+1} \notin H$. We say that E is *stacking-free* if it has no stacking indices. Thus, a normalized expression E is stacking-free if each atom is either of type (c) or a constant in H.

▶ **Lemma 4.2.** *Given a normalized* $\mathsf{HKP}^{\pm}(G \wr H)$*-instance, one can effectively construct an equivalent finite set of stacking-free, normalized* $\mathsf{HKP}^{\pm}(G \wr H)$*-instances.*

Let us sketch the proof of Lemma 4.2. We use the notion of an address from [10]. An *address* of E is a pair (i, h) with $i \in [1, n+1]$ and $h \in H$ such that $h \in \mathsf{supp}(\gamma(\alpha_i))$. The set of addresses A_E of E is finite and can be computed. Intuitively, an address represents a position in a knapsack expression where a point in H can be visited.

Intuitively, instead of placing elements of G by atoms of type (b) and by α_{n+1}, we introduce loop and disjointness constraints guaranteeing that in points visited by these atoms, a solution would have placed elements that multiply to $1 \in G$. To this end, we pick an address $(i, h) \in A$ of a stacking index i and then guess a set $C \subseteq A$ of addresses such that the point $h' \in H$ visited at (i, h) is visited by exactly the addresses in C. The latter condition is formulated using loop and disjointness constraints in an $\mathsf{HKP}^{\pm}(G \wr H)$-instance I_C. In I_C, we do not place elements at C anymore; instead, we construct a set S_C of exponent equations over G that express that indeed the point h' carries $1 \in G$ in the end. Note that this eliminates one address with stacking index. We repeat this until we are left with a set of stacking-free instances of $\mathsf{HKP}^{\pm}(G \wr H)$, each together with an accumulated set of exponent equations over G. We then take the subset $\mathcal{I}$ of $\mathsf{HKP}^{\pm}(G \wr H)$-instances whose associated $\mathsf{ExpEq}(G)$-instance has a solution. This will be our set for Lemma 4.2.

The last step of the non-abelian case is to construct $\mathsf{KP}^{\pm}(H)$-instances.

▶ **Lemma 4.3.** *Given a stacking-free, normalized* $\mathsf{HKP}^{\pm}(G \wr H)$*-instance, one can effectively construct an equivalent finite set of* $\mathsf{KP}^{\pm}(H)$*-instances.*

We are given an instance (E, L, D) with $E = \alpha_1 \cdots \alpha_n$ and write $g_i = \gamma(\alpha_i)$ for $i \in [1, n]$. As (E, L, D) is normalized and stacking-free, only atoms of type (c) with $g_i \notin H$ can place non-trivial elements of G. Moreover, if α_i and α_j are such atoms, then the elements $\sigma(g_i)$ and $\sigma(g_j)$ are either non-commensurable or equal. In the first case, the two rays produced by α_i and α_j can intersect in at most one point; in the second case, they intersect along subrays corresponding to intervals $I_i \subseteq [0, \nu(x_i)]$ and $I_j \subseteq [0, \nu(x_j)]$.

Thus, the idea is to split up each ray wherever the intersection with another ray starts or ends: We guess for each ray as above the number $m \leq 2 \cdot |A_E| - 1$ of subrays it will be split into and replace $g_i^{x_i}$ with $g_i^{y_1} \cdots g_i^{y_m}$. After the splitting, subrays are either equal or disjoint. We guess an equivalence relation on the subrays; using loop constraints, we ensure that subrays in the same class are equal; using disjointness constraints, we ensure disjointness of subrays in distinct classes. Finally, we have to check that for each equivalence class C, the element of G produced by the rays in C does indeed multiply to $1 \in G$. This can be checked because $\mathsf{ExpEq}(G)$ (and thus the word problem for G) is decidable.

Abelian case. We now come to the case of abelian G: We show that $\mathsf{KP}(G \wr H)$ is decidable, but only using instances of $\mathsf{KP}^{+}(H)$ instead of $\mathsf{KP}^{\pm}(H)$. Here, the key insight is that we can use the same reduction, except that we just do not impose the disjointness constraints. In the above reduction, we use disjointness constraints to control exactly which positions in our walk visit the same point in H. Then we can check that in the end, each point in H carries $1 \in G$. However, if G is abelian, it suffices to make sure that the set of positions in our walk decomposes into subsets, each of which produces $1 \in G$: If several of these subsets do visit the same point in H, the end result will still be $1 \in G$.

We illustrate this in a slightly simpler setting. Suppose we have a product $g = {}^{h_1}a_1 \cdots {}^{h_n}a_n$ with $h_1, \ldots, h_n \in H$ and $a_1, \ldots, a_n \in G$. Then g is obtained by placing a_1 at $h_1 \in H$, then a_2 at $h_2 \in H$, etc. For a subset $S = \{s_1, \ldots, s_k\} \subseteq [1, n]$ with $s_1 < \cdots < s_k$, we define $g_S = {}^{h_{s_1}}a_{s_1} \cdots {}^{h_{s_k}}a_{s_k}$. Hence, we only multiply those factors from S. An equivalence relation $\equiv$ on $[1, n]$ is called *cancelling* if $g_C = 1$ for every class C of $\equiv$. Moreover, $\equiv$ is called *equilocal* if $i \equiv j$ if and only if $h_i = h_j$. It is called *weakly equilocal* if $i \equiv j$ implies $h_i = h_j$. Now observe that for any G, we have $g = 1$ if and only if there is an equilocal cancelling equivalence on $[1, n]$. However, if G is abelian, then $g = 1$ if and only if there is a *weakly* equilocal equivalence on $[1, n]$. Since weak equilocality can be expressed using only equalities (and no disequalities), it suffices to impose loop conditions in our instances.

Comparison to previous approach in [7]. The reduction from $\mathsf{KP}(G \wr H)$ to $\mathsf{ExpEq}(G)$ and $\mathsf{KP}^{\pm}(H)$ ($\mathsf{KP}^{+}(H)$ respectively) uses similar ideas as the proof of [7, Theorem 4], where it is shown $\mathsf{ExpEq}(K)$ is in NP if K is an iterated wreath product of $\mathbb{Z}^r$ for some $r \in \mathbb{N}$.

Let us compare our reduction with the proof of [7, Theorem 4]. In [7], one solves $\mathsf{ExpEq}(K)$ by writing $K = G \wr H$ where G is abelian and H is orderable and knapsack-semilinear. In both proofs, solvability of an instance (of $\mathsf{ExpEq}(G \wr H)$ in [7] and $\mathsf{KP}(G \wr H)$ here) is translated into a set of conditions by using similar decomposition arguments. Then, the two proofs differ in how satisfiability of these conditions is checked.

In [7], this set of conditions is expressed in Presburger arithmetic, which is possible due to knapsack-semilinearity of H. In our reduction, we have to translate the conditions in $\mathsf{ExpEq}(G)$ and $\mathsf{KP}^{+}(H)$ ($\mathsf{KP}^{\pm}(H)$) instances. Here, we use loop constraints where in Presburger arithmetic, once can compare variables directly. Moreover, our reduction uses disjointness constraints to express solvability in the case that G is non-abelian. This case does not occur in [7, Theorem 4]. Finally, we have to check whether the elements from G written at the same point of H multiply to 1. The reduction of [7] can express this directly in Presburger arithmetic since G is abelian. Here, we use instances of $\mathsf{ExpEq}(G)$.

5 From intersection knapsack to wreath products

In this section, we prove the "only if" direction of Theorem 3.1. Since it is known that for infinite H, decidability of $\mathsf{KP}(G \wr H)$ implies decidability of $\mathsf{ExpEq}(G)$ [10, Proposition. 3.1, Proposition 5.1], it remains to reduce (i) $\mathsf{KP}^{+}(H)$ to $\mathsf{KP}(G \wr H)$ for any group $G \neq 1$, and (ii) $\mathsf{KP}^{\pm}(H)$ to $\mathsf{KP}(G \wr H)$ for any non-abelian group G. In the following, let G be a non-trivial group and H be any group and suppose $\mathsf{KP}(G \wr H)$ is decidable.

First let us illustrate how to reduce $\mathsf{KP}^{+}(H)$ to $\mathsf{KP}(G \wr H)$. Suppose we want to verify whether a product $h_1 \ldots h_m = 1$ over H satisfies a set of loop constraints $L \subseteq [0, m]^2$, i.e. $h_{i+1} \ldots h_j = 1$ for all $(i, j) \in L$. To do so we insert into the product for each $(i, j) \in L$ a function $f \in G^{(H)}$ after the element h_i and its inverse f^{-1} after the element h_j. We call these functions *loop words* since their supports are contained in a cyclic subgroup $\langle t \rangle$ of H. We can choose the loop words such that this modified product evaluates to 1 if and only if the loop constraints are satisfied. For the reduction from $\mathsf{KP}^{\pm}(H)$ we need to make the construction more robust since we simultaneously need to simulate disjointness constraints.

If H is a torsion group then $\mathsf{KP}^+(H)$ and $\mathsf{KP}^\pm(H)$ are decidable if the word problem of H is decidable: For each exponent, we only have to check finitely many candidates. Since $\mathsf{KP}(G \wr H)$ is decidable, we know that $\mathsf{KP}(H)$ is decidable and hence also the word problem. Thus, we assume H not to be a torsion group and may fix an element $t \in H$ of infinite order.

Periodic complexity. Let K be a group. The following definitions will be employed with $K = \mathbb{Z}$ or $K = H$. For any subset $D \subseteq K$, let $G^{(D)}$ be the group of all functions $u \colon K \to G$ whose support $\mathsf{supp}(u) = \{h \in K \mid u(h) \neq 1\}$ is finite and contained in D. A function $f \in G^{(K)}$ is *basic periodic* if there exists a progression D in K and $c \in G$ such that $f(h) = c$ for all $h \in D$ and $f(h) = 1$ otherwise. The *value* of such a function f is the element c; a *period* of f is a period of its support. We will identify a word $u = c_1 \dots c_n \in G^*$ with the function $u \in G^{(\mathbb{Z})}$ where $u(i) = c_i$ for $i \in [1, n]$ and $u(i) = 1$ otherwise. Recall that for $u \in G^{(\mathbb{Z})}$ and $s \in \mathbb{Z}$, we have ${}^s u(n) = u(n - s)$. We extend this to $s \in \mathbb{Z}_\infty := \mathbb{Z} \cup \{\infty\}$ by setting ${}^\infty u(n) = 1$ for all $n \in \mathbb{Z}$. The *periodic complexity* of $u \in G^{(\mathbb{Z})}$ is the minimal number $\mathsf{pc}(u) = k$ of basic periodic functions $u_1, \dots, u_k$ such that $u = \prod_{i=1}^k u_i$. Given a progression $D = \{p + qn \mid n \in [0, \ell]\}$ in $\mathbb{Z}$ and a function $u \in G^{(\mathbb{Z})}$ we define $\pi_D(u)(n) = u(p + qn)$ for all $n \in \mathbb{Z}$ and say that $\pi_D(u)$ is a *periodic subsequence* of u. Note that periodic subsequences of basic periodic functions are again basic periodic. Furthermore, since $\pi_D \colon G^{(\mathbb{Z})} \to G^{(\mathbb{Z})}$ is a homomorphism, taking periodic subsequences does not increase the periodic complexity.

▶ **Lemma 5.1.** *Given $n, k \in \mathbb{N}$ and $a \in G \setminus \{1\}$, one can compute $u_1, \dots, u_n \in \langle a \rangle^{(\mathbb{N})}$ such that $\prod_{i=1}^n {}^{p_i}u_i {}^{q_i}u_i^{-1}$ has periodic complexity $\geq k$ for all $(p_1, \dots, p_n) \neq (q_1, \dots, q_n) \in \mathbb{Z}_\infty^n$.*

Here is a proof sketch for Lemma 5.1. First we construct a word with large periodic complexity: In the full version [5] we prove that $(a)^{2^{2^k}} (1)^{2^{2^k}} \dots (a)^{2^{2^k}} (1)^{2^{2^k}}$, consisting of $4k$ many blocks, has periodic complexity at least k, where $(b)^n$ is the sequence consisting of n many b's. The case $n = 1$ can be shown by taking such a sequence $v = a_1 \dots a_m \in \langle a \rangle^{(\mathbb{N})}$ with large periodic complexity and defining $u_1 = a_1 (1)^{m-1} a_2 (1)^{m-1} \dots a_m (1)^{m-1} a_1 \dots a_m$. If $p, q \in \mathbb{Z}_\infty$ are distinct then ${}^p u_1 {}^q u_1^{-1}$ always contains v or v^{-1} as a periodic subsequence and thus has large periodic complexity. For $n > 1$ we define u_i $(i > 1)$ to be stretched versions of u_1 such that the supports of any two functions ${}^p u_i$, ${}^q u_j$ where $i \neq j$ intersect in at most one point. This allows to argue that $\prod_{i=1}^n {}^{p_i}u_i {}^{q_i}u_i^{-1}$ still has large periodic complexity as soon as $p_i \neq q_i$ for some i.

Expressing loop constraints. We now show how to use Lemma 5.1 to encode loop constraints over a product $h_1 \dots h_m$ over H in an instance of $\mathsf{KP}(G \wr H)$.

Recall that a loop constraint (i, j) stipulates that $\sigma(g_{i+1} \dots g_j) = 1$. If we only want to reduce $\mathsf{KP}^+(H)$, it is not hard to see that it would suffice to guarantee $\prod_{i=1}^n {}^{p_i}u_i {}^{q_i}u_i^{-1} \neq 1$ in Lemma 5.1. In that case, we could essentially use the functions u_i as loop words. However, in order to express disjointness constraints in $\mathsf{KP}^\pm(H)$, we will construct expressions over $G \wr H$ that place additional “disjointness patterns” in the Cayley graph of H. We shall make sure that the disjointness patterns are tame: Roughly speaking, this means they are basic periodic and either (i) place elements from a fixed subgroup $\langle a \rangle$ or (ii) can intersect a loop word at most once. Here, the high periodic complexity of $\prod_{i=1}^n {}^{p_i}u_i {}^{q_i}u_i^{-1}$ will allow us to conclude that tame patterns cannot make up for a violated loop constraint.

Let us make this precise. Recall that two elements $g, h \in H$ are called *commensurable* if $g^x = h^y$ for some $x, y \in \mathbb{Z} \setminus \{0\}$. Let $a \in G \setminus \{1\}$. Let $\mathsf{P}_{a,t}(G \wr H)$ be the set of elements $g \in G \wr H$ such that $\tau(g)$ is basic periodic and either, (i) its value belongs to $\langle a \rangle$, or (ii) its period is not commensurable to t. In particular, a power $(ch)^k$ (where $c \in G$, $h \in H$, $k \in \mathbb{N}$)

belongs to $\mathsf{P}_{a,t}(G \wr H)$ if $c \in \langle a \rangle$ or h is not commensurable to t. Note that since loop words are always placed along the direction t, this guarantees tameness: In case (ii), the period of $\tau(g)$ being non-commensurable to t implies that the support of any $h'g$, $h' \in H$, can intersect the support of a loop word in $\langle a \rangle^{(\langle t \rangle)}$ at most once. Using Lemma 5.1, we show the following.

▶ **Lemma 5.2.** *Given $a \in G \backslash \{1\}$, $m \in \mathbb{N}$ and $L \subseteq [0,m]^2$ we can compute $f_0, \ldots, f_m \in \langle a \rangle^{(t^*)}$ such that:*

1. *Let $h_1, \ldots, h_m \in H$. Then $h_1 \ldots h_m = 1$ and $h_{i+1} \ldots h_j = 1$ for all $(i,j) \in L$ if and only if $f_0 h_1 f_1 \ldots h_m f_m = 1$.*
2. *Let $g_1, \ldots, g_m \in \mathsf{P}_{a,t}(G \wr H)$ such that $\sigma(g_{i+1} \ldots g_j) \neq 1$ for some $(i,j) \in L$. Then $f_0 g_1 f_1 \ldots g_m f_m \neq 1$.*

Observe that the first constraint says that if we only use the loop words f_i, then they allow us to express loop constraints. The second constraint tells us that a violated loop constraint cannot be compensated even with perturbations $g_1, \ldots, g_m$, provided that they are tame.

The abelian case. Lemma 5.2 provides a simple reduction from $\mathsf{KP}^+(H)$ to $\mathsf{KP}(G \wr H)$. Given an instance $(E = e_1 \ldots e_n, L)$ of $\mathsf{KP}^+(H)$ we compute $f_0, \ldots, f_m \in \langle a \rangle^{(t^*)}$ using Lemma 5.2. Then $\nu \colon X \to \mathbb{N}$ satisfies $\nu(E) = 1$ and $\nu(e_{i+1} \ldots e_j)$ for all $(i,j) \in L$ if and only if $\nu(f_0 e_1 f_1 \ldots e_n f_n) = 1$. Hence (E, L) has a solution if and only if $\nu(f_0 e_1 f_1 \ldots e_n f_n) = 1$ does.

The non-abelian case. Now let G be a non-abelian group. In the following we will reduce $\mathsf{KP}^\pm(H)$ to $\mathsf{KP}(G \wr H)$. The first step is to construct from an $\mathsf{KP}^\pm(H)$-instance I an equivalent $\mathsf{HKP}^+(G \wr H)$-instance $\hat{I}$ using a nontrivial commutator $[a,b] \neq 1$ in G. In a second step we apply the "loop words"-construction from Lemma 5.2 (point 2) to $\hat{I}$, going to a (pure) knapsack instance. It guarantees that, if a loop constraint is violated, then the knapsack instance does not evaluate to 1. Furthermore, if a disjointness constraint is violated then there exists a large number of pairwise distant points in the Cayley graph of H which are labeled by a nontrivial element. These points cannot be canceled by the functions f_i from Lemma 5.2. Finally, if all loop and disjointness constraints are satisfied then the induced walk in the Cayley graph provides enough "empty space" such that the loop words can be shifted to be disjoint from the original walk induced by $\hat{I}$ (encoding the disjointness constraints).

Normalization. Let $I = (E = e_1 \ldots e_n, L, D)$ be a $\mathsf{KP}^\pm(H)$-instance where e_i is either a constant $e_i = h_i$ or a power $e_i = h_i^{x_i}$. We will start by establishing the following useful properties. We call I *torsion-free* if h_i has infinite order for all $i \in P_E$. Call I *orthogonalized* for all $(i,j) \in D \cap P_E^2$ such that we have $\langle h_i \rangle \cap \langle h_j \rangle = \{1\}$. If I is torsion-free and orthogonalized then it is called *normalized.* The orthogonality will be crucial for the tameness of the disjointness patterns since at most one of the elements h_i, h_j for $(i,j) \in D \cap P_E^2$ is commensurable to t. Furthermore, it guarantees that there is at most one intersection point for any pair $(i,j) \in D$.

▶ **Lemma 5.3.** *One can compute a finite set $\mathcal{I}$ of normalized instances of $\mathsf{KP}^\pm(H)$ such that I has a solution if and only if there exists $I' \in \mathcal{I}$ which has a solution.*

Here, torsion-freeness is easily achieved: If h_i has finite order, then $h_i^{x_i}$ can only assume finitely many values, so we replace $h_i^{x_i}$ by one of finitely many constants. Orthogonality requires an observation: If $\langle h_i \rangle \cap \langle h_j \rangle \neq \{1\}$, then any two intersecting progressions π_i, π_j

with periods h_i and h_j, respectively, must intersect periodically, meaning there exists an intersection point that is close to an endpoint of π_i or π_j. This means, in lieu of $(i,j) \in D$, we can require disjointness of one power with a constant.

Expressing disjointness constraints. Hence we can assume that I is normalized. To express disjointness constraints, we must assume that G is non-abelian. Let $a, b \in G$ with $aba^{-1}b^{-1} = [a,b] \neq 1$. Our starting point is the following idea. To express that two progressions π_i and π_j, induced by a valuation of E, are disjoint, we construct an expression over $G \wr H$ that first places a at each point in π_i, then b at each point in π_j, then again a^{-1} at each point in π_i, and finally b^{-1} at each point in π_j, see (2). Here we need loop constraints that express that the start and endpoints of the two traversals of π_i (and π_j) coincide. Then, if π_i and π_j are disjoint, the effect will be neutral; otherwise any intersection point will carry $aba^{-1}b^{-1} \neq 1$.

However, this leads to two problems. First, there might be more than one disjointness constraint: If k disjointness constraints are violated by the same point $h'' \in H$, then h'' would carry $[a,b]^k$, which can be the identity (for example, G may be finite). Second, when we also place loop words (which multiply elements from $\langle a \rangle$), those could also interfere with the commutator (for example, instead of $aba^{-1}b^{-1}$, we might get $aba^{-1}(a)b^{-1}(a^{-1}) = 1$).

Instead, we do the following. Let $t \in H$ be the element of infinite order used for the loop words. Moreover, let $D = \{(i_1, j_1), \ldots, (i_d, j_d)\}$. For each $(i_k, j_k) \in D$, instead of performing the above "commutator construction" once, we perform it $n + d$ times, each time shifted by $t^{N_k} \in H$ for some large N_k. The numbers $N_0 < N_1 < \cdots$ are chosen so large that for at least one commutator, there will be no interference from other commutators or from loop words.

Let us make this precise. Since I is orthogonalized, we may assume that for each $(i,j) \in D \cap P_E^2$, the elements h_j and t are not commensurable; otherwise we swap i and j. The resulting $\mathsf{HKP}^+(G \wr H)$-instance $\hat{I}$ will have length $m = n + 4d(n+d)(n+2)$. In preparation, we can compute a number N such that the functions $f_0, \ldots, f_m$ from Lemma 5.2 for any $L \subseteq [0,m]^2$ satisfy $\mathsf{supp}(f_i) \subseteq \{t^j \mid j \in [0, N-1]\}$. For each $i \in [1,n]$, $c \in G$, $s \in \mathbb{N}$, we define the knapsack expression $E_{i,c,s}$ over $G \wr H$ as

$$E_{i,c,s} = \begin{cases} e_1 \ldots e_{i-1}\,(t^s)\,(c\,t^{-s} h_i t^s)^{x_i}\,(ct^{-s})\,e_{i+1} \ldots e_n, & \text{if } e_i = h_i^{x_i}, \\ e_1 \ldots e_{i-1}\,(t^s)\,(c\,t^{-s} h_i t^s)\;\;(ct^{-s})\,e_{i+1} \ldots e_n, & \text{if } e_i = h_i. \end{cases} \tag{1}$$

The parentheses indicate the atoms. We define

$$\hat{E} = E \cdot \prod_{k=1}^{d} \prod_{s \in S_k} \Big(E_{i_k,a,s} \cdot E_{j_k,b,s} \cdot E_{i_k,a^{-1},s} \cdot E_{j_k,b^{-1},s} \Big) \tag{2}$$

where $S_k = \{j(n+d)^{2k}N \mid j \in [1, n+d]\}$ for all $k \in [1,d]$, and all occurrences of expressions of the form $E_{i,c,s}$ use fresh variables. Note that $E_{i_k,a,s} \cdot E_{j_k,b,s} \cdot E_{i_k,a^{-1},s} \cdot E_{j_k,b^{-1},s}$ performs the commutator construction for (i_k, j_k), shifted by t^s. Let $\hat{E} = \hat{e}_1 \ldots \hat{e}_m$ be the resulting expression. Notice that its length is indeed $m = n + 4d(n+d)(n+2)$ as claimed above.

Finally, in our $\mathsf{HKP}^+(G \wr H)$ instance, we also add a set $J \subseteq [0,m]^2$ of loop constraints stating that for each $k \in [1,d]$ and $s \in S_k$, the i_k-th atom in $E_{i_k,a,s}$ arrives at the same place in H as the i_k-th atom in E (and analogously for $E_{j_k,b,s}$, $E_{i_k,a^{-1},s}$, $E_{j_k,b^{-1},s}$). See [5] for details.

Let $f_0, \ldots, f_m \in \langle a \rangle^{(t^*)}$ be the loop words from Lemma 5.2 for the set $J \subseteq [0,m]^2$. It is now straightforward to verify that the elements $\hat{e}_i$ are all tame as explained above. In other words, for every valuation ν and $i \in [1,m]$, we have $\nu(\hat{e}_i) \in \mathsf{P}_{a,t}$ (see [5] for a proof).

Shifting loop words. By construction, we now know that if the instance $f_0\hat{e}_1f_1\cdots\hat{e}_mf_m$ of $\mathsf{KP}(G \wr H)$ has a solution, then so does our normalized instance I of $\mathsf{KP}^{\pm}(H)$. However, there is one last obstacle: Even if all loop and disjointness constraints can be met for I, we cannot guarantee that $f_0\hat{e}_1f_1\cdots\hat{e}_mf_m$ has a solution: It is possible that some loop words interfere with some commutator constructions so as to yield an element $\neq 1$.

The idea is to *shift* all the loop words $f_0,\ldots,f_m$ in direction t by replacing f_i by $t^rf_it^{-r} = {}^{t^r}\!f_i$ for some $r\in\mathbb{N}$. We shall argue that for some r in some bounded interval, this must result in an interference free expression; even though the elements $\hat{e}_i$ may modify an unbounded number of points in H. To this end, we use again that the $\hat{e}_i$ are tame: Each of them either (i) places elements from $\langle a\rangle$, or (ii) has a period non-commensurable to t. In the case (i), there can be no interference because the f_i also place elements in $\langle a\rangle$, which is an abelian subgroup. In the case (ii), $\hat{e}_i$ can intersect the support of each f_j at most once. Hence, there are at most m points each f_j has to avoid after shifting. The following simple lemma states that one can always shift finite sets F_i in parallel to avoid finite sets A_i, by a bounded shift. Notice that the bound does not depend on the size of the elements in the sets F_i and A_i.

▶ **Lemma 5.4.** *Let $F_1,\ldots,F_m\subseteq\mathbb{Z}$ with $|F_i|\le N$ and $A_1,\ldots,A_m\subseteq\mathbb{Z}$ with $|A_i|\le\ell$. There exists a shift $r\in[0,Nm\ell]$ such that $(r+F_i)\cap A_i=\emptyset$ for each $i\in[1,m]$.*

Proof. For every $a\in\mathbb{Z}$ there exist at most $|F_i|\le N$ many shifts $r\in\mathbb{N}$ where $a\in r+F_i$. Therefore there must be a shift $r\in[0,Nm\ell]$ such that $(r+F_i)\cap A_i=\emptyset$ for each $i\in[1,m]$. ◀

We can thus prove the following lemma, which clearly completes the reduction from $\mathsf{KP}^{\pm}(H)$ to $\mathsf{KP}(G\wr H)$.

▶ **Lemma 5.5.** *$I=(E,L,D)$ has a solution if and only if ${}^{t^r}\!f_0\hat{e}_1{}^{t^r}\!f_1\ldots\hat{e}_m{}^{t^r}\!f_m$ has a solution for some $r\in[0,Nm^2]$.*

6 Applications

The discrete Heisenberg group. Here, we prove that $\mathsf{SAT}^+(H_3(\mathbb{Z}))$ is undecidable. Together with Theorem 3.1 and Theorem 3.2, this directly implies Theorem 3.3. Define the matrices $A=\left(\begin{smallmatrix}1&1&0\\0&1&0\\0&0&1\end{smallmatrix}\right)$, $B=\left(\begin{smallmatrix}1&0&0\\0&1&1\\0&0&1\end{smallmatrix}\right)$, and $C=\left(\begin{smallmatrix}1&0&1\\0&1&0\\0&0&1\end{smallmatrix}\right)$. The group $H_3(\mathbb{Z})$ is generated by A and B and we have $AC=CA$ and $BC=CB$. It is well-known that (I) $A^iC^j=A^{i'}C^{j'}$ iff $i=i'$ and $j=j'$; and (II) $B^iC^j=B^{i'}C^{j'}$ iff $i=i'$ and $j=j'$; and (III) $A^iB^jA^{-i'}B^{-j'}=C^k$ if and only if $i=i'$, $j=j'$, and $k=ij$. For proofs, see the full version [5].

We show undecidability of $\mathsf{SAT}^+(H_3(\mathbb{Z}))$ by reducing from solvability of Diophantine equations over natural numbers. Hence, we are given a finite system $\bigwedge_{j=1}^m E_j$ of equations of the form $x=a$, $z=x+y$, and $z=xy$. It is well-known that solvability of such equation systems is undecidable [23]. Given such an equation system over a set of variables X we define a $\mathcal{C}^+(H_3(\mathbb{Z}))$-formula containing the variables $\{g_x\mid x\in X\}\cup\{g_0\}$ with the interpretation that $g_x=g_0C^x$. First we state that $g_0\xrightarrow{C}{}^*g_x$ for all $x\in X$. Expressing $x=a$ is done simply with $g_0\xrightarrow{C^a}g_x$. For $z=x+y$, we use

$$C^xA^*\cap A^{x'}C^*\cap(AC)^*\neq\emptyset\quad\wedge\quad A^{x'}C^*\cap C^zA^*\cap C^y(AC)^*\neq\emptyset.$$

This can be expressed in $\mathcal{C}^+(H_3(\mathbb{Z}))$ with a fresh variable $f_{x'}$ for $g_0A^{x'}$: For example, the first conjunct holds iff there exists $h\in H_3(\mathbb{Z})$ such that $g_0\xrightarrow{A}{}^*f_{x'}$, $g_x\xrightarrow{A}{}^*h$, $f_{x'}\xrightarrow{C}{}^*h$, $g_0\xrightarrow{AC}{}^*h$. By (I) and $AC=CA$, the first conjunct holds iff $x=x'$. Similarly, the second conjunct holds iff $z=x'+y$, hence $z=x+y$. For $z=xy$, we use:

$$C^xA^*\cap A^{x'}C^*\cap(AC)^*\neq\emptyset\ \wedge\ B^{y'}C^*\cap C^yB^*\cap(BC)^*\neq\emptyset\ \wedge\ A^{x'}B^*(A^{-1})^*\cap B^{y'}C^*\cap C^zB^*\neq\emptyset.$$

Like above, the first and second conjunct express $x'=x$ and $y'=y$. The third says that $A^{x'}B^r(A^{-1})^s=B^{y'}C^z$ for some $r,s\geq 0$, so by (III), it states $z=x'y'$, hence $z=xy$.

Solvable Baumslag-Solitar groups. We show that $\mathsf{SAT}^{\pm}(\mathsf{BS}(1,q))$ is decidable for every $q\geq 1$. By Theorem 3.1 and Theorem 3.2, this proves Theorem 3.5. Our proof is based on the following observation, which is shown in the full version [5].

▶ **Proposition 6.1.** *The first-order theory of $\mathcal{C}^+(\mathsf{BS}(1,q))$ is decidable.*

For Proposition 6.1, we show that given any finite subset $F\subseteq\mathsf{BS}(1,q)$, the structure $(\mathsf{BS}(1,q),(\xrightarrow{g})_{g\in F},(\xrightarrow{g}{}^*)_{g\in F})$ is effectively an automatic structure, which implies that its first-order theory is decidable [15, Corollary 4.2]. This uses a straightforward extension of the methods in [21]. In [21, proof of Theorem 4.1], it is shown that $\mathsf{KP}(\mathsf{BS}(1,q))$ can be reduced to the existential fragment of the structure $(\mathbb{Z},+,V_q)$, where $V_q(n)$ is the largest power of q that divides n. The structure $(\mathbb{Z},+,V_q)$ is called *Büchi arithmetic* and is well-known to be automatic. Here, we show that $(\mathsf{BS}(1,q),(\xrightarrow{g})_{g\in F},(\xrightarrow{g}{}^*)_{g\in F})$ can be interpreted in a slight extension of Büchi arithmetic that is still automatic. From Proposition 6.1, we can derive a stronger statement, which clearly implies decidability of $\mathsf{SAT}^{\pm}(\mathsf{BS}(1,q))$:

▶ **Theorem 6.2.** *The first-order theory of $\mathcal{C}^{\pm}(\mathsf{BS}(1,q))$ is decidable.*

Indeed, since $\mathsf{BS}(1,q)$ is torsion-free, we can express the predicate $\perp_{g,h}$ using universal quantification: We have $(g_1,g_2)\perp_{g,h}(h_1,h_2)$ if and only if $g_1\xrightarrow{g}{}^*g_2$ and $h_1\xrightarrow{h}{}^*h_2$ and

$$\forall f,f'\in\mathsf{BS}(1,q)\colon\left(g_1\xrightarrow{g}{}^*f\wedge f\xrightarrow{g}{}^*g_2\wedge h_1\xrightarrow{h}{}^*f'\wedge f'\xrightarrow{h}{}^*h_2\right)\to f\neq f'.$$

7 Conclusion

We have shown that for non-trivial groups G and infinite groups H, the problem $\mathsf{KP}(G\wr H)$ is decidable if and only if $\mathsf{ExpEq}(G)$ is decidable and either (i) G is abelian and $\mathsf{KP}^+(H)$ is decidable or (ii) G is non-abelian and $\mathsf{KP}^{\pm}(H)$ is decidable. This reduces the study of decidablity of $\mathsf{KP}(G\wr H)$ to decidability questions about the factors G and H.

Intersection knapsack ($\mathsf{KP}^{\pm}$) vs positive intersection knapsack (KP^+). However, we leave open whether there is a group H where $\mathsf{KP}^+(H)$ is decidable, but $\mathsf{KP}^{\pm}(H)$ is undecidable. It is clear that both are decidable for all groups in the class of knapsack-semilinear groups. This class contains a large part of the groups for which knapsack has been studied. For example, it contains graph groups [20, Theorem 3.11] and hyperbolic groups [17, Theorem 8.1]. Moreover, knapsack-semilinearity is preserved by a variety of constructions: This includes wreath products [10, Theorem 5.4], graph products [8], free products with amalgamation and HNN-extensions over finite identified subgroups [8], and taking finite-index overgroups [8]. Moreover, the groups $H_3(\mathbb{Z})$ and $\mathsf{BS}(1,q)$ for $q\geq 2$ are also unable to distinguish KP^+ and $\mathsf{KP}^{\pm}$: We have shown here that KP^+ is undecidable in $H_3(\mathbb{Z})$ and $\mathsf{KP}^{\pm}$ is decidable in $\mathsf{BS}(1,q)$. To the best of the authors' knowledge, among the groups for which knapsack is currently known to be decidable, this only leaves $\mathsf{BS}(p,q)$ for p,q coprime, and $G\wr\mathsf{BS}(1,q)$ (with decidable $\mathsf{ExpEq}(G)$) as candidates to distinguish KP^+ and $\mathsf{KP}^{\pm}$.

Complexity. Another aspect that our work does not settle is the complexity of $\mathsf{KP}(G \wr H)$ for each G and H. We refer to [7] for a current overview on this.

The reductions presented here have high complexity. For example, our reduction from $\mathsf{KP}(G \wr H)$ involves several transformations of instances of $\mathsf{KP}(G \wr H)$, $\mathsf{HKP}(G \wr H)$, $\mathsf{KP}^{\pm}(H)$, $\mathsf{KP}(H)$, or $\mathsf{ExpEq}(G)$. In multiple of these transformations, as an auxiliary step, we take an instance I, extract from it a set of knapsack equations $a^x b^y = c$ with $a, b, c \in H$, find minimal solutions, and use them to compute a new instance I'. Thus, the complexity of our reduction depends on the size of minimal solutions to (two-variable) knapsack equations in H. Moreover, even if one assumes a polynomial bound on such solution sizes (which is known to hold, for example, in graph groups defined by transitive forests [20, Theorem 4.10] and in hyperbolic groups [27] (see also [17, Theorem 8.1])), our reduction still involves multiple steps that incur an exponential blow-up.

Furthermore, our reduction from $\mathsf{KP}^{\pm}(H)$ (or $\mathsf{KP}^{+}(H)$) to $\mathsf{KP}(G \wr H)$ produces double-exponentially many instances of $\mathsf{KP}(G \wr H)$, each of which is doubly exponential in size.

References

1 L. Babai, R. Beals, J. Cai, G. Ivanyos, and E. M. Luks. Multiplicative equations over commuting matrices. In *Proceedings of SODA 1996*, pages 498–507. ACM/SIAM, 1996.

2 G. Baumslag and D. Solitar. Some two-generator one-relator non-Hopfian groups. *Bulletin of the American Mathematical Society*, 68(3):199–201, 1962. doi:10.1090/S0002-9904-1962-10745-9.

3 P. Bell, V. Halava, T. Harju, J. Karhumäki, and I. Potapov. Matrix equations and hilbert's tenth problem. *International Journal of Algebra and Computation*, 18(8):1231–1241, 2008. doi:10.1142/S0218196708004925.

4 P. Bell, I. Potapov, and P. Semukhin. On the mortality problem: From multiplicative matrix equations to linear recurrence sequences and beyond. In *Proceedings of MFCS 2019*, pages 83:1–83:15, 2019. doi:10.4230/LIPIcs.MFCS.2019.83.

5 P. Bergsträßer, M. Ganardi, and G. Zetzsche. A characterization of wreath products where knapsack is decidable, 2021. arXiv:2101.06132.

6 F. A. Dudkin and A. V. Treyer. Knapsack problem for baumslag–solitar groups. *Siberian Journal of Pure and Applied Mathematics*, 18(4):43–55, 2018.

7 M. Figelius, M. Ganardi, M. Lohrey, and G. Zetzsche. The complexity of knapsack problems in wreath products. In *Proceedings of ICALP 2020*, pages 126:1–126:18, 2020. doi:10.4230/LIPIcs.ICALP.2020.126.

8 M. Figelius, M. Lohrey, and G. Zetzsche. Closure properties of knapsack semilinear groups, 2019. arXiv:1911.12857.

9 E. Frenkel, A. Nikolaev, and A. Ushakov. Knapsack problems in products of groups. *Journal of Symbolic Computation*, 74:96–108, 2016. doi:10.1016/j.jsc.2015.05.006.

10 M. Ganardi, D. König, M. Lohrey, and G. Zetzsche. Knapsack problems for wreath products. In *Proceedings of STACS 2018*, volume 96 of *LIPIcs*, pages 32:1–32:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.32.

11 M. Ganardi and M. Lohrey, 2020. Personal communication.

12 M. Gromov. Groups of polynomial growth and expanding maps. *Publications Mathématiques de l'Institut des Hautes Études Scientifiques*, 53(1):53–78, 1981.

13 F. Gul, M. Sohrabi, and A. Ushakov. Magnus embedding and algorithmic properties of groups $f/n^{(d)}$. *Transactions of the American Mathematical Society*, 369(9):6189–6206, 2017.

14 M. I. Kargapolov and J. I. Merzljakov. *Fundamentals of the Theory of Groups*. Springer-Verlag, New York, 1979. Translated from the second Russian edition.

15 B. Khoussainov and A. Nerode. Automatic presentations of structures. In *International Workshop on Logic and Computational Complexity*, pages 367–392. Springer, 1994.

16 D. König, M. Lohrey, and G. Zetzsche. Knapsack and subset sum problems in nilpotent, polycyclic, and co-context-free groups. In *Algebra and Computer Science*, volume 677 of *Contemporary Mathematics*, pages 138–153. American Mathematical Society, 2016. `doi:10.1090/conm/677`.

17 M. Lohrey. Knapsack in hyperbolic groups. *Journal of Algebra*, 545:390–415, 2020. `doi:10.1016/j.jalgebra.2019.04.008`.

18 M. Lohrey, B. Steinberg, and G. Zetzsche. Rational subsets and submonoids of wreath products. *Information and Computation*, 243:191–204, 2015. `doi:10.1016/j.ic.2014.12.014`.

19 M. Lohrey and G. Zetzsche. Knapsack in graph groups, HNN-extensions and amalgamated products. In *Proceedings of STACS 2016*, volume 47 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50:1–50:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.STACS.2016.50`.

20 M. Lohrey and G. Zetzsche. Knapsack in graph groups. *Theory of Computing Systems*, 62(1):192–246, 2018. `doi:10.1007/s00224-017-9808-3`.

21 M. Lohrey and G. Zetzsche. Knapsack and the power word problem in solvable baumslag-solitar groups. In *Proceedings of MFCS 2020*, pages 67:1–67:15, 2020. `doi:10.4230/LIPIcs.MFCS.2020.67`.

22 W. Magnus. On a theorem of Marshall Hall. *Annals of Mathematics. Second Series*, 40:764–768, 1939.

23 Y. V. Matiyasevich. *Hilbert's Tenth Problem.* MIT Press, Cambridge, Massachusetts, 1993.

24 J. Matthews. The conjugacy problem in wreath products and free metabelian groups. *Transactions of the American Mathematical Society*, 121(2):329–339, 1966.

25 A. Miasnikov, S. Vassileva, and A. Weiß. The conjugacy problem in free solvable groups and wreath products of abelian groups is in TC^0. *Theory of Computing Systems*, 63(4):809–832, 2019.

26 A. Mishchenko and A. Treier. Knapsack problem for nilpotent groups. *Groups Complexity Cryptology*, 9(1):87–98, 2017.

27 A. Myasnikov, A. Nikolaev, and A. Ushakov. Knapsack problems in groups. *Mathematics of Computation*, 84:987–1016, 2015. `doi:10.1090/S0025-5718-2014-02880-9`.

28 V. Remeslennikov and V. Sokolov. Some properties of a magnus embedding. *Algebra and Logic*, 9(5):342–349, 1970.

Synchronizing Strongly Connected Partial DFAs

Mikhail V. Berlinkov ✉
Institute of Natural Sciences and Mathematics, Ural Federal University, Ekaterinburg, Russia

Robert Ferens ✉
Institute of Computer Science, University of Wrocław, Poland

Andrew Ryzhikov ✉
Université Gustave Eiffel, LIGM, Marne-la-Vallée, France

Marek Szykuła ✉
Institute of Computer Science, University of Wrocław, Poland

Abstract

We study synchronizing partial DFAs, which extend the classical concept of synchronizing complete DFAs and are a special case of synchronizing unambiguous NFAs. A partial DFA is called synchronizing if it has a word (called a reset word) whose action brings a non-empty subset of states to a unique state and is undefined for all other states. While in the general case the problem of checking whether a partial DFA is synchronizing is PSPACE-complete, we show that in the strongly connected case this problem can be efficiently reduced to the same problem for a complete DFA. Using combinatorial, algebraic, and formal languages methods, we develop techniques that relate main synchronization problems for strongly connected partial DFAs with the same problems for complete DFAs. In particular, this includes the Černý and the rank conjectures, the problem of finding a reset word, and upper bounds on the length of the shortest reset words of literal automata of finite prefix codes. We conclude that solving fundamental synchronization problems is equally hard in both models, as an essential improvement of the results for one model implies an improvement for the other.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Mathematics of computing → Combinatorics

Keywords and phrases Černý conjecture, literal automaton, partial automaton, prefix code, rank conjecture, reset threshold, reset word, synchronizing automaton, synchronizing word

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.12

Related Version *Full Version*: `https://arxiv.org/abs/2101.05057`

Funding *Robert Ferens*: Supported in part by the National Science Centre, Poland under project number 2017/25/B/ST6/01920.
Marek Szykuła: Supported in part by the National Science Centre, Poland under project number 2017/25/B/ST6/01920.

Acknowledgements We are grateful to the anonymous reviewers for useful comments and recent literature references.

1 Introduction

Synchronization is a concept in various domains of computer science which consists in regaining control over a system by applying (or observing) a specific set of input instructions. These instructions are usually required to lead the system to a fixed state no matter in which state it was at the beginning. This idea has been studied for automata (deterministic [8, 36], non-deterministic [17], unambiguous [2], weighted and timed [10], partially observable [22], register [1], nested word [9]), parts orienting in manufacturing [13, 23], testing of reactive systems [30], variable length codes [5], and Markov Decision Processes [11, 12].

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 12; pp. 12:1–12:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we study the synchronization of partial DFAs, which are a generalization of complete DFAs and a special case of unambiguous NFAs. We are motivated by applications of this model and its connections with others, as well as the need for new techniques applied to partial DFAs. The problems for strongly connected partial DFAs are a motivation for further development and generalization of the methods applied for complete DFAs, since, as we show, these models are closely related. We also hope that our methods will serve as a step toward studying a wider class of strongly connected unambiguous NFAs.

1.1 Observing a reactive system

Consider a finite-state reactive system modeled by a partial DFA (by partial we mean that for some states there can be no outgoing transitions corresponding to some letters). The observer knows the structure of the DFA but does not know its current state. At every step, the DFA reads a letter (also known to the observer) and transits to another state. The observer wants to eventually learn the actual state of the DFA. Since the DFA is deterministic, once a state is known, it will be known forever.

In this setting, the actual state is known if and only if the system reads a *reset word* – a word which transits a non-empty set of states to a single state and is undefined for all other states. The presence of undefined transitions indicates that certain actions cannot be performed from certain states, which can be essential for synchronization.

For several identical systems running in parallel and receiving the same input (but possibly starting from different states), the presence of a reset word guarantees that all systems end up in the same state. This idea can be used in robotics, where a sequence of passive obstacles is used for orienting a large number of arbitrarily rotated parts arriving simultaneously on a conveyor belt ([13, 23], see also [36] for an illustrative example).

Reactive systems (such as Web servers, communication protocols, operating systems and processors) are systems developed to run without termination and interact through visible events, so it is natural to assume that the system can return to any state from any other state (NFAs with this property are called *strongly connected*). The probabilistic version of the described problem for strongly connected partial DFAs has been considered in the context of ε-machines [34]. In particular, the observer knows the state of an ε-machine precisely if and only if a reset word for the underlying partial DFA was applied. Some experimental results on finding shortest reset words for partial DFAs were recently presented in [31].

1.2 Synchronizing automata

There exist several definitions that generalize the notion of a synchronizing complete DFA to larger classes of NFAs. In this subsection, we describe the notion which preserves most of the properties of the complete DFAs case, and in 1.5 we briefly describe alternative notions.

An NFA is called *unambiguous* if for every two states p, q and every word w, there is at most one path from p to q labeled by w [2]. In the strongly connected case, this is equivalent to a more classical definition of an unambiguous automaton with chosen initial and final states, if there is a unique initial state and a unique final state. An unambiguous NFA is called *synchronizing* if there exist two non-empty subsets C, R of its states and a word w (called a *reset* word) such that its action maps every state in C exactly to the whole set R, and is undefined for all states outside C [2]. For partial DFAs, the set R has size one [5], and, for complete DFAs, the set C is also the whole set of states [36].

Partial DFAs are a natural intermediate class between unambiguous NFAs and complete DFAs. The bounds on the length of shortest reset words in strongly connected partial DFAs have not been studied before. The famous Černý conjecture, which is one of the most longstanding open problems in automata theory, states that for an n-state complete DFA we can always find a reset word of length at most $(n-1)^2$, unless there are no reset words. The best known upper bound is cubic in n [32, 33], and the problem of deciding whether a complete DFA is synchronizing is solvable in time quadratic in n [36]. For an n-state strongly connected unambiguous NFA the best known upper bound on the length of a shortest reset word is n^5, and the existence of a reset word is verifiable in time $\mathcal{O}(n^5)$ [28]. The same upper bound holds for the length of the shortest mortal words in strongly connected unambiguous NFAs [21], whereas partial DFAs admit a tight quadratic bound [26].

1.3 Synchronizing codes

A *variable length code* X (which we call a *code*) is a set of finite words over a finite alphabet Σ, such that no word over Σ can be written as a concatenation of codewords of X in two different ways. Such codes (especially Huffman codes [16]) are widely used for lossless data compression. Since the lengths of codewords can be different, one transmission error can spoil the whole decoding process, causing a major data loss. Also, in general, decoding a part of a message (e.g., a segment of a compressed video stream) is not possible without decoding the whole message.

These issues can be addressed by using synchronizing codes. A code X is called *synchronizing* if there exists a *synchronizing* word $w \in X^*$ such that for every $uwv \in X^*$ we have $uw, wv \in X^*$. The occurrence of the word ww thus stops error propagation and allows parallel decoding of the two parts of the message. More generally, each appearance of the word ww in a coded message allows running decoding independently from the position after the first w.

A code is called *prefix* if none of its codewords is a prefix of another codeword. Such codes allow obtaining the correct partition of a message into codewords one by one by going from left to right. Even if a code is synchronizing, there are no guarantees that a synchronizing word will appear in a message. Codes where every long enough concatenation of codewords is synchronizing are called *uniformly synchronizing* [5, 7]. A prefix code is called *maximal* if it is not a subset of another prefix code. All non-trivial uniformly synchronizing finite prefix codes are non-maximal [5].

1.4 Automata for X^*

A code recognized by an NFA as a language is called *recognizable*. In particular, every finite code is recognizable. To argue about synchronization properties of a recognizable code X, special NFAs recognizing X^* are studied. These NFAs have a unique initial and final state r such that the set of words labeling paths from r to itself coincides with X^*, thus they are also strongly connected. Provided a recognizable code X, an NFA with the described properties can be chosen to be unambiguous [5]. Moreover, this NFA can be chosen to be a partial (respectively, a complete) DFA if and only if X is a recognizable prefix (respectively, recognizable maximal prefix) code [5].

For such an unambiguous NFA with the properties as above, X is synchronizing if and only if the NFA is synchronizing, and the length of a shortest synchronizing word for X is at most the length of a shortest reset word of the NFA plus twice its number of states [5, Chapter 4].

Finite prefix codes admit a direct construction of partial DFAs with the described properties, called literal automata. Let X be a finite prefix code over an alphabet Σ. The *literal automaton* $\mathscr{A}_X = (Q, \Sigma, \delta)$ is constructed as follows. The set of states Q is the set of all proper prefixes of the words in X, the transition function is defined as follows: $\delta(q, x) = qx$

if $qx \notin X$ and qx is a proper prefix of a word in X, $\delta(q,x) = \varepsilon$ if $qx \in X$, and $\delta(q,x) = \bot$ otherwise. The state corresponding to the empty prefix ε is called the *root state*. The *height* of a literal automaton is the length of a longest path of its transitions without repetition of states; equivalently, this is the length of the longest word in X minus one. Note that the number of states of $\mathscr{A}_X$ is at most the total length of all codewords of X, which allows to directly transfer upper bounds from literal automata to finite prefix codes. An example of a literal automaton is shown in Fig. 1(right). The literal automaton of a prefix code can be used as a decoder for this code by adding output labels to the transitions [5].

1.5 Carefully synchronizing DFAs

For general NFAs, synchronizability can be generalized to Di-directability for $i = 1, 2, 3$ [17]. As discussed in [38, Section 6.3], for partial DFAs the notions of D1- and D3-directing words both coincide with *carefully synchronizing* words. These are words sending every state of a partial DFA to the same state, not using any undefined transitions at all. A D2-directing word for a partial DFA is either carefully synchronizing or mortal (undefined for every state). The definitions of carefully synchronizing and D2-directing words are different from our definition of synchronizing words.

A carefully synchronizing word can be applied to a partial DFA at any moment without the risk of using an undefined transition. This comes at a high cost: even for strongly connected partial DFAs, the shortest carefully synchronizing words can have exponential length [38, Proposition 9], and the problem of checking the existence of such a word is PSPACE-complete [38, Theorem 12], in contrast with the case of complete DFAs. On the contrary, the notion of a synchronizing partial DFA preserves most of the properties of a synchronizing complete DFA, at least in the strongly connected case. Note that every carefully synchronizing word is synchronizing, but the converse is not true.

While for complete DFAs the property of being strongly connected is not essential for many synchronization properties [36], the situation changes dramatically for partial DFAs. Partial DFAs, which are not strongly connected, can have exponentially long shortest reset words, and the problem of checking the existence of a reset word is PSPACE-complete [3]. Thus, strong connectivity is indeed necessary to obtain good bounds and algorithms. As explained above, for reactive systems and prefix codes this requirement comes naturally.

1.6 Our contribution and organization of the paper

We prove a number of results for strongly connected partial DFAs connected with the Černý conjecture and its generalizations. Where possible, we look up for methods that allow relating the partial case with the complete case, instead of directly reproving known results in this more general setting. In this way, we do not have to go into the existing proofs, and future findings concerning the complete case should be often immediately transferable to the partial case.

We start from basic properties and introduce more advanced techniques along with their applications. First, we investigate the rank conjecture, which is a generalization of the Černý conjecture from the case of synchronizing automata to the case of all automata. We show that the rank conjecture for complete DFAs implies it also for partial DFAs (Theorem 10). For this, we introduce our first basic tool called *fixing automaton*, which is a complete automaton obtained from a partial one and sharing some properties. Our result shows a general way for transferring upper bounds from the case of complete DFAs to partial DFAs, e.g., we immediately get that the rank conjecture holds for partial Eulerian automata (Corollary 11).

To connect the Černý conjecture for the cases of complete and partial DFAs, we need more involved techniques, since the construction developed for the rank conjecture does not preserve the property of being synchronizing. We introduce a *collecting automaton*, which extends the concept of the fixing automaton. We use it to show that all upper bounds on the length of the shortest reset words, up to a subquadratic additive component, are equivalent for partial and complete DFAs (Theorem 17). We also use it to prove that the problems of determining synchronizability and finding a reset word of a strongly connected partial DFA can be effectively reduced to the same problem for a complete DFA (Section 3.5). This also means that possible improvements of the complexity of the best-known algorithms for these problems for complete DFAs should directly apply to partial DFAs.

As discussed in 1.3 and 1.4, one of the main motivations for studying synchronization of strongly connected partial DFAs is a direct correspondence with synchronization of recognizable prefix codes. An important special case is when the prefix code is finite. We investigate it by studying literal automata of finite prefix codes and obtain stronger upper bounds than those for the general case of strongly connected partial (or complete) DFAs. We show that the length of the shortest reset words for literal automata of finite prefix codes is at most $\mathcal{O}(n \log^3 n)$, where n is the number of states of the automaton (Corollary 23). This upper bound is the same as the best known for maximal prefix codes (which becomes now a special case), but it is not transferred directly, as key statements do not hold in the same way for non-maximal prefix codes. To prove it, we first show that literal automata of finite prefix codes admit a word of linear length whose action sends all the states to a non-empty subset of small size (Theorem 22). It establishes a natural combinatorial property of finite prefix codes and constitutes the most involved proof in this paper. Once we show the existence of such a word, we use one more construction called the *induced automaton*, which is a generalization of linear algebraic techniques to the case of partial DFAs (Section 3.7). This particular construction extends the existing techniques originally developed for complete DFAs but simultaneously comes with a new simpler and more general proof.

Finally, we show that the lower bounds for strongly connected partial DFAs are asymptotically the same even if we ensure the existence of undefined transitions (Section 4). In other words, undefined transitions do not help in general, as we cannot significantly improve upper bounds for such automata without doing that for the complete case.

2 Preliminaries

A *partial deterministic finite automaton* $\mathscr{A}$ (which we call *partial automaton* throughout the paper) is a triple (Q, Σ, δ), where Q is a set of *states*, Σ is an *input alphabet*, and δ is partial function $Q \times \Sigma \to Q$ called the *transition function*. Note that the automata we consider do not have any initial or final states. We extend δ to a partial function $Q \times \Sigma^* \to Q$ as usual: we set $\delta(q, wa) = \delta(\delta(q, w), a)$ for $w \in \Sigma^*$ and $a \in \Sigma$. For a state $q \in Q$ and a word $w \in \Sigma^*$, if the action $\delta(q, w)$ is undefined, then we write $\delta(q, w) = \perp$. Note that if $\delta(q, w) = \perp$ for a word $w \in \Sigma^*$, then $\delta(q, wu) = \perp$ for every word $u \in \Sigma^*$. An automaton is *complete* if all its transitions are defined, and it is *incomplete* otherwise. An automaton is *strongly connected* if for every two states $p, q \in Q$ there is a word $w \in \Sigma^*$ such that $\delta(p, w) = q$.

By Σ^i we denote the set of all words over Σ of length exactly i and by $\Sigma^{\le i}$ the set of all words over Σ of length at most i. For two sets of words $W_1, W_2 \in \Sigma^*$, by $W_1 W_2$ we denote their product $\{w_1 w_2 \in \Sigma^* \mid w_1 \in W_1, w_2 \in W_2\}$. The empty word is denoted by ε. Throughout the paper, by n we always denote the number of states $|Q|$.

Given $S \subseteq Q$, the *image* of S under the action of w is $\delta(S, w) = \{\delta(q, w) \mid q \in S, \delta(q, w) \neq \perp\}$. The *preimage* of S under the action of w is $\delta^{-1}(S, w) = \{q \in Q \mid \delta(q, w) \in S\}$. Since

$\mathscr{A}$ is deterministic, for disjoint subsets $S, T \subseteq Q$, their preimages under the action of every word $w \in \Sigma^*$ are also disjoint.

The *rank* of a word w is the size of the image of Q under the action of this word, i.e., $|\delta(Q, w)|$. In contrast with complete automata, partial automata may admit words of rank zero; these words are called *mortal*. Words of non-zero rank are called *non-mortal*. A word of rank 1 is called *reset*, and if the automaton admits such a word then it is *synchronizing*. The *reset threshold* $\mathrm{rt}(\mathscr{A})$ is the length of the shortest reset words of $\mathscr{A}$.

We say that a word w *compresses* a subset $S \subseteq Q$, if $|\delta(S, w)| < |S|$ but $\delta(S, w) \neq \emptyset$. A subset that admits a compressing word is called *compressible*. There are two ways to compress a subset $S \subseteq Q$ with $|S| \geq 2$. One possibility is the *pair compression*, which is the same as in the case of a complete automaton, i.e., mapping at least two states $p, q \in S$ to the same state (but not to $\perp$). The other possibility is to map at least one state from S to $\perp$, but not all states from S to $\perp$. Sometimes, a subset can be compressed in both ways simultaneously.

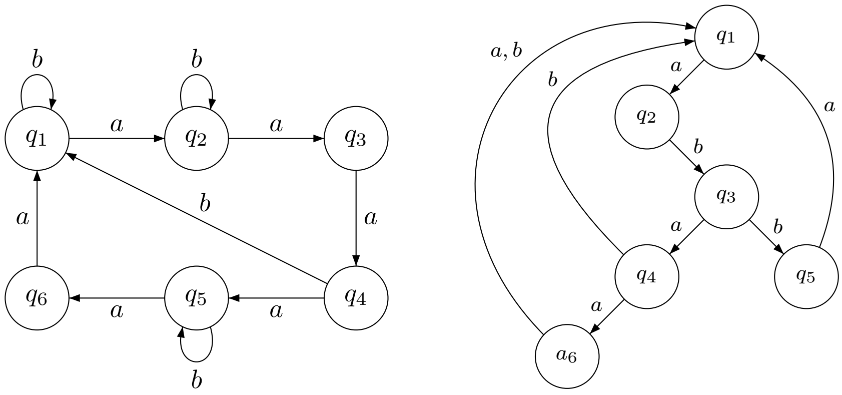


Figure 1 Left: a strongly connected partial 6-state binary automaton; right: the literal automaton of the prefix code $\{abaaa, abaab, abab, abba\}$.

An example of a strongly connected partial automaton is shown in Fig. 1 (left). We have two undefined transitions: $\delta(q_3, b) = \delta(q_6, b) = \perp$. The unique shortest reset word is bab: $\delta(Q, b) = \{q_1, q_2, q_5\}$, $\delta(Q, ba) = \{q_2, q_3, q_6\}$, and $\delta(Q, bab) = \{q_2\}$. However, in contrast with the case of a complete automaton, the preimage $\delta^{-1}(\{q_2\}, bab) = \{q_1, q_4\}$ is not Q.

3 Upper Bounds

3.1 Inseparability Equivalence

Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a partial automaton. We define the inseparability relation $\equiv$ on Q. Two states are *separable* if they can be separated by mapping exactly one of them to $\perp$, leaving the other one.

▶ **Definition 1.** *The* inseparability equivalence $\equiv$ *on* Q *is defined as follows:*

$$p \equiv q \quad \textit{if and only if} \quad \forall_{u \in \Sigma^*} \ (\delta(p, u) \in Q \Leftrightarrow \delta(q, u) \in Q)\,.$$

The same relation is considered in [5, Section 1.4] if all states of the partial automaton are final. Also, if we replace $\perp$ with a unique final state, then $\equiv$ is the well-known Myhill-Nerode congruence on words in a complete automaton. Under a different terminology, it also appears in the context of ε-machines, where non-equivalent states are called *topologically distinct* [34].

For a subset $S \subseteq Q$, let $\kappa(S)$ be the number of equivalence classes that have a non-empty intersection with S. In the automaton from Fig. 1(left) we have three equivalence classes, where $q_1 \equiv q_4$, $q_2 \equiv q_5$, and $q_3 \equiv q_6$.

Our first auxiliary lemma states that a subset $S \subseteq Q$ that intersects at least two equivalence classes can be compressed and the number of intersected classes can be decreased with a short word. This is done by mapping to $\perp$ all the states of S from at least one equivalence class, but not the whole set S. A linear upper bound can be inferred from a standard analysis of the corresponding Myhill-Nerode congruence (e.g., [35]), but we will need a more precise bound in terms of $\kappa(S)$.

▶ **Lemma 2.** *Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a partial automaton, and let $S \subseteq Q$ be a subset such that $\kappa(S) \geq 2$. Then there is a word $w \in \Sigma^*$ of length at most $\kappa(Q) - \kappa(S) + 1 \leq n - |S| + 1$ and such that $1 \leq \kappa(\delta(S, w)) < \kappa(S)$.*

By an iterative application of Lemma 2, we can easily compress any subset of states to a subset of a single equivalence class.

▶ **Corollary 3.** *Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a partial automaton, and let $S \subseteq Q$ be a non-empty subset. There is a word w of length at most $(\kappa(S) - 1)(\kappa(Q) - \kappa(S)/2)$ such that $\delta(S, w)$ is non-empty and is contained in one inseparability class.*

3.2 Fixing Automaton

The other possibility of compressing a subset in a partial automaton is the classical pair compression. This is the only way for compressing a subset with all states in one equivalence class, which is always the case in a complete automaton.

Our next tool to deal with this way of compression is the *fixing automaton*. This is a complete automaton obtained from a partial one, defined as follows.

▶ **Definition 4** (Fixing automaton). *For a partial automaton $\mathscr{A}(Q, \Sigma, \delta)$, the* fixing automaton *is the complete automaton $\mathscr{A}^{\mathrm{F}} = (Q, \Sigma, \delta^{\mathrm{F}})$ such that the states are fixed instead of having an undefined transition: for every $q \in Q$ and $a \in \Sigma$, we have $\delta^{\mathrm{F}}(q, a) = q$ if $\delta(q, a) = \perp$, and $\delta^{\mathrm{F}}(q, a) = \delta(q, a)$ otherwise.*

We list some useful properties of the fixing automaton.

▶ **Lemma 5.** *Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a partial automaton, let $S \subseteq Q$, and let $w \in \Sigma^*$. We have $\delta(S, w) \subseteq \delta^{\mathrm{F}}(S, w)$. Moreover, if for every state $q \in S$ we have $\delta(q, w) \neq \perp$, then $\delta(S, w) = \delta^{\mathrm{F}}(S, w)$.*

▶ **Lemma 6.** *Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a partial automaton and let $S \subseteq Q$ be a non-empty subset. For every word $w \in \Sigma^*$, there exists a word $w' \in \Sigma^*$ of length $|w'| \leq |w|$ such that $\emptyset \neq \delta(S, w') \subseteq \delta^{\mathrm{F}}(S, w)$. In particular, if w has rank r in $\mathscr{A}^{\mathrm{F}}$, then w' has rank $1 \leq r' \leq r$ in $\mathscr{A}$.*

▶ **Corollary 7.** *The minimal non-zero rank of a partial automaton $\mathscr{A}$ is at most the minimal rank of $\mathscr{A}^{\mathrm{F}}$.*

In the general case of a partial automaton, it can happen that we cannot compress some subset S even if there exists a word of non-zero rank smaller than $|S|$. This is the reason why the shortest words of the minimal non-zero rank can be exponentially long and why deciding if there is a word of a given rank is PSPACE-complete [3].

However, in the case of a strongly connected partial automaton, as well as for a complete automaton, every non-mortal word can be extended to a word of the minimal non-zero rank. This is a fundamental difference that allows constructing compressing words iteratively. Note that the fixing automaton of a strongly connected partial one is also strongly connected.

▶ **Lemma 8.** *Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a strongly connected partial automaton, and let r be the minimal non-zero rank over all words. For every non-empty subset $S \subseteq Q$, there exists a non-mortal word w such that $|\delta(S, w)| \le r$.*

3.3 Rank Conjecture

The *rank conjecture* (sometimes called *Černý-Pin conjecture*) is a well-known generalization of the Černý conjecture to non-synchronizing automata (e.g., [25]). The rank conjecture is a weaker version of the conjecture originally stated by Pin that was not restricted to the minimal rank and turned out to be false [18]. Some further results on the rank conjecture for strongly connected complete automata are provided in [20].

▶ **Conjecture 9** (The rank conjecture)**.** *For an n-state complete automaton where r is the minimal rank over all words, there exists a word of rank r and of length at most $(n-r)^2$.*

For partial automata, the rank conjecture is analogous with the exception that r is the minimal non-zero rank.

▶ **Theorem 10.** *Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a strongly connected partial automaton. If the rank conjecture holds for the fixing automaton $\mathscr{A}^{\mathrm{F}}$, then it also holds for $\mathscr{A}$.*

Proof. Let r be the minimal rank in $\mathscr{A}^{\mathrm{F}}$ over all words. From the conjecture and by Lemma 6, there exists a word w' of length at most $(n-r)^2$ and such that $\emptyset \neq \delta(Q, w') \subseteq \delta^{\mathrm{F}}(Q, w)$.

Let $r' \le r$ be the minimal rank of $\mathscr{A}$. For every $s = r, r-1, \dots, r'+1$, we inductively construct a word of non-zero rank less than s, of length at most $(n-(s-1))^2$, and such that w' is its prefix. Let $w'v$ be a word of rank at most s and of length at most $(n-s)^2$, and let $S = \delta(Q, w'v)$. Suppose that $\kappa(S) = 1$. Since s is not the minimal rank of $\mathscr{A}$, by Lemma 8, S must be compressible. Since its states are inseparable, there must be two distinct states $p, q \in S$ and a word u such that $\delta(q, u) = \delta(p, u) \neq \perp$. But (from Lemma 5) $\{p, q\} \subseteq \delta(Q, w'v) \subseteq \delta^{\mathrm{F}}(Q, w'v) \subseteq \delta^{\mathrm{F}}(Q, wv)$, thus $\delta^{\mathrm{F}}(Q, wv)$ is compressible in $\mathscr{A}^{\mathrm{F}}$, which contradicts that w has the minimal rank in $\mathscr{A}^{\mathrm{F}}$. Hence $\kappa(S) \ge 2$, and by Lemma 2, $\delta(Q, w'v)$ can be compressed with a word u of length at most $n-s+1$. We have $|w'vu| \le (n-s)^2 + n - s + 1 \le (n-(s-1))^2$, which proves the induction step. ◀

The theorem implies that, in the strongly connected case, the rank conjecture is true for complete automata if and only if it is true for partial automata. For instance, we immediately get the result for the class of Eulerian automata. A partial automaton is *Eulerian* if it is strongly connected and the numbers of outgoing and incoming transitions are the same at every state, i.e., for every $q \in Q$, we have $|\{a \in \Sigma \mid \delta(q, a) \in Q\}| = |\{(p, a) \in Q \times \Sigma \mid \delta(p, a) = q\}|$. The following corollary follows from the facts that the rank conjecture holds for complete Eulerian automata [20] and that the fixing automaton of a partial Eulerian automaton is also Eulerian.

▶ **Corollary 11.** *The rank conjecture is true for partial Eulerian automata.*

3.4 Collecting Automaton

The fixing automaton allows relating the behavior of words in a partial and a complete automaton, but its main disadvantage is that it is not necessarily synchronizing. Therefore, we will need one more tool, called *collecting automaton*. It is an extension of the fixing automaton by an additional letter that allows a quick synchronization into one inseparability class, while it does not affect the length of a shortest synchronizing word for any particular inseparability class.

By $\mathscr{A}/_{\equiv} = (Q_{\mathscr{A}/_{\equiv}}, \Sigma, \delta_{\mathscr{A}/_{\equiv}})$, we denote the quotient automaton by the inseparability relation. $\mathscr{A}/_{\equiv}$ is also a partial automaton, and if $\mathscr{A}$ is strongly connected, then so is $\mathscr{A}/_{\equiv}$. By $[p] \in Q_{\mathscr{A}/_{\equiv}}$, we denote the class of a state $p \in Q$ of the original automaton $\mathscr{A}$.

A *collecting tree* of $\mathscr{A}$ is a tree T with the set of vertices $Q_{\mathscr{A}/_{\equiv}}$ and directed edges labeled by letters from Σ in the following way: (a) Edges are labeled by letters from Σ and correspond to transitions in $\mathscr{A}/_{\equiv}$: each edge $([p], [p'], a)$ is such that $\delta_{\mathscr{A}/_{\equiv}}([p], a) = [p']$. (b) There is a root $[r]$ such that the tree is directed toward it. See Fig. 2 in Appendix for an example. Equivalently, it can be seen as a specific partial automaton being a subautomaton of $\mathscr{A}/_{\equiv}$ whose underlying digraph is a tree directed toward one state. An automaton can have many collecting trees, even for the same $[r]$, and every strongly connected automaton has a collecting tree for every class $[r]$.

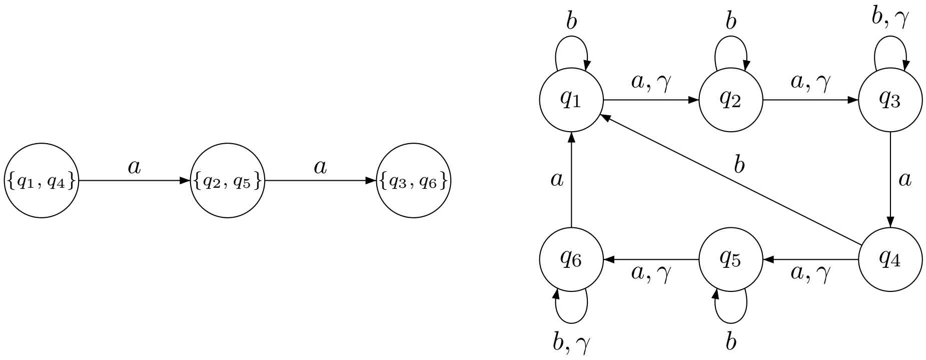


Figure 2 Left: a collecting tree with root $[q3] = \{q_3, q_6\}$; right: the corresponding collecting automaton of the example from Fig. 1(left).

▶ **Definition 12** (Collecting automaton). *Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a strongly connected partial automaton and let T be one of its collecting trees with a root $[r]$. The* collecting automaton *$\mathscr{A}^{\mathrm{C}(T)} = (Q, \Sigma \cup \{\gamma\}, \delta^{\mathrm{C}(T)})$ is defined as follows:*

- *The transition function $\delta^{\mathrm{C}(T)}$ on Σ is defined as in the fixing automaton $\mathscr{A}^{\mathrm{F}}$.*
- *$\gamma \notin \Sigma$ is a fresh letter. Its action is defined according to the edges in T: Let $q_1 \in Q \setminus [r]$ be a state. Since T is a tree directed toward $[r]$, there is exactly one edge outgoing from $[q_1]$, say $([q_1], [q_2], a) \in T$ for some $[q_2] \in Q_{\mathscr{A}/_{\equiv}}$ and $a \in \Sigma$. We set $\delta^{\mathrm{C}(T)}(q_1, \gamma) = \delta(q_1, a)$. Finally, the transition of γ on each state in $[r]$ is the identity.*

A collecting automaton is always strongly connected, as it contains all transitions of the fixing automaton. We prove several properties connecting partial automata and their collecting complete automata. They are preliminary steps toward relating the Černý conjecture for strongly connected partial and complete automata.

▶ **Lemma 13.** *Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a strongly connected partial automaton, and let T be one of its collecting trees with a root $[r]$. If there is a word over $\Sigma \cup \{\gamma\}$ synchronizing $[r]$ in $\mathscr{A}^{\mathrm{C}(T)} = (Q, \Sigma \cup \{\gamma\}, \delta^{\mathrm{C}(T)})$, then there is such a word over Σ that is not longer.*

▶ **Lemma 14.** *Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a strongly connected partial automaton and let T be one of its collecting trees. Then $\mathscr{A}$ is synchronizing if and only if the collecting automaton $\mathscr{A}^{\mathrm{C}(T)} = (Q, \Sigma \cup \{\gamma\}, \delta^{\mathrm{C}(T)})$ is synchronizing.*

This also means that the choice of T does not matter: $\mathscr{A} = (Q, \Sigma, \delta)$ is synchronizing if and only if all $\mathscr{A}^{\mathrm{C}(T)}$ are synchronizing.

3.5 Algorithmic Issues

Checking if a strongly connected partial automaton is synchronizing and finding a minimum-rank word can be done similarly as for a complete automaton, by a suitable generalization of the well-known Eppstein algorithm [13, Algorithm 1]. The same algorithm for checking synchronizability, under different terminology, was described in the context of ε-machines [34].

▶ **Proposition 15.** *Checking if a given strongly connected partial automaton with n states over an alphabet Σ is synchronizing can be done in $\mathcal{O}(|\Sigma|n^2)$ time and $\mathcal{O}(n^2 + |\Sigma|n)$ space. Finding a word of the minimum rank can be done in $\mathcal{O}(|\Sigma|n^3)$ time and $\mathcal{O}(n^2 + |\Sigma|n)$ space.*

Furthermore, the problem of checking the synchronizability of a strongly connected partial automaton can be reduced in smaller time to the case of a complete automaton.

▶ **Theorem 16.** *Given a strongly connected partial automaton, in $\mathcal{O}(|\Sigma|n \log n)$ time, we can construct a complete automaton that is synchronizing if and only if the given partial automaton is synchronizing.*

Proof. We can compute all inseparability classes in $\mathcal{O}(|\Sigma|n \log n)$ time. This is done by the Hopcroft minimization algorithm [15], if we interpret the partial DFA as a language-accepting DFA with an arbitrary initial state and the sink state $\perp$ that is its only final state.

Having computed the classes, we can construct a collecting automaton for an arbitrary collecting tree. Note that it can be done in $\mathcal{O}(|\Sigma|n)$ time by a breadth-first search from a class $[r]$. The desired property follows from Lemma 14. ◀

3.6 Černý Conjecture

The famous Černý conjecture is the rank conjecture for $r = 1$. Let $\mathfrak{C}(n)$ be the maximum length of the shortest reset words of all n-state synchronizing complete automata. It is well known that $\mathfrak{C}(n) \geq (n-1)^2$ [8]. The Černý conjecture states that $\mathfrak{C}(n) = (n-1)^2$, but the best proved upper bound is cubic [32, 33].

Let $\mathfrak{C}_{\mathrm{P}}(n)$ be the maximal length of the shortest reset words of all n-state synchronizing strongly connected partial automata. We show that if the Černý conjecture is true (or another upper bound holds), then a slightly weaker upper bound holds for synchronizing strongly connected partial automata.

To prove the following theorem, we combine several techniques, in particular, the inseparability equivalence, the collecting automaton, and an algebraic upper bound on the reset threshold of a complete automaton with a word of small rank [4].

▶ **Theorem 17.**

$$\mathfrak{C}_{\mathrm{P}}(n) \leq \mathfrak{C}(n) + \mathcal{O}(n^{4/3}).$$

Proof. Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a synchronizing partial automaton with n states. Let T be a collecting tree of $\mathscr{A}$ with a root class $[r]$ containing the smallest number of states. We consider the collecting automaton $\mathscr{A}^{\mathrm{C}(T)} = (Q, \Sigma \cup \{\gamma\}, \delta^{\mathrm{C}(T)})$. By Lemma 14, $\mathscr{A}^{\mathrm{C}(T)}$ is synchronizing. We have two cases, depending on the number $\kappa(Q)$ of inseparability classes of $\mathscr{A}$.

First, suppose that $\kappa(Q) \le n^{2/3}$. Then, by Corollary 3 (for $S = Q$), there is a word v of length at most $(\kappa(Q)-1)\kappa(Q)/2 < n^{4/3}$ such that $\delta(Q, v)$ is non-empty and is contained in one equivalence class, say $[p]$. Since $\mathscr{A}$ is strongly connected, there is a word $u_{p \to r}$ of length at most $n-1$ whose action maps $[p]$ into $[r]$. Let w' be a reset word for $\mathscr{A}^{\mathrm{C}(T)}$ of length at most $\mathfrak{C}(n)$. In particular, w' synchronizes $[r]$, so by Lemma 13, we get a word w of length at most $\mathfrak{C}(n)$ that synchronizes $[r]$ in $\mathscr{A}$. Then, $v u_{p \to r} w$ is a reset word for $\mathscr{A}$ of length at most $n^{4/3} + n - 1 + \mathfrak{C}(n)$.

In the second case, we have $\kappa(Q) > n^{2/3}$. Then the size of $[r]$, which has been chosen to have the smallest size, has at most $n^{1/3}$ states. Note that γ^{n-1} is a word of rank at most $n^{1/3}$. Then we can apply Corollary 20 (cf. [4, Theorem 2]) for $\mathscr{A}^{\mathrm{C}(T)}$ with this word, obtaining that the reset threshold of $\mathscr{A}^{\mathrm{C}(T)}$ is upper bounded by $(n-1) + 2(n-1)\mathfrak{C}(n^{1/3})$. Using the well-known general cubic upper bound $n^3/6$ on the reset threshold of a complete automaton (e.g., [25]), we get that there is a reset word w' for $\mathscr{A}^{\mathrm{C}(T)}$ of length at most $(n-1) + 2(n-1)n/6$. Now, we return to $\mathscr{A}$. By Corollary 3 (for $S = Q$), we get a word v of length at most $(n-1)n/2$ such that $\delta(Q, u)$ is non-empty and is contained in one equivalence class $[p]$. As in the first case, there is a word $u_{p \to r}$ of length at most $n-1$ whose action maps $[p]$ into $[r]$. By Lemma 13, from w' we obtain a word w that synchronizes $[r]$ and is no longer than $(n-1) + 2(n-1)n/6$. Finally, $v u_{p \to r} w$ is a reset word for $\mathscr{A}$ of length at most $(n-1)n/2 + (n-1) + (n-1) + 2(n-1)n/6 \le (n-1)^2$ for $n \ge 18$.

From both cases, we conclude that $\mathrm{rt}(\mathscr{A}) \le \mathfrak{C}(n) + \mathcal{O}(n^{4/3})$. ◀

From Theorem 17, it follows that all upper bounds on the reset threshold of a complete automaton transfer to upper bounds for partial automata, up to a subquadratic component. Thus $\mathfrak{C}_{\mathrm{P}}(n) \le 0.1654n^3 + \mathcal{O}(n^2)$ [32]. It can be also seen from the proof that if we have a better general upper bound on $\mathfrak{C}(n)$, we get a smaller additional term, e.g., if $\mathfrak{C}(n) \in \mathcal{O}(n^2)$, then $\mathfrak{C}_{\mathrm{P}}(n) = \mathfrak{C}(n) + \mathcal{O}(n)$, so the additional term is at most linear. The additional component is likely not needed, but it is difficult to completely get rid of it in general, as for that we could not lengthen by any means the reset word assumed for a complete automaton. However, it is easy to omit it when reproving particular bounds for complete automata, both combinatorial [25] and based on avoiding words [32, 33]. We conjecture that $\mathfrak{C}_{\mathrm{P}}(n) = \mathfrak{C}(n)$ for all n.

3.7 Induced Automaton

We develop an algebraic technique applied to partial automata. It will allow us deriving upper bounds on reset thresholds, in particular, in the cases when there exists a short word of a small rank, which is the case of the literal automaton of a prefix code. We base on the results from [4] for complete automata and generalize them to be applied to partial automata. The existing linear algebraic proofs for complete automata do not work for partial ones, because the matrices of transitions may not have a constant sum of the entries in each row.

We need to introduce a few definitions from linear algebra for automata (see, e.g., [4, 19, 24, 33]). Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a partial DFA. Without loss of generality we assume that $Q = \{1, \ldots, n\}$. By $\mathbb{R}^n$, we denote the real n-dimensional linear space of row vectors. For a vector $v \in \mathbb{R}^n$ and an $i \in Q$, we denote the vector's value at the i-th position by $v(i)$. Similarly, for a matrix M, we denote its value in an i-th row and a j-th column by $M(i,j)$.

A vector g is *non-negative* if $g(i) \geq 0$ for all i, and it is *non-zero* if $g(i) \neq 0$ for some i. For a word $w \in \Sigma^*$, by $\mathrm{M}(w)$ we denote the $n \times n$ matrix of the transformation of w in δ: $\mathrm{M}(w)(p,q) = 1$ if $\delta(p,w) = q$, and $\mathrm{M}(w)(p,q) = 0$ otherwise. Note that if $\delta(p,w) = \bot$, then we have $\mathrm{M}(w)(p,q) = 0$ for all $q \in Q$. The usual scalar product of two vectors u, v is denoted by $u \odot v$. The linear subspace *spanned* by a set of vectors V is denoted by $\mathrm{span}(V)$.

Given a transition function δ (which defines matrices $\mathrm{M}(w)$), call a set of words $W \subseteq \Sigma^*$ *complete for a subspace* $V \subseteq \mathbb{R}^n$ *with respect to a vector* $g \in V$, if $V \subseteq \mathrm{span}(\{g\mathrm{M}(w) \mid w \in W\})$. A set of words $W \subseteq \Sigma^*$ is *complete for a subspace* $V \subseteq \mathbb{R}^n$ if for every non-negative non-zero vector $g \in V$, W is complete for V with respect to g. Let $\chi(p)$ denote the characteristic (unitary) vector of $\{p\}$. For a subset $S \subseteq Q$, we define $\mathbb{V}(S) = \mathrm{span}(\{\chi(p) \mid p \in S\}) \subseteq \mathbb{R}^n$.

For example, consider the automaton from Fig. 1(left). Let $V = \mathbb{V}(\{q_1, q_2, q_5\})$ and let $W = \{ab, aab\}a^{\leq 5}$. Let $g \in V$ be a non-negative non-zero vector, and let i be such that $g(i) \neq 0$. If $i = 1$ then let $u = ab$, and otherwise let $u = aab$; then $g\mathrm{M}(u)$ has exactly one non-zero entry. Then, for each $j \in \{1,2,5\}$, the vector $g\mathrm{M}(ua^{j'})$ for some j' has the unique non-zero entry at q_j. These vectors generate V, thus W is complete for V with respect to g.

The induced automaton of a partial one is another partial automaton acting on a subset of states $R \subseteq Q$. It is built from two sets of words. Let W_1 be a set of words such that $R = \bigcup_{w \in W_1} \delta(Q,w)$. For each state in R, there is some state mapped to it by a word from W_1. The second set W_2 is any non-empty set of words that enriches its transitions. The induced automaton is $\mathscr{A}$ restricted to R with alphabet W_2W_1. Note that its transition function is well defined, which is ensured by the fact that every word of the form $w_2w_1 \in W_2W_1$ has the action mapping every state $q \in Q$ into R or to $\bot$.

▶ **Definition 18** (Induced automaton). *Let $W_1, W_2 \subseteq \Sigma^*$ be non-empty and $R = \{\delta(q,w) \mid q \in Q,\ w \in W_1, \delta(q,w) \neq \bot\}$. If R is non-empty, we define the* induced automaton *$\mathscr{A}^{\mathrm{I}(W_1,W_2)} = (R, W_2W_1, \delta_{\mathscr{A}^{\mathrm{I}(W_1,W_2)}})$, where the transition function is defined in compliance with the actions of words in $\mathscr{A}$, i.e., $\delta_{\mathscr{A}^{\mathrm{I}(W_1,W_2)}}(q,w) = \delta(q,w)$ for all $q \in R$ and $w \in W_2W_1$.*

We can analyze an induced automaton as a separate one, and synchronize the whole automaton using it, which is particularly profitable when R is small. Following our previous example, for Fig. 1(left) with $W_1 = \{b\}$ and $W_2 = \{ab, aab\}a^{\leq 5}$ we obtain the induced automaton on $R = \{q_1, q_2, q_5\}$. Furthermore, it is synchronizing already by a letter from W_2W_1 (e.g., abb), and each its reset word corresponds to a reset word of the original $\mathscr{A}$.

The following lemma states that the completeness of a set of words together with the synchronizability and strong connectivity of the whole automaton transfer to the induced automaton. It generalizes [4, Theorem 2] to partial automata, and the proof uses a recursion instead of an augmenting argument.

▶ **Lemma 19.** *Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a strongly connected synchronizing partial automaton and let W_1 and W_2 be two non-empty sets of words over Σ. Let $\mathscr{A}^{\mathrm{I}(W_1,W_2)} = (R, W_2W_1, \delta_{\mathscr{A}^{\mathrm{I}(W_1,W_2)}})$ be the induced automaton. If W_2 is complete for $\mathbb{V}(Q) = \mathbb{R}^n$, then W_2W_1 is complete for $\mathbb{V}(R)$, and $\mathscr{A}^{\mathrm{I}(W_1,W_2)}$ is synchronizing and strongly connected.*

▶ **Corollary 20.** *Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a strongly connected synchronizing partial automaton with n states, and let $w \in \Sigma^*$ be a word such that $R = \delta(Q,w) \neq \emptyset$. Let $W_1 = \{w\}$, $W_2 = \Sigma^{\leq n-1}$, and $\mathscr{A}^{\mathrm{I}(W_1,W_2)} = (R, \Sigma^{\leq n-1}\{w\}, \delta_{\mathscr{A}^{\mathrm{I}(W_1,W_2)}})$ be the induced automaton. Then* $\mathrm{rt}(\mathscr{A}) \leq |w| + (|w| + n - 1) \cdot \mathrm{rt}(\mathscr{A}^{\mathrm{I}(W_1,W_2)})$.

The corollary directly follows from Lemma 19, since $\Sigma^{\leq n-1}$ is always complete for $\mathbb{V}(Q)$ in the case of a strongly connected synchronizing partial automaton. It is useful for deriving upper bounds for automata with a word of small rank. Having such a word w, we can further synchronize R through the induced automaton instead of trying to do this directly. Although every letter of $\mathscr{A}^{\mathrm{I}(W_1,W_2)}$ corresponds to a word of length $|w|+n-1$ in the original $\mathscr{A}$, if R is small enough, this yields a better upper bound. We show its application in the next subsection.

3.8 The Literal Automaton of a Finite Prefix Code

We use the obtained results about induced automata to get better bounds for partial literal automata of finite prefix codes. To do so, we first need to prove that such automata admit short enough words of small rank. It is known that every complete literal automaton over an alphabet Σ has a word of rank and length at most $\lceil \log_{|\Sigma|} n \rceil$ ([4, Lemma 16], cf. [6, Lemma 14]). However, this is no longer true for non-mortal words in partial literal automata [29] and no similar statement was known for any wider class than complete literal automata. We prove that there exist $\mathcal{O}(\log n)$-rank non-mortal words of length $\mathcal{O}(n)$ in such automata, excluding the case of a code with only one word. Then we use this result to provide an $\mathcal{O}(n \log^3 n)$ upper bound on the reset threshold of n-state synchronizing partial literal automata, matching the known upper bound for complete literal automata [4].

We start with a special case of one-word codes. A non-empty word w is called *primitive* if it is not a power of another word, i.e., $w \neq u^k$ for every word u and $k \geq 2$. The upper bound on $\mathrm{rt}(\mathscr{A}_X)$ follows from a result of Weinbaum ([14, 39]).

▶ **Proposition 21.** *Let $X = \{x\}$ be a one-word prefix code, and suppose that $x = y^k$, where y is a non-empty primitive word and $k \geq 1$. Then $\mathscr{A}_X$ has rank k. If $\mathscr{A}_X$ is synchronizing, then $\mathrm{rt}(\mathscr{A}_X) \leq \frac{|x|}{2}$, and this bound is tight.*

In the remaining cases, there always exists a word of linear length and logarithmic rank.

▶ **Theorem 22.** *Let X be a prefix code with at least two words. Let $\mathscr{A}_X = (Q, \Sigma, \delta)$ be its partial literal automaton with n states and height h. Then there exists a word of length at most $2h$ and of rank at most $\lceil \log_2 hn \rceil + \lceil \log_2 h \rceil$ for $\mathscr{A}_X$. Moreover, such a word can be found in polynomial time in $n = |Q|$.*

Proof idea. The general idea is as follows. We construct a word from the theorem in two phases. First, we define an auxiliary *filtering* algorithm computing some function $\alpha \colon \Sigma^* \to \Sigma^*$. We consider the results of the algorithm for a lot of short (logarithmic) input words w and show that at least one of them satisfies that $\alpha(w)$ is non-mortal, has length at most h, and every state from Q is either sent to $\perp$ or goes through the root by its action. Then, we use specific properties of the image $\delta(Q, \alpha(w))$ to divide it into two disjoint sets: one that has up to h states, but on a single specific path, and the other one with a small (logarithmic) number of states. In the second phase, we construct a word v of length also bounded by h, such that its action map all the states from the mentioned specific path to a subset of at most logarithmic size. The concatenation of both words $\alpha(w)v$ is a word of length at most $2h$ satisfying the theorem. ◀

▶ **Corollary 23.** *Let $\mathscr{A}_X$ be a partial literal automaton with n states. If it is synchronizing, its reset threshold is at most $\mathcal{O}(n \log^3 n)$. If the Cerny conjecture holds, then it is $\mathcal{O}(n \log^2 n)$.*

Proof. If $|X| = 1$ then the bound follows from Proposition 21. If $|X| \geq 2$, from Theorem 22, we get a word w of length $\mathcal{O}(n)$ and rank $\mathcal{O}(\log n)$. Then we use Corollary 20 with w, which yields the upper bound $\mathcal{O}(n) + \mathcal{O}(n) \cdot \mathrm{rt}(\mathscr{B})$, where $\mathscr{B}$ is an induced automaton with $\mathcal{O}(\log n)$ states. Then we use upper bounds on the reset threshold of a complete DFA ([32, 33]) transferred to strongly connected partial DFAs by Theorem 17. ◀

4 Lower Bounds for Properly Incomplete Automata

We conclude with observations for transferring lower bounds from the complete case to the partial case. Of course, in general, this is trivial, since a complete automaton is a special case of a partial one. On the other hand, letters with all transitions undefined cannot be used for synchronization. Hence we need to add a restriction to exclude these cases and see the effect of usable incomplete transitions. A partial automaton is *properly incomplete* if there is at least one letter whose transition is defined for some state and is undefined for some other state.

For an automaton $\mathscr{A}$, let the length of the shortest words of rank r be called the *rank threshold* $\mathrm{rt}(\mathscr{A}, r)$. We show that bounding the rank/reset threshold of a strongly connected properly incomplete automaton is related to bounding the corresponding threshold of a complete automaton. A general construction for this is the following.

▶ **Definition 24** (Duplicating automaton). *For a complete automaton $\mathscr{A} = (Q, \Sigma, \delta_{\mathscr{A}})$, we construct the* duplicating automaton $\mathscr{A}^{\mathrm{D}} = (Q \cup Q', \Sigma \cup \{\gamma\}, \delta_{\mathscr{A}^{\mathrm{D}}})$ *as follows. Assume that $Q = \{q_1, \ldots, q_n\}$. Then $Q' = \{q'_1, \ldots, q'_n\}$ is a set of fresh states disjoint with Q and $\gamma \notin \Sigma$ is a fresh letter. For all $1 \le i \le n$ and $a \in \Sigma$, we define: $\delta_{\mathscr{A}^{\mathrm{D}}}(q_i, a) = q_i$, $\delta_{\mathscr{A}^{\mathrm{D}}}(q_i, \gamma) = q'_i$, $\delta_{\mathscr{A}^{\mathrm{D}}}(q'_i, a) = \delta_{\mathscr{A}}(q_i, a)$, and $\delta_{\mathscr{A}^{\mathrm{D}}}(q'_i, \gamma) = \bot$.*

The duplicating automaton turns out to be a partial DFA counterpart to the recent Volkov's construction of a complete DFA [37]. The duplicating automaton $\mathscr{A}^{\mathrm{D}}$ has twice the number of states of $\mathscr{A}$ and is properly incomplete. Also, it is strongly connected if $\mathscr{A}$ is.

▶ **Proposition 25.** *Let $\mathscr{A} = (Q, \Sigma, \delta_{\mathscr{A}})$ be a strongly connected complete automaton. For all $1 \le r < n$, we have $\mathrm{rt}(\mathscr{A}^{\mathrm{D}}, r) = 2\,\mathrm{rt}(\mathscr{A}, r)$.*

From Proposition 25, it follows that we cannot expect a better upper bound on the reset threshold of a properly incomplete strongly connected automaton than $0.04135n^3 + \mathcal{O}(n^2)$, unless we can improve the best general upper bound on the reset threshold of a complete automaton, which currently is roughly $0.1654n^3 + \mathcal{O}(n^2)$ [32, 33]. We can also show a lower bound on the largest possible reset threshold, using the Rystsov's construction of automata with long shortest mortal words [27].

▶ **Proposition 26.** *For every n, there exists a strongly connected properly incomplete n-state automaton with the reset threshold $\frac{n^2-n}{2}$.*

References

1 P. Babari, K. Quaas, and M. Shirmohammadi. Synchronizing data words for register automata. In *MFCS*, pages 15:1–15:15, 2016.

2 M.-P. Béal, E. Czeizler, J. Kari, and D. Perrin. Unambiguous automata. *Mathematics in Computer Science*, 1(4):625–638, 2008.

3 M. Berlinkov. On Two Algorithmic Problems about Synchronizing Automata. In *Developments in Language Theory*, LNCS, pages 61–67. Springer, 2014.

4 M. Berlinkov and M. Szykuła. Algebraic synchronization criterion and computing reset words. *Information Sciences*, 369:718–730, 2016.

5 J. Berstel, D. Perrin, and C. Reutenauer. *Codes and Automata*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2009.

6 M. T. Biskup and W. Plandowski. Shortest synchronizing strings for Huffman codes. *Theoretical Computer Science*, 410(38-40):3925–3941, 2009.

7 V. Bruyère. On maximal codes with bounded synchronization delay. *Theoretical Computer Science*, 204(1):11–28, 1998.

8 J. Černý. Poznámka k homogénnym eksperimentom s konečnými automatami. *Matematicko-fyzikálny Časopis Slovenskej Akadémie Vied*, 14(3):208–216, 1964. In Slovak.

9 D. Chistikov, P. Martyugin, and M. Shirmohammadi. Synchronizing automata over nested words. *Journal of Automata, Languages and Combinatorics*, 24(2-4):219–251, 2019.

10 L. Doyen, L. Juhl, K. G. Larsen, N. Markey, and M. Shirmohammadi. Synchronizing words for weighted and timed automata. In *FSTTCS*, pages 121–132, 2014.

11 L. Doyen, T. Massart, and M. Shirmohammadi. Robust Synchronization in Markov Decision Processes. In *CONCUR*, pages 234–248, 2014.

12 L. Doyen, T. Massart, and M. Shirmohammadi. The complexity of synchronizing Markov decision processes. *J. Comput. Syst. Sci.*, 100:96–129, 2019.

13 D. Eppstein. Reset sequences for monotonic automata. *SIAM Journal on Computing*, 19:500–510, 1990.

14 T. Harju and D. Nowotka. On unique factorizations of primitive words. *Theor. Comput. Sci.*, 356(1-2):186–189, 2006.

15 J. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In Zvi Kohavi and Azaria Paz, editors, *Theory of Machines and Computations*, pages 189–196. Academic Press, 1971.

16 D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

17 B. Imreh and M. Steinby. Directable nondeterministic automata. *Acta Cybern.*, 14(1):105–115, 1999.

18 J. Kari. A counter example to a conjecture concerning synchronizing word in finite. *EATCS Bulletin*, 73:146–147, 2001.

19 J. Kari. Synchronizing finite automata on Eulerian digraphs. *Theoretical Computer Science*, 295(1-3):223–232, 2003.

20 J. Kari, A. Ryzhikov, and A. Varonka. Words of minimum rank in deterministic finite automata. In *Developments in Language Theory - 23rd International Conference, DLT 2019, Warsaw, Poland, August 5-9, 2019, Proceedings*, pages 74–87, 2019.

21 S. Kiefer and C. Mascle. On Finite Monoids over Nonnegative Integer Matrices and Short Killing Words. In *STACS*, LIPIcs, 2019.

22 K. G. Larsen, S. Laursen, and J. Srba. Synchronizing strategies under partial observability. In *International Conference on Concurrency Theory*, pages 188–202. Springer, 2014.

23 B. K. Natarajan. An algorithmic approach to the automated design of parts orienters. In *Foundations of Computer Science, 27th Annual Symposium on*, pages 132–142, 1986.

24 J.-E. Pin. Utilisation de l'algèbre linéaire en théorie des automates. In *Actes du 1er Colloque AFCET-SMF de Mathématiques Appliquées II*, AFCET, pages 85–92, 1978. In French.

25 J.-E. Pin. On two combinatorial problems arising from automata theory. In *Proceedings of the International Colloquium on Graph Theory and Combinatorics*, volume 75 of *North-Holland Mathematics Studies*, pages 535–548, 1983.

26 I. K. Rystsov. Polynomial complete problems in automata theory. *Information Processing Letters*, 16(3):147–151, 1983.

27 I. K. Rystsov. Reset words for commutative and solvable automata. *Theoretical Computer Science*, 172(1-2):273–279, 1997.

28 A. Ryzhikov. Mortality and synchronization of unambiguous finite automata. In *Combinatorics on Words - 12th International Conference, WORDS 2019, Loughborough, UK, September 9-13, 2019, Proceedings*, pages 299–311, 2019.

29 A. Ryzhikov and M. Szykuła. Finding Short Synchronizing Words for Prefix Codes. In Igor Potapov, Paul Spirakis, and James Worrell, editors, *MFCS 2018*, volume 117 of *LIPIcs*, pages 21:1–21:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

30 S. Sandberg. Homing and synchronizing sequences. In *Model-Based Testing of Reactive Systems*, volume 3472 of *LNCS*, pages 5–33. Springer, 2005.

31 H. Shabana. Exact synchronization in partial deterministic automata. *Journal of Physics: Conference Series*, 1352:012047, 2019.

32 Y. Shitov. An Improvement to a Recent Upper Bound for Synchronizing Words of Finite Automata. *Journal of Automata, Languages and Combinatorics*, 24(2–4):367–373, 2019.

33 M. Szykuła. Improving the Upper Bound on the Length of the Shortest Reset Word. In *STACS 2018*, LIPIcs, pages 56:1–56:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

34 N. F. Travers and J. P. Crutchfield. Exact Synchronization for Finite-State Sources. *Journal of Statistical Physics*, 145(5):1181–1201, 2011.

35 L. Trevisan. Notes on state minimization. `https://people.eecs.berkeley.edu/~luca/cs172-07/notemindfa.pdf`, 2007.

36 M. V. Volkov. Synchronizing automata and the Černý conjecture. In *Language and Automata Theory and Applications*, volume 5196 of *LNCS*, pages 11–27. Springer, 2008.

37 M. V. Volkov. Slowly synchronizing automata with idempotent letters of low rank. *Journal of Automata, Languages and Combinatorics*, 24(2–4):375–386, 2019.

38 V. Vorel. Subset synchronization and careful synchronization of binary finite automata. *Int. J. Found. Comput. Sci.*, 27(5):557–578, 2016.

39 C. M. Weinbaum. Unique subwords in nonperiodic words. *Proceedings of the American Mathematical Society*, 109(3):615–619, 1990.

On Euclidean Steiner $(1+\varepsilon)$-Spanners

Sujoy Bhore
Université Libre de Bruxelles, Brussels, Belgium

Csaba D. Tóth
California State University Northridge, Los Angeles, CA, USA
Tufts University, Medford, MA, USA

Abstract

Lightness and sparsity are two natural parameters for Euclidean $(1+\varepsilon)$-spanners. Classical results show that, when the dimension $d \in \mathbb{N}$ and $\varepsilon > 0$ are constant, every set S of n points in d-space admits an $(1+\varepsilon)$-spanners with $O(n)$ edges and weight proportional to that of the Euclidean MST of S. Tight bounds on the dependence on $\varepsilon > 0$ for constant $d \in \mathbb{N}$ have been established only recently. Le and Solomon (FOCS 2019) showed that Steiner points can substantially improve the lightness and sparsity of a $(1+\varepsilon)$-spanner. They gave upper bounds of $\tilde{O}(\varepsilon^{-(d+1)/2})$ for the minimum lightness in dimensions $d \geq 3$, and $\tilde{O}(\varepsilon^{-(d-1))/2})$ for the minimum sparsity in d-space for all $d \geq 1$. They obtained lower bounds only in the plane ($d = 2$). Le and Solomon (ESA 2020) also constructed Steiner $(1+\varepsilon)$-spanners of lightness $O(\varepsilon^{-1}\log\Delta)$ in the plane, where $\Delta \in \Omega(\log n)$ is the *spread* of S, defined as the ratio between the maximum and minimum distance between a pair of points.

In this work, we improve several bounds on the lightness and sparsity of Euclidean Steiner $(1+\varepsilon)$-spanners. Using a new geometric analysis, we establish lower bounds of $\Omega(\varepsilon^{-d/2})$ for the lightness and $\Omega(\varepsilon^{-(d-1)/2})$ for the sparsity of such spanners in Euclidean d-space for all $d \geq 2$. We use the geometric insight from our lower bound analysis to construct Steiner $(1+\varepsilon)$-spanners of lightness $O(\varepsilon^{-1}\log n)$ for n points in Euclidean plane.

2012 ACM Subject Classification Mathematics of computing → Approximation algorithms; Mathematics of computing → Paths and connectivity problems; Theory of computation → Computational geometry

Keywords and phrases Geometric spanner, $(1+\varepsilon)$-spanner, lightness, sparsity, minimum weight

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.13

Related Version *Previous Version*: `https://arxiv.org/abs/2010.02908`

Funding *Sujoy Bhore*: Research on this paper was supported by the Fonds de la Recherche Scientifique-FNRS under Grant no MISU F 6001.
Csaba D. Tóth: Research on this paper was partially supported by the NSF award DMS-1800734.

1 Introduction

For an edge-weighted graph G, a subgraph H of G is a *t-spanner* if $\delta_H(u,v) \leq t \cdot \delta_G(u,v)$, where $\delta_G(u,v)$ denotes the shortest path distance between vertices u and v. A subgraph H of G is a *t-spanner*, for some $t \geq 1$, if for every $pq \in \binom{V(G)}{2}$, we have $d_G(p,q) \leq t \cdot w(pq)$. The parameter t is called the *stretch factor* of the spanner. Spanners are fundamental graph structures with many applications in the area of distributed systems and communication, distributed queuing protocol, compact routing schemes, etc.; see [16, 24, 34, 35]. Two important parameters of a spanner H are *lightness* and *sparsity*. The *lightness* of H is the ratio $w(H)/w(\mathrm{MST})$ between the total weight of H and the weight of a minimum spanning tree (MST). The *sparsity* of H is the ratio $|E(H)|/|E(\mathrm{MST})| \approx |E(H)|/|V(G)|$ between the number of edges of H and an MST. As H is connected, the trivial lower bound for both the lightness and the sparsity of a spanner is 1. When the vertices of G are points in a metric

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 13; pp. 13:1–13:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

space, the edge weights obey the triangle inequality. The most important examples include Euclidean d-space and, in general, metric spaces with constant doubling dimensions (the doubling dimension of $\mathbb{R}^d$ is d).

In a *geometric spanner*, the underlying graph $G = (S, \binom{S}{2})$ is the complete graph on a finite point set S in $\mathbb{R}^d$, and the edge weights are the Euclidean distances between vertices. Euclidean spanners are one of the fundamental geometric structures that find application across domains, such as, topology control in wireless networks [38], efficient regression in metric spaces [21], approximate distance oracles [23], and others. Rao and Smith [36] showed the relevance of Euclidean spanners in the context of other geometric NP-hard problems, e.g., Euclidean traveling salesman problem and Euclidean minimum Steiner tree problem, and introduced the so called *banyans*[1], which is a generalization of graph spanners. Apart from lightness and sparsity, various other optimization criteria have been considered, e.g., bounded-degree spanners [6] and α-diamond spanners [13]. Several distinct construction approaches have been developed for Euclidean spanners, that each found further applications in geometric optimization, such as well-separated pair decomposition (WSPD) based spanners [7, 22], skip-list spanners [2], path-greedy and gap-greedy spanners [1, 3], and more. For an excellent survey of results and techniques on Euclidean spanners up to 2007, we refer to the book by Narasimhan and Smid [33].

Sparsity. A large body of research on spanners has been devoted to *sparse spanners* where the objective is to obtain a spanner with small number edges, preferably $O(|S|)$, with $1+\varepsilon$ stretch factor, for any given $\varepsilon > 0$. Chew [9] was the first to show that there exists a Euclidean spanner with a linear number of edges and *stretch factor* $\sqrt{10}$. The stretch factor was later improved to 2 [10]. Clarkson [11] designed the first Euclidean $(1+\varepsilon)$-spanner, for arbitrary small $\varepsilon > 0$; an alternative algorithm was presented by Keil [25]. Later, Keil and Gutwin [26] showed that the Delaunay triangulation of the point set S is a 2.42-spanner. Moreover, these papers introduced the fixed-angle Θ-graph[2] as a potential new tool for designing spanners in $\mathbb{R}^2$, which was later generalized to higher dimension by Ruppert and Seidel [37]. One can construct an $(1+\varepsilon)$-spanner with $O(n\varepsilon^{-d+1})$ edges by taking the angle Θ to be proportional to ε in any constant dimension $d \geq 1$. A fundamental question in this area is whether the trade-off between the stretch factor $1+\varepsilon$ and the sparsity $O(n\varepsilon^{-d+1})$ is tight.

Lightness. For a set of points S in a metric space, the lightness is the ratio of the spanner weight (i.e., the sum of all edge weights) to the weight of the minimum spanning tree $\mathrm{MST}(S)$. Das et al. [12] showed that *greedy-spanner* ([1]) has constant lightness in $\mathbb{R}^3$. This was generalized later to $\mathbb{R}^d$, for all $d \in \mathbb{N}$, by Das et al. [14]. However the dependencies on ε and d has not been addressed. Rao and Smith showed that the greedy spanner has lightness $\varepsilon^{-O(d)}$ in $\mathbb{R}^d$ for every constant d, and asked what is the best possible constant in the exponent. A complete proof for $(1+\varepsilon)$-spanner with lightness $O(\varepsilon^{-2d})$ is in the book on geometric spanners [33]. Recently, Borradaile et al. [5] showed that the greedy $(1+\varepsilon)$-spanner of a finite metric space of doubling dimension d has lightness $\varepsilon^{-O(d)}$.

[1] A $(1+\varepsilon)$-banyan for a set of points A is a set of points A' and line segments S with endpoints in $A \cup A'$ such that a $1+\varepsilon$ optimal Steiner Minimum Tree for any subset of A is contained in S

[2] The Θ-graph is a type of geometric spanner similar to Yao graph [42], where the space around each point $p \in P$ is partitioned into cones of angle Θ, and S will be connected to a point $q \in P$ whose orthogonal projection to some fixed ray contained in the cone is closest to S.

Dependence on $\varepsilon > 0$ for constant dimension d. The dependence of the lightness and sparsity on $\varepsilon > 0$ for constant $d \in \mathbb{N}$ has been studied only recently. Le and Solomon [27] constructed, for every $\varepsilon > 0$ and constant $d \in \mathbb{N}$, a set S of n points in $\mathbb{R}^d$ for which any $(1+\varepsilon)$-spanner must have lightness $\Omega(\varepsilon^{-d})$ and sparsity $\Omega(\varepsilon^{-d+1})$, whenever $\varepsilon = \Omega(n^{-1/(d-1)})$. Moreover, they showed that the greedy $(1+\varepsilon)$-spanner in $\mathbb{R}^d$ has lightness $O(\varepsilon^{-d}\log\varepsilon^{-1})$.

Steiner points are additional vertices in a network (via points) that are not part of the input, and a t-spanner must achieve stretch factor t only between pairs of the input points in S. A classical problem on Steiner points arises in the context of minimum spanning trees. The *Steiner ratio* is the supremum ratio between the weight of a *minimum Steiner tree* and a *minimum spanning tree* of a finite point set, and it is at least $\frac{1}{2}$ in any metric space due to triangle inequality.

Le and Solomon [27] noticed that Steiner points can substantially improve the bound on the lightness and sparsity of an $(1+\varepsilon)$-spanner. Previously, Elkin and Solomon [19] and Solomon [39] showed that Steiner points can improve the weight of the network in the single-source setting. In particular, the so-called *shallow-light trees* (SLT) is a single-source spanning tree that concurrently approximates a shortest-path tree (between the source and all other points) and a minimum spanning tree (for the total weight). They proved that Steiner points help to obtain exponential improvement on the lightness SLTs in a general metric space [19], and quadratic improvement on the lightness in Euclidean spaces [39].

Le and Solomon, used Steiner points to improve the bounds for lightness and sparsity of Euclidean spanners. For minimum sparsity, they gave an upper bound of $O(\varepsilon^{(1-d)/2})$ for d-space and a lower bound of $\Omega(\varepsilon^{-1/2}/\log\varepsilon^{-1})$ in the plane ($d = 2$) [27]. For minimum lightness, Le and Solomon [28] gave an upper bound of $O(\varepsilon^{-1}\log\Delta)$ in the plane and $O(\varepsilon^{-(d+1)/2}\log\Delta)$ in dimension $d \geq 3$, where Δ is the *spread* of the point set, defined as the ratio between the maximum and minimum distance between a pair of points. Note that in any space with doubling dimension d (including $\mathbb{R}^d$), we have $\Delta \geq \Omega(\log_d n)$, but the spread Δ is in fact unbounded. Very recently, Le and Solomon [30] constructed Steiner $(1+\varepsilon)$-spanners with lightness $\tilde{O}(\varepsilon^{-(d+1)/2})$ in dimensions $d \geq 3$.

Our Contributions. In this work, we improve the bounds on the lightness and sparsity of Euclidean Steiner $(1+\epsilon)$-spanners. First, in Section 3, we prove the following lower bounds.

▶ **Theorem 1.** *Let a positive integers d and real $\varepsilon > 0$ be given such that $\varepsilon \leq 1/d$. Then there exists a set S of n points in $\mathbb{R}^d$ such that any Euclidean Steiner $(1+\varepsilon)$-spanner for S has lightness $\Omega(\varepsilon^{-d/2})$ and sparsity $\Omega(\varepsilon^{(1-d)/2})$.*

For lightness in dimension $d = 2$, this improves the earlier bound of $\Omega(\varepsilon^{-1}\log^{-1}(\varepsilon^{-1}))$ by Le and Solomon [27] by a logarithmic factor; and it is the first lower bound in dimensions $d \geq 3$. The point set S in Theorem 1 is fairly simple, it consists of two square grids in two parallel hyperplanes in $\mathbb{R}^d$. However, our lower-bound analysis is significantly simpler than that of [27]. In particular, our analysis does not depend on planarity, and it generalizes to higher dimensions. The key new insight pertains to a geometric property of Steiner $(1+\varepsilon)$-spanners: If the length of an ab-path S between points $a, b \in \mathbb{R}^d$ is at most $(1+\varepsilon)\|ab\|$, then "most" of the edges of S are almost parallel to ab. We expand on this idea in Section 2.

Then, in Section 4 we prove the following theorem on light spanners.

▶ **Theorem 2.** *For every set S of n points in Euclidean plane, there exists a Steiner $(1+\varepsilon)$-spanner of lightness $O(\varepsilon^{-1}\log n)$.*

This result improves on an earlier bound of $O(\varepsilon^{-1}\log\Delta)$ by Le and Solomon [28], where Δ is the *spread* of the point set, defined as the ratio between the maximum and minimum distance between a pair of points. Note that $\Delta \geq \Omega(\log n)$ in every metric space of constant doubling dimension. Recently, Le and Solomon [29] noted in the revised version of their paper that the $\log \Delta$ factor can be reduced to a $\log n$ factor by a general discretization technique (see, e.g., Chan et al. [8]). Very recently, Bhore and Tóth [4] achieved the optimal dependence on ε and showed that, for every finite points set $S \subset \mathbb{R}^2$ and $\varepsilon > 0$, there exists a Euclidean Steiner $(1+\varepsilon)$-spanner of weight $O(\frac{1}{\varepsilon}\,\|\mathrm{MST}(S)\|)$. The spanner construction in [4] is a far-reaching generalization of the methods we develop in the proof of Theorem 2. In particular, both papers use *directional spanners*, introduced in Section 4 of this paper, as a key ingredient and construct a Euclidean Steiner $(1+\varepsilon)$-spanner as a union of $O(\varepsilon^{-1/2})$ directional spanners.

2 Preliminaries

Let $d \geq 2$ be an integer, and S a set of n points in $\mathbb{R}^d$. For $a, b \in \mathbb{R}^d$, the Euclidean distance between a and b is denoted by $\|ab\|$. For a set E of line segments in $\mathbb{R}^d$, let $\|E\| = \sum_{e\in E} \|e\|$ be the total weight of all segments in E. For a geometric graph $G = (S, E)$, where $S \subset \mathbb{R}^d$, we also use the notation $\|G\| = \|E\|$, which is the Euclidean weight of graph G.

We briefly review a few geometric primitives in d-space. For $a, b \in \mathbb{R}^d$, the locus of points $c \in \mathbb{R}^d$ with $\|ac\| + \|cb\| \leq (1+\varepsilon)\|ab\|$ is an ellipsoid $\mathcal{E}_{ab}$ with foci a and b, and major axis of length $(1+\varepsilon)\|ab\|$; see Fig. 1(a). Note that all $d-1$ minor axes of $\mathcal{E}_{ab}$ are $\sqrt{(1+\varepsilon)^2 - 1}\|ab\| = \sqrt{2\varepsilon + \varepsilon^2}\|ab\| < \sqrt{3\varepsilon}\|ab\|$ when $\varepsilon < 1$. In particular, the aspect ratio of the minimum bounding box of $\mathcal{E}_{ab}$ is roughly $\sqrt{\varepsilon}$. By the triangle inequality, $\mathcal{E}_{ab}$ contains every ab-path of weight at most $(1+\varepsilon)\|ab\|$.

The unit vectors in $\mathbb{R}^d$ are on the $(d-1)$-sphere $\mathbb{S}^{d-1}$; the direction vectors of a line in $\mathbb{R}^d$ can be represented by vectors of a hemisphere. The *angle* between two unit vectors, $\overrightarrow{u}_1$ and $\overrightarrow{u}_2$ is $\angle(\overrightarrow{u}_1, \overrightarrow{u}_2) = \arccos(\overrightarrow{u}_1 \cdot \overrightarrow{u}_2) \in (-\pi, \pi)$. Between two (undirected) edges e_1 and e_2 with unit direction vectors $\pm\overrightarrow{u}_1$ and $\pm\overrightarrow{u}_2$, we define the angle as $\angle(e_1, e_2) = \arccos|\overrightarrow{u}_1 \cdot \overrightarrow{u}_2| \in [0, \pi)$. Let $\mathrm{proj}_{ab}(e)$ denote the orthogonal projection of an edge e to the supporting line of ab, see Fig. 1(b); and note that $\|\mathrm{proj}_{ab}(e)\| = \|e\| \cos(ab, e)$.

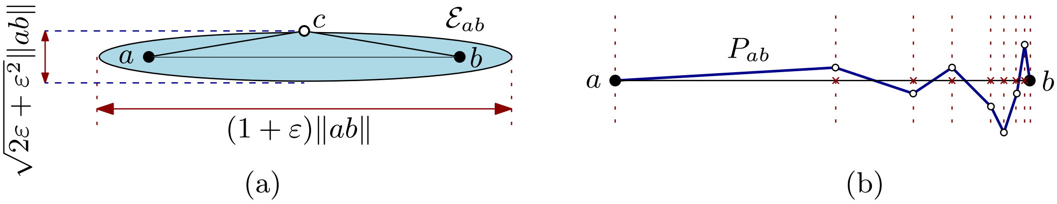


Figure 1 (a) An ellipse $\mathcal{E}_{ab}$ with foci a and b, and major axis $(1+\varepsilon)\|ab\|$. (b) A monotone ab-path P_{ab}, and the projections of its edges to ab.

Characterization for Short ab-Paths. Let $a, b \in \mathbb{R}^d$, and let P_{ab} be a polygonal ab-path of weight at most $(1+\varepsilon)\|ab\|$. We show that "most" edges along P_{ab} must be "nearly" parallel to ab. Specifically, for an angle $\alpha \in [0, \pi/2)$, we distinguish between two types of edges in P_{ab}. Denote by $E(\alpha)$ the set of edges e in P_{ab} with $\angle(ab, e) \leq \alpha$; and let $F(\alpha)$ be the set of all other edges of P_{ab}. Clearly, we have $\|P_{ab}\| = \|E(\alpha)\| + \|F(\alpha)\|$ for all α.

▶ **Lemma 3.** *Let $a, b \in \mathbb{R}^d$ and let P_{ab} be an ab-path of weight $\|P_{ab}\| \leq (1+\varepsilon)\|ab\|$. Then for every $i \in \{1, \ldots, \lfloor 1/\sqrt{\varepsilon} \rfloor\}$, we have $\|E(i \cdot \sqrt{\varepsilon})\| \geq (1 - 2/i^2)\,\|ab\|$.*

Proof. Suppose, to the contrary, that $\|E(i\cdot\sqrt{\varepsilon})\| < (1-2/i^2)\,\|ab\|$ for an $i \in \{1,\dots,\lfloor 1/\sqrt{\varepsilon}\rfloor\}$. We have

$$\sum_{e\in E(i\sqrt{\varepsilon})\cup F(i\sqrt{\varepsilon})} \|\mathrm{proj}_{ab}(e)\| \geq \|ab\|, \tag{1}$$

which implies

$$\begin{aligned}\sum_{e\in F(i\sqrt{\varepsilon})} \|\mathrm{proj}_{ab}(e)\| &\geq \|ab\| - \sum_{e\in E(i\sqrt{\varepsilon})} \|\mathrm{proj}_{ab}(e)\| \\ &\geq \|ab\| - \sum_{e\in E(i\sqrt{\varepsilon})} \|e\| \\ &= \|ab\| - \|E(i\sqrt{\varepsilon})\|.\end{aligned} \tag{2}$$

Recall that for every edge $e \in F(i\sqrt{\varepsilon})$, we have $\angle(e, ab) \geq i\cdot\sqrt{\varepsilon}$. Using the Taylor estimate $\frac{1}{\cos(x)} \geq 1 + \frac{x^2}{2}$, for every $e \in F(i\sqrt{\varepsilon})$, we obtain

$$|e\| \geq \frac{\|\mathrm{proj}_{ab}(e)\|}{\cos(i\cdot\sqrt{\varepsilon})} \geq \|\mathrm{proj}_{ab}(e)\| \left(1 + \frac{(i\sqrt{\varepsilon})^2}{2}\right) = \|\mathrm{proj}_{ab}(e)\| \left(1 + \frac{i^2\,\varepsilon}{2}\right),$$

Combined with (2), this yields

$$\begin{aligned}\|P_{ab}\| &= \sum_{e\in E(i\sqrt{\varepsilon})} \|e\| + \sum_{e\in F(i\sqrt{\varepsilon})} \|e\| \\ &\geq \sum_{e\in E(i\sqrt{\varepsilon})} \|e\| + \sum_{e\in F(i\sqrt{\varepsilon})} \|\mathrm{proj}_{ab}(e)\| \left(1 + \frac{i^2\,\varepsilon}{2}\right) \\ &\geq \|E(i\sqrt{\varepsilon})\| + \big(\|ab\| - \|E(i\sqrt{\varepsilon})\|\big)\left(1 + \frac{i^2\,\varepsilon}{2}\right) \\ &= \left(1 + \frac{i^2\,\varepsilon}{2}\right)\|ab\| - \frac{i^2\,\varepsilon}{2}\,\|E(i\sqrt{\varepsilon})\| \\ &> \left(1 + \frac{i^2\,\varepsilon}{2}\right)\|ab\| - \frac{i^2\,\varepsilon}{2}\left(1 - \frac{2}{i^2}\right)\|ab\| \\ &\geq \left(1 + \frac{i^2\,\varepsilon}{2}\right)\|ab\| - \left(\frac{i^2}{2} - 1\right)\varepsilon\,\|ab\| \\ &= (1+\varepsilon)\|ab\|,\end{aligned}$$

which is a contradiction. ◀

We use Lemma 3 in the analysis of our lower bound construction in Section 1. We can also derive a converse of Lemma 3 for monotone ab-paths. An ab-path is *monotone* if $\angle(\overrightarrow{ab}, \overrightarrow{e}) > 0$ for every directed edge $\overrightarrow{e}$ of P_{ab}, where the path is directed from a to b. Equivalently, an ab-path is monotone if it crosses every hyperplane orthogonal to ab at most once. We show that if the angle $\angle(\overrightarrow{ab}, \overrightarrow{d})$ is sufficiently small for "most" of the edges of P_{ab}, then $\|P_{ab}\| \leq (1+\varepsilon)\|ab\|$.

▶ **Lemma 4.** *For every $\delta > 0$, there is a $\kappa > 0$ with the following property. For $a, b \in \mathbb{R}^d$ and a monotone an ab-path P_{ab}, if $\|F(i\sqrt{\varepsilon\kappa})\| \leq \|P_{ab}\|/i^{2+\delta}$ for all $i \in \{1,\dots,\lceil \pi/\sqrt{\varepsilon\kappa}\rceil\}$, then $\|P_{ab}\| \leq (1+\varepsilon)\|ab\|$.*

Proof. Let P_{ab} be an ab-path with edge set E. Note that, by definition, $F(0) = E$. For angles $0 \le \alpha < \beta \le \pi/2$, let $E(\alpha,\beta)$ denote the set of edges $e \in E$ with $\alpha \le \angle(ab,e) < \beta$. For convenience, we put $m = \lceil \pi/\sqrt{\varepsilon\kappa}\rceil$. Using the Taylor estimate $\cos x \ge 1 - x^2/2$, we can bound the excess weight of P_{ab} as follows.

$$\begin{aligned}
\|P_{ab}\| - \|ab\| &= \sum_{e\in E} \|e\| - \sum_{e\in E} \|\mathrm{proj}_{ab} e\| \\
&= \sum_{e\in E} \|e\|(1-\cos\angle(ab,e)) \\
&\le \sum_{i=1}^{m} \|E((i-1)\sqrt{\varepsilon\kappa}, i\sqrt{\varepsilon\kappa})\|(1-\cos(i\sqrt{\varepsilon\kappa})) \\
&\le \sum_{i=1}^{m} \|E((i-1)\sqrt{\varepsilon\kappa}, i\sqrt{\varepsilon\kappa})\| \cdot \frac{i^2\,\varepsilon\kappa}{2} \\
&\le \sum_{i=1}^{m} \big(\|F((i-1)\sqrt{\varepsilon\kappa})\| - \|F(i\sqrt{\varepsilon\kappa})\|\big) \cdot \frac{i^2\,\varepsilon\kappa}{2} \\
&= F(0)\cdot\frac{1^2\varepsilon\kappa}{2} + \sum_{i=1}^{m} \|F(i\sqrt{\varepsilon\kappa})\| \left(\frac{(i+1)^2\,\varepsilon\kappa}{2} - \frac{i^2\,\varepsilon\kappa}{2}\right) \\
&\le \|P_{ab}\|\cdot\frac{\varepsilon\kappa}{2} + \sum_{i=1}^{m} \frac{\|P_{ab}\|}{i^{2+\delta}}\cdot\frac{(2i+1)\varepsilon\kappa}{2} \\
&\le \frac{\varepsilon\kappa}{2}\cdot\|P_{ab}\|\left(1+\sum_{i=1}^{\infty}\frac{2i+1}{i^{2+\delta}}\right)
\end{aligned}$$

For $\kappa = 2(1+\sum_{i=1}^{\infty}(2i+1)/2^{2+\delta})^{-1}$, we obtain

$$\|P_{ab}\| - \|ab\| \le \frac{\varepsilon}{2}\,\|P_{ab}\|,$$

which readily implies $\|P_{ab}\| \le (1-\varepsilon/2)^{-1}\|ab\| < (1+\varepsilon)\|ab\|$, as required. ◀

The criteria in Lemma 4 can certify that a geometric graph G is a Euclidean Steiner $(1+\varepsilon)$-spanner for a point set S. Intuitively, an $(1+\varepsilon)$-spanner should contain, for all point pairs $a, b \in S$, an ab-path in which the majority of edges e satisfy $\angle(ab,e) \le O(\sqrt{\varepsilon})$, with exceptions quantified by Lemma 4. This property has already been used by Solomon [39] in the single-source setting, for the design of shallow-light trees. We use shallow-light trees in our upper bound (Section 4), instead of Lemma 4. However, the characterization of ab-paths of weight at most $(1+\varepsilon)\|ab\|$, presented in this section, may be of independent interest.

Shallow-light trees. Shallow-light trees (SLT) were introduced by Solomon [39]. Given a source s and a point set S in $\mathbb{R}^d$, an (α,β)-*SLT* is a Steiner tree rooted at s that contains a path of weight at most $\alpha\,\|ab\|$ between the *source* s and any point $t \in S$, and has weight $\beta\,\|\mathrm{MST}(S)\|$. For our upper bounds in Section 4, we use the following variant of shallow-light trees, between a source s and a set S of collinear points in the plane; see Figure 2.

▶ **Lemma 5** (Solomon [39, Section 2.1]). *Let $0 < \varepsilon < 1$, let S be a set of points in the $[-\frac{1}{2},\frac{1}{2}]$ interval of the x-axis, and let $s = (0,\varepsilon^{-1/2})$ be a point on the y-axis. Then there exists a geometric graph of weight $O(\varepsilon^{-1/2})$ that contains, for every point $t \in S$, an st-path P_{st} with $\|P_{st}\| \le (1+\varepsilon)\,\|st\|$.*

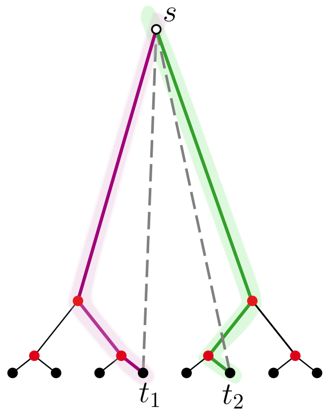


Figure 2 An illustration of a shallow-light tree for a source s and a set S of collinear points. The input points and the Steiner points are colored black and red, respectively.

3 Lower Bounds

In this section we prove the following lower bound on the lightness of Euclidean Steiner spanners in $\mathbb{R}^d$. Our lower bound construction is a direct generalization of the 2-dimensional lower bound construction by Le and Solomon [27]. However, our analysis is significantly simpler than that of [27] and it does not depend on planarity. As a result, it easily extends to higher dimensions.

▶ **Theorem 1.** *Let a positive integers d and real $\varepsilon > 0$ be given such that $\varepsilon \leq 1/d$. Then there exists a set S of n points in $\mathbb{R}^d$ such that any Euclidean Steiner $(1+\varepsilon)$-spanner for S has lightness $\Omega(\varepsilon^{-d/2})$ and sparsity $\Omega(\varepsilon^{(1-d)/2})$.*

Proof. First we establish the result for a point set of size $\Theta_d(\varepsilon^{(1-d)/2})$ and then generalize to arbitrary n. Let $Q = [0,1]^d$ be a unit cube in $\mathbb{R}^d$; see Fig. 3. The point set S will consist of two square grids in two opposite faces of Q, with $d/\sqrt{\varepsilon}$ spacing. Specifically, consider the lattice $L = (d/\sqrt{\varepsilon}) \cdot \mathbb{Z}^d$. Let Q_0 and Q_1, respectively, be the two faces of Q orthogonal to the x_d-axis. Now let $S_0 = L \cap Q_0$ and $S_1 = L \cap Q_1$. We have $|S_0| = |S_1| = \lfloor 1/(d\sqrt{\varepsilon}) \rfloor^{d-1} = \Theta_d(\varepsilon^{(1-d)/2})$, hence $|S| = \Theta_d(\varepsilon^{(1-d)/2})$.

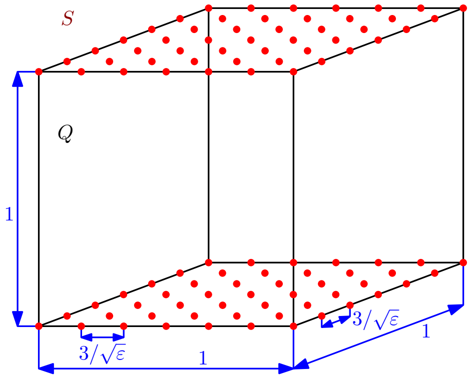


Figure 3 A schematic image of S in $\mathbb{R}^3$.

Let N be a Euclidean Steiner $(1+\varepsilon)$-spanner for S. For a point pair $(a,b) \in S_0 \times S_1$, we have $1 \leq \|ab\| \leq \operatorname{diam}(Q) = \sqrt{d}$. The spanner N contains an ab-path P_{ab} of weight at most $(1+\varepsilon)\|ab\|$, which lies in the ellipsoid $\mathcal{E}_{ab}$ with foci at a and b, and major axis $(1+\varepsilon)\|ab\|$.

The ellipsoid $\mathcal{E}_{ab}$ is, in turn, contained in an infinite cylinder C_{ab} with axis ab and radius $\frac{\sqrt{(1+\varepsilon)^2-1^2}}{2}\|ab\| < \sqrt{\varepsilon}\|ab\| \leq \sqrt{\varepsilon d}$. The intersection of the cylinder C_{ab} with hyperplanes containing Q_0 and Q_1, resp., is an ellipsoid of half-diameter at most $\sqrt{d-1}\cdot\sqrt{\varepsilon d} < \sqrt{\varepsilon}\, d$, and their centers are a and b, respectively. In particular, all point in S, other than a and b, are in the exterior of C_{ab}.

We distinguish between two types of edges in the ab-path P_{ab}. An edge e of P_{ab} is *near-parallel* to ab if $\angle(ab,e) < 2\cdot\sqrt{\varepsilon}$. Let $E(ab)$ be the set of edges of P_{ab} that are near-parallel to ab, and $F(ab)$ the set of all other edges of P_{ab}. Lemma 3 with $i=2$ yields

$$\|E(ab)\| \geq \frac{1}{2}\|ab\| \geq \frac{1}{2}. \tag{3}$$

Notice that for two pairs $(a_1,b_1),(a_2,b_2)\in S_0\times S_1$, if $\{a_1,b_1\}\neq\{a_2,b_2\}$, then $E(a_1b_1)\cap E(a_2b_2)=\emptyset$. If $\angle(a_1b_1,a_2b_2)\geq 4\sqrt{\varepsilon}$, this follows from the fact that the *directions* near-parallel to a_1b_1 and a_2b_2, resp., are disjoint. If $\angle(a_1b_1,a_2b_2) < 4\sqrt{\varepsilon}$, then a_1b_1 and a_2b_2 are parallel, consequently the cylinders $C_{a_1b_1}$ and $C_{a_2b_2}$ have disjoint interiors, and so $E(a_1b_1)\cap E(a_2b_2)=\emptyset$. Combined with (3), this yields

$$\|N\| \geq \sum_{(a,b)\in S_0\times S_1} \|E(ab)\| \geq |S_0|\cdot|S_1|\cdot\frac{1}{2} \geq \Theta_d(\varepsilon^{1-d}). \tag{4}$$

Similarly to [27, Claim 5.3], we may assume that $N\subseteq Q$ (indeed, we can replace every vertex of N outside of Q by the closest point in the boundary of Q; such replacements do not increase the weight of N). In follows that the weight of every edge is at most $\mathrm{diam}(Q)=\sqrt{d}$. Consequently,

$$|E(N)| \geq \frac{\|N\|}{\max_{e\in E(N)}\|e\|} = \frac{\Omega_d(\varepsilon^{1-d})}{\sqrt{d}} = \Omega_d(\varepsilon^{1-d}).$$

The sparsity of N is $|E(N)|/|S| = \Omega_d(\varepsilon^{1-d}/\varepsilon^{(1-d)/2}) = \Omega_d(\varepsilon^{(1-d)/2})$, as required.

The MST for the point set S contains one unit-weight edge between S_0 and S_1, and the remaining $|S|-2$ edges each have weight $d\sqrt{\varepsilon}$, which is the minimum distance between lattice points in L (see [40] for the asymptotic behavior of the MST of a section of the lattice). Therefore $\|\mathrm{MST}(S)\| = 1+(|S|-2)d\sqrt{\varepsilon} = \Theta_d(\varepsilon^{1-d/2})$. It follows that the lightness of N is $\|N\|/\|\mathrm{MST}(S)\| = \Omega_d(\varepsilon^{1-d}/\varepsilon^{1-d/2}) = \Omega_d(\varepsilon^{-d/2})$, as claimed. This completes the proof when $n=\Theta_d(\varepsilon^{(1-d)/2})$.

General Case. Finally, if $n\geq|S|=\Theta_d(\varepsilon^{(1-d)/2})$, we can generalize the above construction by duplication. Assume that $n=k\,|S|$ for some integer $k\geq 1$. Let $Q_1,\ldots,Q_k$, be disjoint axis-aligned unit hypercubes, such that they each have an edge along the x_1-axis, and two consecutive cubes are at distance 3 apart. Let S' be the union of k translates of the point set S, on the boundaries of $Q_1,\ldots,Q_k$. Let N' be a Euclidean Steiner $(1+\varepsilon)$-spanner for S'.

Since the ellipses induced by point pairs in different copies of S are disjoint, we have $\|N'\|\geq k\,\|N\| = \Omega_d(k\varepsilon^{1-d})$ and $|E(N')|\geq k\,|E(N)|$. This immediately implies that the sparsity of N' is $|E(N')|/n = |E(N)|/|S| \geq \Omega_d(\varepsilon^{(1-d)/2})$.

The MST of S' consists of k translated copies of $\mathrm{MST}(S)$ and $k-1$ edges of weight 3 between consecutive copies. That is, $\|\mathrm{MST}(S')\| = k\,\|\mathrm{MST}(S)\|+3(k-1) = \Theta_d(k\varepsilon^{1-d/2})$. It follows that the lightness of N' is $\Omega_d(\varepsilon^{-d/2})$, as claimed. ◀

4 Upper Bound

In this section, we prove Theorem 2 and construct, for every $\varepsilon > 0$ and every set of n points in $\mathbb{R}^2$, a Euclidean Steiner $(1+\varepsilon)$-spanner of lightness $O(\varepsilon^{-1}\log n)$. This matches the lower bound of Theorem 1 up to a $O(\log n)$ factor, and improves upon the previous bound of $O(\varepsilon^{-1}\log\Delta)$ by Le and Solomon [28, Theorem 1.2].

Directional $(1+\varepsilon)$-spanners. Let S be a set of n points in the plane. Assume, without loss of generality, that $\mathrm{diam}(S) \geq 1/2$ and $S \subset [0,1]^2$. Then the weight of the Euclidean spanning tree of S is bounded by $\|\mathrm{MST}(S)\| \geq \mathrm{diam}(S) \geq 1/2$, and $\|\mathrm{MST}(S)\| \leq O(n^{1/2})$ by a classical result by Few [20]. Both bounds are tight in the worst case. Note that the weight of the MST is in a rather broad range, which makes it challenging to bound the weight of the Steiner $(1+\varepsilon)$-spanner by $O(\|\mathrm{MST}(S)\| \cdot \varepsilon^{-1})$.

The *direction* of a line L in the plane is given by the counterclockwise angle $\theta \in [0,\pi)$ between the positive x-axis and L. A line segment pq inherits its direction from its supporting line. The *distance* between directions $\theta_1, \theta_2 \in [0,\pi)$, of lines L_1 and L_2, resp., is

$$\angle(L_1, L_2) = \min\{|\theta_1 - \theta_2|, \pi - |\theta_1 - \theta_2|\}.$$

For an interval $D \subset [0,\pi)$ of directions, we construct a Euclidean Steiner $(1+\varepsilon)$-spanner restricted to points pairs whose directions are in D. We define a spanner in this restricted sense as follows.

▶ **Definition 6.** *Let S be a finite point set in $\mathbb{R}^2$, and let $D \subset [0,\pi)$ be an set of directions. A geometric graph G is a* directional $(1+\varepsilon)$-spanner *for S and D if for every $a, b \in S$, where the direction of ab is in D, graph G contains an ab-path of weight at most $(1+\varepsilon)\|ab\|$.*

Our main lemma is the following.

▶ **Lemma 7.** *For a set S of n points in the plane, and for the interval $D = [\frac{\pi-\sqrt{\varepsilon}}{8}, \frac{\pi+\sqrt{\varepsilon}}{8}]$ of directions, there exists a directional $(1+\varepsilon)$-spanner G of weight $O(\|MST(S)\|\varepsilon^{-1/2}\log n)$.*

We prove Lemma 7 in Section 4.2 below. Here we show that Lemma 7 implies Theorem 2.

▶ **Theorem 2.** *For every set S of n points in Euclidean plane, there exists a Steiner $(1+\varepsilon)$-spanner of lightness $O(\varepsilon^{-1}\log n)$.*

Proof of Theorem 2. Let S be a set of n points in the plane. Let $\varepsilon > 0$ be given, and let $k = \Theta(\varepsilon^{-1/2})$ be an integer. We partition the space of directions into k intervals as

$$[0,\pi) = \bigcup_{i=1}^{k} D_i, \text{ where } D_i = \left[\frac{(i-1)\,\pi}{k}, \frac{i\,\pi}{k}\right) \text{ for } i \in \{1,\ldots,k\}.$$

By Lemma 7, for $i = 1, \ldots, k$, there exists a geometric graph N_i of weight $O(\|\mathrm{MST}\|\log(n) \cdot \varepsilon^{-1/2})$ such that for every point pair $a, b \in S$, if the direction of ab is in D_i, then N_i contains an ab-path of weight at most $(1+\varepsilon)\|ab\|$.

Let $N = \bigcup_{i=1}^{k} N_i$ be the union of the networks N_i for $i \in \{1,\ldots,k\}$; see Figure 4 for an illustration. Since $[0,\pi) = \bigcup_{i=1}^{k} D_i$, the graph N contains a path of weight at most $(1+\varepsilon)\|ab\|$ for all point pairs $a, b \in S$, and so N is a Euclidean Steiner $(1+\varepsilon)$-spanner for S. The weight of N is

$$\|N\| = \sum_{i=1}^{k} \|N_i\| \leq k \cdot O\left(\frac{\|\mathrm{MST}(S)\|\log n}{\sqrt{\varepsilon}}\right) \leq O\left(\frac{\|\mathrm{MST}(S)\|\log n}{\varepsilon}\right),$$

as required. ◀

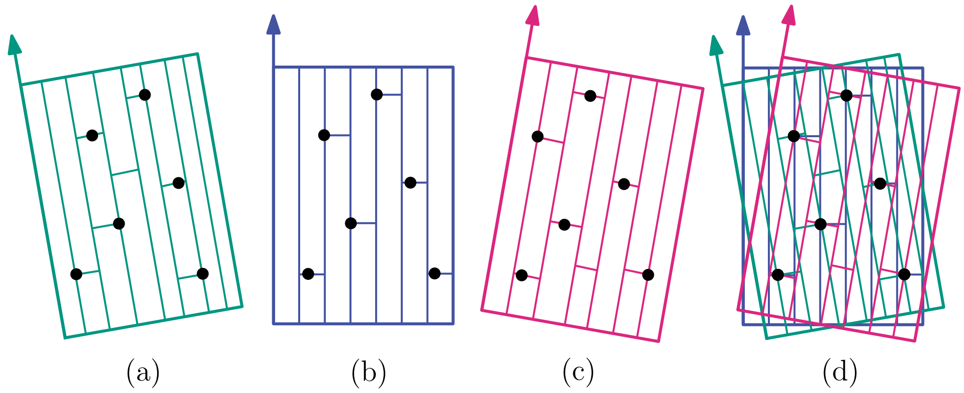


Figure 4 (a)–(c) Schematic figures for directional spanners N_1, N_2, and N_3 for three disjoint intervals of directions, for a set of 6 points in the plane. (d) The union $N_1 \cup N_2 \cup N_3$ of the networks.

The second component of our approach is a subdivision of the bounding box of S into regions that are long and skinny in one particular direction. We start with discussing the special cases of a single rectangle.

4.1 Rectangles

We illustrate our general strategy with a simple special case, where the points in S lie on the boundary of an axis-aligned rectangle. We first assume that R is narrow and tall, and we construct an directional $(1+\varepsilon)$-spanner for an interval of near-vertical directions. For further reference, we define the *aspect ratio* of an axis-parallel rectangle R by

$$\text{aratio}(R) = \frac{\text{width}(R)}{\text{height}(R)}.$$

▶ **Lemma 8.** *Let R be an axis-aligned rectangle with $\frac{1}{8}\sqrt{\varepsilon} \le aratio(R) \le \frac{1}{4}\sqrt{\varepsilon}$. Then for a finite point set S on the boundary of R, and for the interval $D = [\frac{\pi-\sqrt{\varepsilon}}{8}, \frac{\pi+\sqrt{\varepsilon}}{8}]$ of directions, there exists a directional $(1+\varepsilon)$-spanner G with $\|G\| \le O(height(R))$.*

Proof. We construct a graph G as a union of the boundary of R and a finite number of shallow-light trees. If both a and b are in the same side of R, then the boundary of R contains an ab-path of weight $\|ab\|$. We next consider cases in which a and b are in different sides of R. Since $\frac{1}{8}\sqrt{\varepsilon} \le \text{aratio}(R)$, if a and b are in the interior of the left and right side of R, resp., then the direction of the segment ab falls outside of D.

Let c be the center of the left edge of R; refer to Fig. 5(a). Take two shallow-light trees between c and each horizontal side of R with parameter $\varepsilon' = \varepsilon/4$. By Lemma 5, the weight of the two trees is $O(\text{height}(R))$. If a and b are on the top and bottom sides of R, resp., then $\|ab\| \ge \text{height}(R)$. The union of the two shallow-light trees rooted at c contains an ab-path of length

$$\begin{aligned}(1+\varepsilon')(\|as\| + \|sb\|) &\le (1+\varepsilon')\cdot 2\cdot\sqrt{\left(\frac{\text{height}(R)}{2}\right)^2 + \left(\frac{\text{width}(R)}{2}\right)^2}\\ &< \left(1+\frac{\varepsilon}{4}\right)\left(1+\frac{\varepsilon}{2}\right)\text{height}(R)\\ &< (1+\varepsilon)\,\text{height}(R)\\ &\le (1+\varepsilon)\|ab\|.\end{aligned}$$

It remains to construct ab-paths for point pairs on adjacent sides of R. We describe the construction for a left and bottom sides of R; the constructions for all other pairs of adjacent sides is analogous, and we can take the union of all constructions. Without loss of generality, assume that the lower-left corner of R is the origin o; see Fig. 5(b). For every positive integer $i \in \mathbb{N}^+$, let $p_i = (0, \text{height}(R)/2^i)$, and $q_i = (\text{width}(R)/2^{i-1})$. Note that p_i is on the left side of R, and q_i is on the bottom side of R for all $i \in \mathbb{N}^+$. By Lemma 5, there exists a shallow-light tree T_i with parameter $\varepsilon' = \varepsilon/4$ of weight $O(\text{height}(R)/2^i)$ from the root p_i to the line segment oq_i. Between any point $a \in p_{i-1}p_i$ and $b \in q_iq_{i+1}$, we can combine a vertical segment ap_i with a path in the tree T_i from p_i to b to obtain a path of weight at most $(1+\varepsilon')\|ab\|$.

The weight of the union of the trees T_i, for all $i \in \mathbb{N}^+$, is $O(\sum_{i=1}^{\infty} \text{height}(R)/2^i) = O(\text{height}(R))$. In fact, we do not need infinitely many trees, since S is finite, hence it contains a finite number of point in the interior of the left side of R. It suffices to construct the trees T_i, $i = 1, \ldots, m$, such that all points in S in the left side of R are at or above p_m. ◀

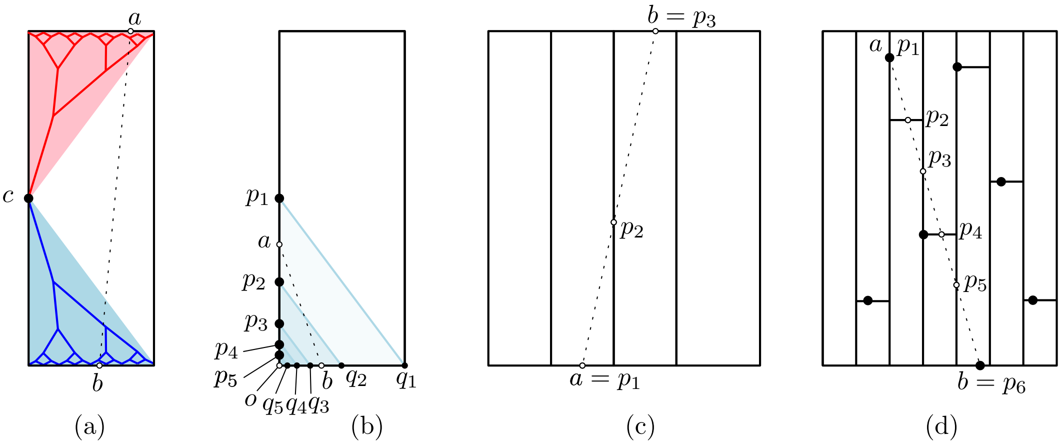


Figure 5 (a) A rectangle R with $\frac{1}{8}\sqrt{\varepsilon} \leq \text{aratio}(R) \leq \frac{1}{4}\sqrt{\varepsilon}$, and two shallow-light trees rooted at the midpoint c of the left side of R. (b) A sequence of shallow-light trees rooted at $p_1, \ldots, p_m$ on the left side of R. (c) A subdivision of a rectangle R. (d) A rectangulation of the bounding box of S into rectangels R with $\text{aratio}(R) \leq \frac{1}{4}\sqrt{\varepsilon} < \text{aratio}(R)$.

We can generalize Lemma 8 to axis-aligned rectangles of arbitrary aspect ratio.

▶ **Lemma 9.** *Let R be an axis-aligned rectangle. Then for a finite point set S on the boundary of R, and for the interval $D = [\frac{\pi-\sqrt{\varepsilon}}{8}, \frac{\pi+\sqrt{\varepsilon}}{8}]$ of directions, there exists a directional $(1+\varepsilon)$-spanner G of weight $O(\text{height}(R) + \text{width}(R)/\sqrt{\varepsilon})$.*

Proof. If $\frac{1}{8}\sqrt{\varepsilon} \leq \text{aratio}(R) \leq \frac{1}{4}\sqrt{\varepsilon}$, then Lemma 8 completes the proof. Otherwise, we greedily subdivide R by parallel lines as follows. First assume that $\frac{1}{4}\sqrt{\varepsilon} < \text{aratio}(R)$; refer to Fig. 5(c). For a vertical line L, denote by L^- and L^+, resp., the halfplanes on the left and right of L. Let L be the leftmost vertical line such that $\text{aratio}(R \cap L^-) = \frac{1}{8}\sqrt{\varepsilon}$. Then subdivide R into $R \cap L^-$ and $R \cap L^+$ by a vertical segment of weight height(R), and recurse on $R \cap L^+$. All subdivision edges are vertical, of weight height(R), and their total weight is

$$\text{height}(R) \cdot \left\lfloor \frac{\text{width}(R)/\text{height}(R)}{\frac{1}{8}\sqrt{\varepsilon}} \right\rfloor \leq O\left(\frac{\text{width}(R)}{\sqrt{\varepsilon}}\right). \tag{5}$$

Similarly, if $\text{aratio}(R) < \frac{1}{8}\sqrt{\varepsilon}$, we can greedily subdivide R by horizontal lines into axis-parallel rectangles. For a horizontal line L, denote by $L^\uparrow$ and $L^\downarrow$, resp., the halfplanes above and below L. Let L be the highest horizontal line such that $\mathrm{aratio}(R \cap L^{\uparrow}) = \frac{1}{4}\sqrt{\varepsilon}$. Then subdivide R into $R \cap L^{\uparrow}$ and $R \cap L^{\downarrow}$ by a horizontal segment of weight $\mathrm{width}(R)$, and recurse on $R \cap L^{\downarrow}$. In this case, all subdivision edges are horizontal, they all are of weight $\mathrm{width}(R)$, and their total weight is

$$\mathrm{width}(R) \cdot \left\lfloor \frac{\frac{1}{4}\sqrt{\varepsilon}}{\mathrm{width}(R)/\mathrm{height}(R)} \right\rfloor \leq O\left(\sqrt{\varepsilon} \cdot \mathrm{height}(R)\right). \tag{6}$$

Assume that R has been subdivided into rectangles $R_1, \ldots, R_k$, such that $\frac{1}{8}\sqrt{\varepsilon} \leq \mathrm{aratio}(R_i) \leq \frac{1}{4}\sqrt{\varepsilon}$. For every $i \in \{1, \ldots, k\}$, let S_i be the set of intersection points between the boundary of R_i and the line segments spanned by S. For the point set S_i and the directions D, Lemma 8 yields a directional $(1+\varepsilon)$-spanner G_i of weight $\|G_i\| = \mathrm{height}(R_i)$.

Let $G = \bigcup_{i=1}^{k} G_i$. From (5) and (6), we get $\|G\| = \sum_{i=1}^{k} \|G_i\| = O(\sum_{i=1}^{k} \mathrm{height}(R_i)) = O(\mathrm{height}(R) + \mathrm{width}(R)/\sqrt{\varepsilon})$, as required. It remains to show that G is a directional $(1+\varepsilon)$-spanner. Let $a, b \in S$ with direction in D; see Fig. 5(c). The vertical edges of $R_1, \ldots, R_k$ subdivide ab into a path of collinear line segments $a = p_0, \ldots, p_\ell = b$. Each segment $p_{i-1}p_i$ lies in some rectangle R_j between points $p_{i-1}, p_i \in S_j$, and so G_j contains a $p_{i-1}p_i$-path of weight at most $(1+\varepsilon)\|p_{i-1}p_i\|$. The concatenation of these paths is an ab-path of weight at most $\sum_{i=1}^{\ell}(1+\varepsilon)\|p_{i-1}p_i\| = (1+\varepsilon)\|ab\|$. ◀

4.2 Rectangulations

Let $\mathrm{per}(P)$ denote the perimeter of P. A polygon P is *rectilinear* if every edge is horizontal or vertical. A *rectangulation* of polygon P is a subdivision of P into axis-parallel rectangles. De Berg and van Kreveld [15] proved that for a rectilinear simple polygon P with n vertices, one can efficiently compute a rectangulation of weight $O(\mathrm{per}(P)\log n)$, and this bound is the best possible (already for stair-case polygons).

For an arbitrary set S of n points in the plane, we can construct a rectangulation of the axis-aligned bounding box of S with weight $O(\|\mathrm{MST}\|\log n)$. Combining such a rectangulation with Lemma 9, we are now ready to prove Lemma 7.

▶ **Lemma 7.** *For a set S of n points in the plane, and for the interval $D = [\frac{\pi-\sqrt{\varepsilon}}{8}, \frac{\pi+\sqrt{\varepsilon}}{8}]$ of directions, there exists a directional $(1+\varepsilon)$-spanner G of weight $O(\|MST(S)\|\varepsilon^{-1/2}\log n)$.*

Proof. Let T be the rectilinear MST of S, that is, a spanning tree of minimum weight w.r.t. L_1-norm, realized in the plane such that every edge is an L-shape (the union of a horizontal and a vertical segment). It is well known that $\|T\| \leq \sqrt{2}\,\|\mathrm{MST}(S)\|$. Let R be the minimum axis-aligned bounding box of T. By the minimality of R, the boundary of R contains at least two vertices of T. Consequently, $T \cup \partial R$ is connected, and it subdivides the interior of R into rectilinear simple polygons (*faces*) of total weight at most $2(\|T\| + \mathrm{per}(R)) = O(\|\mathrm{MST}(S)\|)$.

By the result of de Berg and van Kreveld [15], we can rectangulate each face of $T \cup \partial R$. Let $\mathcal{R}$ denote the resulting rectangulation. Since every face has $O(n)$ vertices, and every edge is on the boundary of at most two faces, the total perimeter of the rectangles in $\mathcal{R}$ is $\sum_{R\in\mathcal{R}} \mathrm{per}(R) = O((\|T\| + \mathrm{per}(R))\log n) = O(\|\mathrm{MST}(S)\|\log n)$.

For every $R \in \mathcal{R}$, let $S(R)$ be the set of intersection points between the boundary of R and the line segment induced by S. By Lemma 9, there exists a directional Euclidean Steiner $(1+\varepsilon)$-spanner $G(R)$ for $S(R)$ of weight $O(\mathrm{per}(R)\cdot\varepsilon^{-1/2})$. Let $G = \bigcup_{R\in\mathcal{R}} G(R)$. Its total weight $\|G\| = \sum_{R\in\mathcal{R}} O(\mathrm{per}(R)\cdot\varepsilon^{-1/2}) = O(\|\mathrm{MST}(S)\|\,\varepsilon^{-1/2}\log n)$. We can verify that G is a directional $(1+\varepsilon)$-spanner for S, similarly to the proof of Lemma 9. Let $a, b \in S$ such that the directions of ab is in D; see Fig. 5(d). The rectangulation subdivides ab into a path of collinear segments $a = p_0, \ldots, p_\ell = b$. Each segment $p_{i-1}p_i$ lies in some rectangle $R \in \mathcal{R}$ between points $p_{i-1}, p_i \in S(R)$, and so $G(R)$ contains a $p_{i-1}p_i$-path of weight at most $(1+\varepsilon)\|p_{i-1}p_i\|$. The concatenation of these paths is an ab-path of weight at most $\sum_{i=1}^{\ell}(1+\varepsilon)\|p_{i-1}p_i\| = (1+\varepsilon)\|ab\|$, as required. ◀

▶ Remark 10. The $\log(n)$-factor in Theorem 2 is due to the rectangulations of rectilinear polygons with $O(n)$ vertices. Instead of rectangulations, one could use a minimum-weight Steiner subdivision into convex faces (assuming that Lemmas 8–9 generalize to *convex* polygons). However, this approach would not yield more than a $\log\log(n)$-factor improvement. Dumitrescu and Tóth [17] probed that every simple polygon P with n vertices admits a convex subdivision of weight $O(\text{per}(P)\log n/\log\log n)$, and this bound is the best possible.

▶ Remark 11. Instead of a rectangulation, one could also use a subdivision into histograms. A *histogram* is a rectilinear simple polygon bounded by three axis-parallel line segments and one x- or y-monotone path. A classical *window partition* [32, 41] subdivides a simple rectilinear polygon P into histograms such that every axis-parallel line segment in P intersects (*stabs*) at most three histograms [18, 31]. Due to the stabbing property, the total perimeter of the resulting histograms is $O(\text{per}(P))$. For a point set S, this approach yields a histogram subdivision of the bounding box of S with weight $O(\|\text{MST}(S)\|)$. Very recently, Bhore and Tóth [4] improved the upper bound $O(\varepsilon^{-1}\log n)$ of Theorem 2 to $O(\varepsilon^{-1})$ by combining directional spanners with a modified window partition.

5 Conclusions

In this paper, we have studied Euclidean Steiner $(1+\epsilon)$-spanners under two optimization criteria, *lightness* and *sparsity*, and provided improved lower and upper bounds. Our upper bound of $O(\varepsilon^{-1}\log n)$ on the minimum lightness of Steiner $(1+\varepsilon)$-spanners in the Euclidean plane has recently been improved to the bound $O(\varepsilon^{-1})$ in [4], matching the lower bound of $\Omega(\varepsilon^{-1})$ of Theorem 1. However, for lightness in dimensions $d \geq 3$, an $\tilde{\Theta}(\varepsilon^{1/2})$-factor gap remains between the current upper bound $\tilde{O}(\varepsilon^{-(d+1)/2})$ [30, Theorem 1.6] and the lower bound $\Omega(\varepsilon^{-d/2})$ of Theorem 1.

In Euclidean d-space, the same point sets (grids in two parallel hyperplanes) establish the lower bounds $\Omega(\varepsilon^{-d/2})$ for lightness and $\Omega(\varepsilon^{(1-d)/2})$ for sparsity for Steiner $(1+\varepsilon)$-spanners (cf. Theorem 1). Le and Solomon constructed spanners with sparsity $\tilde{O}(\varepsilon^{(1-d)/2})$ [27, Theorem 1.3], matching the lower bound in every dimension $d \in \mathbb{N}$, but the lightness of these spanners is significantly higher. In dimensions $d \geq 3$, they construct spanners with lightness $\tilde{O}(\varepsilon^{-(d+1)/2})$ [30, Theorem 1.6], but these spanners have significantly higher sparsity.

We conjecture that a Euclidean Steiner $(1+\varepsilon)$-spanner cannot simultaneously attain both lower bounds (lightness and sparsity) of Theorem 1. Therefore, exploring trade-offs between lightness and sparsity in Euclidean d-space remains an open problem.

A critical aspect of graphs is their embeddibility in low-genus surfaces. A geometric graph in $\mathbb{R}^2$ is a *plane graph* if no two edges cross each other. Note that every Steiner spanner $G = (V, E)$ for a point set S can be turned into a plane graph (with the same stretch factor ratio and the same weight) by introducing Steiner points at every edge crossing. However, the number of new Steiner points would be proportional to $O(|E|^2)$, which is prohibitive. It remains an open problem to bound the sparsity of a *plane* Steiner $(1+\varepsilon)$-spanner for n points in Euclidean plane, as a function of n and ε.

Angles and directions play a crucial role in our lower bound analysis (Section 3) and upper bound construction (Section 4). While angles are invariant under rotations only in Euclidean spaces, they can be defined in any inner product space, such as $\mathbb{R}^d$ under L_p norm, for $p \geq 2$. We leave it as an open problem to derive upper and lower bounds on the lightness and sparsity of Steiner $(1+\varepsilon)$-spanners in other inner product spaces.

References

1 Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993. doi:10.1007/BF02189308.

2 Sunil Arya, David M. Mount, and Michiel Smid. Randomized and deterministic algorithms for geometric spanners of small diameter. In *Proc. 35th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 703–712, 1994. doi:10.1109/SFCS.1994.365722.

3 Sunil Arya and Michiel Smid. Efficient construction of a bounded-degree spanner with low weight. *Algorithmica*, 17(1):33–54, 1997. doi:10.1007/BF02523237.

4 Sujoy Bhore and Csaba D. Tóth. Light Euclidean Steiner spanners in the plane. *Preprint*, 2020. arXiv:2012.02216.

5 Glencora Borradaile, Hung Le, and Christian Wulff-Nilsen. Greedy spanners are optimal in doubling metrics. In *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2371–2379, 2019. doi:10.1137/1.9781611975482.145.

6 Prosenjit Bose, Joachim Gudmundsson, and Michiel Smid. Constructing plane spanners of bounded degree and low weight. *Algorithmica*, 42(3-4):249–264, 2005. doi:10.1007/s00453-005-1168-8.

7 Paul B. Callahan. Optimal parallel all-nearest-neighbors using the well-separated pair decomposition. In *Proc. 34th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 332–340, 1993. doi:10.1109/SFCS.1993.366854.

8 T.-H. Hubert Chan, Mingfei Li, Li Ning, and Shay Solomon. New doubling spanners: Better and simpler. *SIAM J. Comput.*, 44(1):37–53, 2015. doi:10.1137/130930984.

9 L. Paul Chew. There is a planar graph almost as good as the complete graph. In *Proc. 2nd Symposium on Computational Geometry*, pages 169–177. ACM Press, 1986. doi:10.1145/10515.10534.

10 L. Paul Chew. There are planar graphs almost as good as the complete graph. *J. Comput. Syst. Sci.*, 39(2):205–219, 1989. doi:10.1016/0022-0000(89)90044-5.

11 Kenneth L. Clarkson. Approximation algorithms for shortest path motion planning. In *Proc. 19th ACM Symposium on Theory of Computing (STOC)*, pages 56–65, 1987. doi:10.1145/28395.28402.

12 Gautam Das, Paul Heffernan, and Giri Narasimhan. Optimally sparse spanners in 3-dimensional Euclidean space. In *Proc. 9th Symposium on Computational Geometry (SoCG)*, pages 53–62. ACM Press, 1993. doi:10.1145/160985.160998.

13 Gautam Das and Deborah Joseph. Which triangulations approximate the complete graph? In *Proc. International Symposium on Optimal Algorithms*, pages 168–192. Springer, 1989. doi:10.1007/3-540-51859-2_15.

14 Gautam Das, Giri Narasimhan, and Jeffrey S. Salowe. A new way to weigh malnourished Euclidean graphs. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 215–222, 1995. URL: https://dl.acm.org/doi/10.5555/313651.313697.

15 Mark de Berg and Marc J. van Kreveld. Rectilinear decompositions with low stabbing number. *Information Processing Letters*, 52(4):215–221, 1994. doi:10.1016/0020-0190(94)90129-5.

16 Michael J. Demmer and Maurice P. Herlihy. The arrow distributed directory protocol. In *Proc. 12th Symposium on Distributed Computing (DISC)*, volume 1499 of *LNCS*, pages 119–133. Springer, 1998. doi:10.1007/BFb0056478.

17 Adrian Dumitrescu and Csaba D. Tóth. Minimum weight convex Steiner partitions. *Algorithmica*, 60(3):627–652, 2011. doi:10.1007/s00453-009-9329-9.

18 Herbert Edelsbrunner, Joseph O'Rourke, and Emmerich Welzl. Stationing guards in rectilinear art galleries. *Computer Vision, Graphics, and Image Processing*, 27(2):167–176, 1984. doi:10.1016/S0734-189X(84)80041-9.

19 Michael Elkin and Shay Solomon. Steiner shallow-light trees are exponentially lighter than spanning ones. *SIAM J. Comput.*, 44(4):996–1025, 2015. doi:10.1137/13094791X.

20 L. Few. The shortest path and the shortest road through n points. *Mathematika*, 2(2):141–144, 1955. doi:10.1112/S0025579300000784.

21 Lee-Ad Gottlieb, Aryeh Kontorovich, and Robert Krauthgamer. Efficient regression in metric spaces via approximate Lipschitz extension. *IEEE Transactions on Information Theory*, 63(8):4838–4849, 2017. doi:10.1109/TIT.2017.2713820.

22 Joachim Gudmundsson, Christos Levcopoulos, and Giri Narasimhan. Fast greedy algorithms for constructing sparse geometric spanners. *SIAM J. Comput.*, 31(5):1479–1500, 2002. doi:10.1137/S0097539700382947.

23 Joachim Gudmundsson, Christos Levcopoulos, Giri Narasimhan, and Michiel Smid. Approximate distance oracles for geometric spanners. *ACM Transactions on Algorithms (TALG)*, 4(1):1–34, 2008. doi:10.1145/1328911.1328921.

24 Maurice Herlihy, Srikanta Tirthapura, and Rogert Wattenhofer. Competitive concurrent distributed queuing. In *Proc. 20th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 127–133, 2001. doi:10.1145/383962.384001.

25 J. Mark Keil. Approximating the complete Euclidean graph. In *Proc. 1st Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 318 of *LNCS*, pages 208–213. Springer, 1988. doi:10.1007/3-540-19487-8_23.

26 J. Mark Keil and Carl A. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete & Computational Geometry*, 7:13–28, 1992. doi:10.1007/BF02187821.

27 Hung Le and Shay Solomon. Truly optimal Euclidean spanners. In *Proc. 60th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1078–1100, 2019. doi:10.1109/FOCS.2019.00069.

28 Hung Le and Shay Solomon. Light Euclidean spanners with Steiner points. In *Proc. 28th European Symposium on Algorithms (ESA)*, volume 173 of *LIPIcs*, pages 67:1–67:22. Schloss Dagstuhl, 2020. doi:10.4230/LIPIcs.ESA.2020.67.

29 Hung Le and Shay Solomon. Light Euclidean spanners with Steiner points. *Preprint*, 2020. Version 2. arXiv:2007.11636.

30 Hung Le and Shay Solomon. A unified and fine-grained approach for light spanners. *Preprint*, 2020. arXiv:2008.10582.

31 Christos Levcopoulos. *Heuristics for Minimum Decompositions of Polygons*. PhD thesis, Linköping, 1987. No. 74 of Linköping Studies in Science and Technology.

32 Anil Maheshwari, Jörg-Rüdiger Sack, and Hristo N. Djidjev. Link distance problems. In Jörg-Rüdiger Sacks and Jorge Urutia, editors, *Handbook of Computational Geometry*, chapter 12, pages 519–558. North-Holland, 2000. doi:10.1016/B978-044482537-7/50013-9.

33 Giri Narasimhan and Michiel Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007. doi:10.1017/CBO9780511546884.

34 David Peleg and Jeffrey D. Ullman. An optimal synchronizer for the hypercube. *SIAM J. Comput.*, 18(4):740–747, 1989. doi:10.1137/0218050.

35 David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *Journal of the ACM (JACM)*, 36(3):510–530, 1989. doi:10.1145/65950.65953.

36 Satish B. Rao and Warren D. Smith. Approximating geometrical graphs via "spanners" and "banyans". In *Proc. 13th ACM Symposium on Theory of Computing (STOC)*, pages 540–550, 1998. doi:10.1145/276698.276868.

37 Jim Ruppert and Raimund Seidel. Approximating the d-dimensional complete Euclidean graph. In *Proc. 3rd Canadian Conference on Computational Geometry (CCCG)*, pages 207–210, 1991.

38 Christian Schindelhauer, Klaus Volbert, and Martin Ziegler. Geometric spanners with applications in wireless networks. *Computational Geometry*, 36(3):197–214, 2007. doi:10.1016/j.comgeo.2006.02.001.

39 Shay Solomon. Euclidean Steiner shallow-light trees. *J. Comput. Geom.*, 6(2):113–139, 2015. doi:10.20382/jocg.v6i2a7.

40 J. Michael Steele and Timothy Law Snyder. Worst-case growth rates of some classical problems of combinatorial optimization. *SIAM J. Comput.*, 18(2):278–287, 1989. doi:10.1137/0218019.

41 Subhash Suri. On some link distance problems in a simple polygon. *IEEE Trans. Robotics Autom.*, 6(1):108–113, 1990. doi:10.1109/70.88124.

42 Andrew Chi-Chih Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comput.*, 11(4):721–736, 1982. doi:10.1137/0211059.

A Nearly Optimal Deterministic Online Algorithm for Non-Metric Facility Location

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

Björn Feldkord
Heinz Nixdorf Institut & Department of Computer Science, Universität Paderborn, Germany

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

Abstract

In the online non-metric variant of the facility location problem, there is a given graph consisting of a set F of facilities (each with a certain opening cost), a set C of potential clients, and weighted connections between them. The online part of the input is a sequence of clients from C, and in response to any requested client, an online algorithm may open an additional subset of facilities and must connect the given client to an open facility.

We give an online, polynomial-time deterministic algorithm for this problem, with a competitive ratio of $O(\log|F| \cdot (\log|C| + \log\log|F|))$. The result is optimal up to loglog factors. Our algorithm improves over the $O((\log|C| + \log|F|) \cdot (\log|C| + \log\log|F|))$-competitive construction that first reduces the facility location instance to a set cover one and then later solves such instance using the deterministic algorithm by Alon et al. [TALG 2006]. This is an asymptotic improvement in a typical scenario where $|F| \ll |C|$.

We achieve this by a more direct approach: we design an algorithm for a fractional relaxation of the non-metric facility location problem with clustered facilities. To handle the constraints of such non-covering LP, we combine the dual fitting and multiplicative weight updates approach. By maintaining certain additional monotonicity properties of the created fractional solution, we can handle the dependencies between facilities and connections in a rounding routine.

Our result, combined with the algorithm by Naor et al. [FOCS 2011] yields the first deterministic algorithm for the online node-weighted Steiner tree problem. The resulting competitive ratio is $O(\log k \cdot \log^2 \ell)$ on graphs of ℓ nodes and k terminals.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Routing and network design problems

Keywords and phrases Online algorithms, deterministic rounding, linear programming, facility location, set cover

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.14

Funding Supported by Polish National Science Centre grant 2016/22/E/ST6/00499 and by German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" under the project number 160364472 – SFB 901/3.

Acknowledgements We thank Marek Adamczyk and Christine Markarian for helpful discussions. We thank anonymous reviewers of an earlier draft for pointing us to the reduction of Kolen and Tamir [20].

1 Introduction

The facility location (FL) problem [1] is one of the best-known examples of network design problems, extensively studied both in operations research and in computer science. Its simple definition, NP-hardness, and rich combinatorial structure have led to developments of tools and solutions in key areas of approximation algorithms, combinatorial optimization, and linear programming.

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 14; pp. 14:1–14:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

An instance of the FL problem consists of a set F of facilities, each with a certain opening cost, and a set C of clients. F and C can be seen as two sides of a bipartite graph. The undirected edges between them have lengths that can either satisfy the triangle inequality (*metric FL*) or be arbitrary (*non-metric FL*). The goal is to open a subset of facilities and connect each client to an open facility. The total cost (the sum of opening and connection costs) is subject to minimization. In the metric scenario, by taking a metric closure, one can assume that each facility is reachable by each client, but it is not the case for the non-metric variant.

Instances and Objectives. In this paper, we focus on an online variant of the non-metric FL problem. We first formalize the offline variant in a way that makes a connection to the online variant more apparent.

A *facility-client graph* $G = (F, C, E, \mathsf{cost})$ is a bipartite graph, whose one side is the set F of facilities and another side is the set of clients C. Set $E \subseteq F \times C$ contains available facility-client connections (edges). We use function cost to denote both costs of opening facilities and connection costs (edge lengths). All costs are non-negative.

An instance of the non-metric FL problem is a pair (G, A), where $G = (F, C, E, \mathsf{cost})$ is a facility-client graph and $A \subseteq C$ is a subset of *active* clients. A feasible solution to such instance is a set of open (purchased) facilities $F' \subseteq F$ and a subset of purchased edges $E' \subseteq E$, such that any active client $c \in A$ is connected by a purchased edge to an open facility. The cost of such solution is equal to $\sum_{f \in F'} \mathsf{cost}(f) + \sum_{e \in E'} \mathsf{cost}(e)$.

For any facility-client graph G, we define its aspect ratio Δ_G as the ratio of the largest to smallest positive cost in G. These costs include both facilities and connection costs.[1] Note that the aspect ratio is a property of G and is independent of the set of active clients A.

Online Scenario. In an *online variant* of the FL problem, the facility-client graph G is known in advance, but neither elements of A nor its cardinality are known up-front by an online algorithm $\textsc{Alg}$. The clients from A appear one by one. Upon seeing a new active client, $\textsc{Alg}$ may purchase additional facilities and edges, with the requirement that facilities and edges purchased so far must constitute a feasible solution to all presented active clients. The total cost of $\textsc{Alg}$ is denoted by $\textsc{Alg}(G, A)$. (We sometimes use $\textsc{Alg}(G, A)$ to also denote the solution computed by $\textsc{Alg}$.) Purchase decisions are final and cannot be revoked later. The goal is to minimize *the competitive ratio*, defined as $\sup_{(G,A)}\{\textsc{Alg}(G, A)/\textsc{Opt}(G, A)\}$, where $\textsc{Opt}$ is the optimal (offline) algorithm.

1.1 Related Work

Most of the prior work has been devoted to the offline scenario. While the metric variant of the FL problem admits O(1)-approximation algorithms [9, 10, 11, 18, 19, 22, 23, 24, 29], the best approximation ratio for the non-metric one is $O(\log |A|)$ [17], and it cannot be asymptotically improved unless $\mathsf{NP} \subseteq \mathsf{DTIME}[n^{O(\log\log n)}]$ [12]. For a more comprehensive treatment of the offline scenario, including a multitude of variants, we refer the reader to the entry in the Encyclopedia of Algorithms [1] or the survey by Shmoys [28].

For the online metric FL, the problem was resolved over ten years ago by Meyerson [25] and Fotakis [14]: the lower and upper bounds on the competitive ratio are $\Theta(\log |A| / \log\log |A|)$, both for deterministic and randomized algorithms. Simpler deterministic algorithms attaining

[1] In the standard definition of the aspect ratio, only distances are taken into account.

slightly worse competitive ratio of $O(\log |A|)$ were given by Anagnostopoulos et al. [4] and Fotakis [13]. Note that the optimal competitive ratio in the metric case is independent of the set C of potential clients.

1.2 Previous Work on Online Non-Metric Facility Location

For the non-metric FL, the first and currently best online algorithm was a *randomized* algorithm by Alon et al. [2]. It achieves the competitive ratio of $O(\log |F| \cdot \log |A|)$. It is based on solving a natural fractional relaxation of the problem: there is a fractional *opening variable* y_f for each facility f and a *connection variable* $x_{c,f}$ for a client c and a *covering* facility f (facility to which c could be connected). Once a client c arrives, for each covering facility f independently, their algorithm increases either y_f or $x_{c,f}$, whichever is smaller, using multiplicative update method (see, e.g., [5]). The client c is considered fractionally served once the sum of terms $\min\{x_{c,f}, y_f\}$ over all covering facilities is at least 1. The resulting competitive ratio is $O(\log |F|)$.

The computed fractional solution can be then rounded using a random threshold θ_f common for an opening variable y_f and all connection variables involving facility f. Once any variable exceeds its threshold, it is rounded up to 1 and the corresponding object (facility or connection) is purchased. Dynamically adjusting θ_f to have expectation $\Theta(1/\log |A|)$ guarantees that the resulting integral solution is feasible with high probability and the rounding part incurs a factor of $O(\log |A|)$ in the competitive ratio.

To the best of our knowledge, no non-trivial deterministic algorithm was published so far. In particular, the online network design problems (including the non-metric FL problem) have been listed as unresolved challenges by Buchbinder and Naor [8, Section 1.1]. That said, the non-metric facility location can be reduced to a set cover. A usable reduction (not inducing an exponential blow-up of the input size) was given by Kolen and Tamir [20]: it preserves the solution costs up to constant factors and creates a set cover instance consisting of $m = \Theta(|F| + |C| \cdot \log \Delta_G)$ sets and $n = \Theta(|C| \cdot \log \Delta_G)$ elements. Using doubling techniques described in Section 4, one could assume that $\Delta_G = O(|F| \cdot |C|)$. Applying the deterministic algorithm for the online set cover problem by Alon et al. [3] yields a solution whose competitive ratio is $O(\log m \cdot \log n) = O((\log |C| + \log |F|) \cdot (\log |C| + \log \log |F|))$.

1.3 Our Result

In our paper, we improve the bound above, replacing the first factor of $O(\log |C| + \log |F|)$ by $O(\log |F|)$. This is an asymptotic improvement in a typical scenario where $|F| \ll |C|$.

▶ **Theorem 1.** *There exists a deterministic polynomial-time $O(\log |F| \cdot (\log |C| + \log \log |F|))$-competitive algorithm for the online non-metric facility location problem on set F of facilities and set C of clients.*

Our algorithm attains a nearly optimal competitive ratio, as no deterministic algorithm can have a ratio smaller than $\Omega(\log |F| \cdot \log |C| / (\log \log |F| + \log \log |C|))$. This follows by the lower bound for the online set cover problem [2, 3] and holds even for uniform facility costs. If we restrict our attention to the *polynomial-time* deterministic solution, then a stricter lower bound of $\Omega(\log |F| \cdot \log |C|)$ holds (assuming $\mathsf{BPP} \neq \mathsf{NP}$) [21].

Challenges. The description of the randomized algorithm by Alon et al. [2] given above seems deceptively simple, but it hides an important and subtle property, implicitly exploited by the authors. Namely, the threshold θ_f is common for facility f and all connections to it.

This ensures the necessary dependency: once $\min\{x_{c,f}, y_f\} \geq \theta_f$, the rounding purchases *both facility f and a connection from c to f*. (Note that the left-hand side of this inequality is the amount that their fractional solution controls.)

It is unclear how to directly extend this property to deterministic rounding. A straightforward attempt would be to focus on facilities only and round them in a deterministic fashion ensuring the necessary coverage of each client. However, neglecting the connection costs in the rounding process easily leads to a situation, where the facilities are rounded "correctly", but the cost of connecting a client to the closest open facility in the integral solution is incomparably larger than the corresponding fractional cost.

We note that all known deterministic schemes that round fractional solutions generated by the multiplicative updates operate in rather limited scenarios, where elements have to be covered or packed and all important interactions between elements are handled at the time of constructing the fractional solution. This is the case for the deterministic rounding for the set cover problem [3, 7] and the throughput-competitive virtual circuit routing problem [6, 8]. These methods are based on derandomizing the method of pessimistic estimators [27] in an online manner, by transforming a pessimistic estimator into a potential function [30] that can be controlled by the deterministic rounding process.

Our Techniques. In our solution, we create a new linear relaxation of the problem. We first round the graph distances to powers of 2. For any client, we cluster facilities that have the same distance to this client. (Note that such clusters are client-dependent.) To solve the fractional variant, we run two schemes in parallel: we increase connection variables $x_{c,t}$ corresponding to clusters at distance t, and increase facility variables y_f for all facilities in "reachable" clusters (where the corresponding connection variables are 1). The increases in these variables use two different frameworks: dual fitting for linear increases of connection variables and a primal-dual scheme involving multiplicative updates for facility variables. Ensuring an appropriate balance between these two different types of updates is one of the technical difficulties that we tackle in this paper.

We stop increasing variables once there exists a collection of clusters that are both "fractionally open" (sum of variables y_f within these clusters is $\Omega(1)$) and "reachable" by the considered client. To argue about the existence of such a collection, we use both LP inequalities and structural properties of our fractional algorithm.

Finally, we construct a deterministic rounding routine. We focus on facilities only, neglecting whether particular clients are active or not and how far they are from a given facility. However, we strengthen rounding properties, ensuring, for (some) collections of clusters, that if the sum of opening variables in these collections is $\Omega(1)$, then the integral solution contains an open facility in one of these clusters. This ensures that, for a considered client c, the integral solution contains a facility whose distance from c is asymptotically not larger than the cost invested for connecting c in the fractional solution. Ultimately, this yields the desired dependency between facilities and connections.

Note about Up-Front Knowledge of the Facility-Client Graph. Unlike for the randomized variant, obtaining sub-linear guarantees for a deterministic solution requires knowing a priori the set of potential client-facility connections. To see this, consider a graph of $|F|$ facilities with unit opening costs and the set of $|C| = |F|$ clients. The graph edges are constructed dynamically as clients are activated and all revealed possible connections are of cost 0. The first active client can be connected to all facilities. Each subsequent client can be connected to all facilities but the ones already open by an algorithm. This way an online algorithm needs to eventually open all facilities, for a total cost of $|F|$. On the other hand, the offline optimal algorithm can open the last facility opened by an online algorithm and connect all clients to this facility paying just 1. Thus, under the unknown-graph assumption, the competitive ratio of any deterministic algorithm would be at least $|F|$.

1.4 Preliminaries and Paper Organization

Let T_G contain all powers of two between the largest and the smallest positive distance (inclusively) and also number 0. In particular, T_G contains all distances in G and $|T_G| \leq 2 + \log \Delta_G$. Whenever G is clear from the context, we drop the G subscript.

We may assume that F contains at least two facilities and C contains at least two clients, as otherwise the problem becomes trivial. For a facility $f \in F$, let $\mathsf{set}(f)$ be the set of clients that may be connected to f. For any client $c \in C$ and distance $t \in T$, cluster $F_{c,t}$ contains all facilities that are incident to c using edges of cost t. Note that for a fixed c, clusters $F_{c,t}$ are disjoint (no client has two connections of different costs to the same facility).

Powers-of-Two Assumption. In the whole paper, we assume that all facilities and connection costs are either equal to 0 or are powers of 2 and are at least 1. This can be easily achieved by initial scaling of positive costs and distances, so that they are at least 1 and rounding positive ones up to the nearest power of two. This transformation changes the competitive ratio at most by a factor of 2.

Paper Overview. Our core approach is to solve a carefully crafted fractional relaxation of the problem (Section 2), and then round it in a deterministic fashion (Section 3). This way, we obtain a deterministic online algorithm $\textsc{Int}$ that on any input $(G = (F, C, E, \mathsf{cost}), A)$ computes a feasible solution of cost

$$\textsc{Int}(G, A) \leq O(\log |F| \cdot (\log |C| + \log \log \Delta_G)) \cdot \textsc{Opt}(G, A) + 2 \cdot \max_{f \in F} \mathsf{cost}(f).$$

Moreover $\textsc{Int}$ runs in time $\text{poly}(|G|, |A|, \max_{e \in E} \mathsf{cost}(e), \max_{f \in F} \mathsf{cost}(f))$. In Section 4, we apply doubling and edge pruning techniques, to get rid of dependencies on costs in the running time and on Δ_G in the competitive ratio, achieving guarantees of Theorem 1.

Application to Node-Weighted Steiner Tree. Our result has an immediate application to the online node-weighted Steiner tree (NWST) problem. Namely, when we combine Theorem 1 with the randomized solution for NWST by Naor et al. [26], we obtain the first deterministic algorithm with polylogarithmic competitive ratio (see Section 5).

2 Fractional Solution

We fix an instance $(G = (F, C, E, \mathsf{cost}), A)$ of the online non-metric facility problem. For each facility f, we introduce an *opening* variable $y_f \geq 0$ (fractional opening of f) and for each client c and each distance $t \in T$ a *connection* variable $x_{c,t} \geq 0$. Intuitively, $x_{c,t}$ denotes how much, fractionally, client c invests into connections to facilities from cluster $F_{c,t}$. For any set F' of facilities we use $y(F')$ as a shorthand for $\sum_{f \in F'} y_f$.

Primal Program. After k clients from A arrive (we denote their set by A_k), we consider the following linear program $\mathcal{P}_k$.

$$\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{f \in F} \mathsf{cost}(f) \cdot y_f + \sum_{c \in A_k} \sum_{t \in T} t \cdot x_{c,t} & \\
\text{subject to} & x_{c,t} \geq z_{c,t} & \text{for all } c \in A_k,\ t \in T, \\
& y(F_{c,t}) \geq z_{c,t} & \text{for all } c \in A_k,\ t \in T, \\
& \displaystyle\sum_{t \in T} z_{c,t} \geq 1 & \text{for all } c \in A_k,
\end{array}$$

and non-negativity of all variables.

Serving Constraints. The LP constraints combined are equivalent to the set of the following (non-linear) requirements

$$\sum_{t \in T} \min\{x_{c,t},\ y(F_{c,t})\} \geq 1 \qquad \text{for all } c \in A_k. \tag{1}$$

We call (1) for client c the *serving constraint* for client c. In our description, we omit variables $z_{c,t}$ and the original constraints, ensuring only that the serving constraints hold and implicitly setting $z_{c,t} = \min\{x_{c,t},\ y(F_{c,t})\}$.

The LP above is indeed a valid relaxation of the FL problem. To see this, take any feasible integral solution. For any facility f opened in the integral solution, set variable y_f to 1. For each client c connected to facility f, set variable $x_{c,\tau}$ to 1, where $\tau = \mathsf{cost}(f, c)$. This guarantees that $\min\{x_{c,\tau},\ y(F_{c,\tau})\} = 1$, and thus the serving constraint (1) is satisfied for each client c.

Dual Program. The program $\mathcal{D}_k$ dual to $\mathcal{P}_k$ is

$$\begin{array}{lll}
\text{maximize} & \displaystyle\sum_{c \in A_k} \gamma_c & \\
\text{subject to} & \gamma_c \leq \alpha_{c,t} + \beta_{c,t} & \text{for all } c \in A_k,\ t \in T, \\
& \alpha_{c,t} \leq t & \text{for all } c \in A_k,\ t \in T, \\
& \displaystyle\sum_{c \in \mathsf{set}(f) \cap A_k} \beta_{c,\,\mathsf{cost}(f,c)} \leq \mathsf{cost}(f) & \text{for all } f \in F,
\end{array}$$

and non-negativity of all variables.

2.1 Overview

Our algorithm Frac creates a solution to $\mathcal{P}_k$, ensuring that the serving constraint (1) holds for all clients $c \in A_k$. As outlined in the introduction, the computed solution guarantees some additional properties that are useful for the rounding part later.

Whenever a client c arrives, Frac increases connection variables $x_{c,t}$ one by one starting from the smallest t, at the pace proportional to $1/t$. We ensure that $x_{c,t} \in [0, 1]$, i.e., once any of these variables reaches 1, Frac stops increasing them. A distance t, for which $x_{c,t} = 1$, is called *saturated.*

In parallel to manipulating variables $x_{c,t}$, Frac increases all variables y_f for facilities reachable from client c using saturated distances. The variables y_f are increased using the multiplicative update rule [5] (scaled appropriately to take costs of facilities into account).

Together with the solution to $\mathcal{P}_k$, Frac also constructs an *almost-feasible* solution to $\mathcal{D}_k$. That is, its solution to $\mathcal{D}_k$ is feasible when all dual variables are scaled down by a factor of $O(\log |F|)$. By the weak duality, the scaled-down value of this solution serves as a lower-bound for the optimum. Thus, as typical for the primal-dual type of analysis, the dual variables can be thought of as budgets whose increase balances the increase of primal variables.

2.2 Algorithm FRAC

At the very beginning, before any client arrives, Frac sets all variables y_f to 0 for all positive-cost facilities and to 1 for zero-cost ones. There are no other variables as the set A_0 of active clients is empty. Note that the dual program already contains the last type of constraints, but the sums on their left-hand sides range over empty sets of β variables, and hence these constraints are trivially satisfied.

Whenever a new client c arrives in step k, Frac updates the primal (dual) programs from $\mathcal{P}_{k-1}$ ($\mathcal{D}_{k-1}$) to $\mathcal{P}_k$ ($\mathcal{D}_k$), and then computes a feasible solution to $\mathcal{P}_k$ (based on the already created solution to $\mathcal{P}_{k-1}$) and a nearly-feasible solution to $\mathcal{D}_k$.

New variables in primal and dual programs: Frac sets $x_{c,t} \leftarrow 0$ for all $t \in T \setminus \{0\}$ and sets $x_{c,0} \leftarrow 1$. In the dual solution, it sets $\gamma_c \leftarrow 0$, $\alpha_{c,t} \leftarrow 0$ and $\beta_{c,t} \leftarrow 0$ for all $t \in T$.

Update primal program: A new serving constraint $\sum_{t \in T} \min\{x_{c,t}, y(F_{c,t})\} \geq 1$ appears in the primal program (and is violated unless $y(F_{c,0}) \geq 1$). As we never decrease primal variables, the serving constraints (1) that existed already in $\mathcal{P}_{k-1}$ are satisfied and will not become violated.

Update dual program: New constraints appear in the dual program and new variables $\beta_{c,t}$ appear on the left-hand side of the already existing inequalities. Since the new variables are initialized to 0, the validity of all dual constraints is unaffected.

Update primal and dual solutions: Let $T_c^1 = \{t \in T : x_{c,t} \geq 1\}$ be the set of saturated distances, i.e., initially Frac sets $T_c^1 \leftarrow \{0\}$. While the serving constraint for c is violated, Frac executes the *update operation* consisting of the following steps:

1. Set $\gamma_c \leftarrow \gamma_c + 1$.
2. For each $t \in T$, independently, adjust one dual variable: if $t \in T_c^1$, then set $\beta_{c,t} \leftarrow \beta_{c,t}+1$ and otherwise set $\alpha_{c,t} \leftarrow \alpha_{c,t} + 1$.
3. If $T_c^1 \subsetneq T$, choose *active distance* $t^* \leftarrow \min(T \setminus T_c^1)$ to be the smallest non-saturated distance, and then set $x_{c,t^*} \leftarrow x_{c,t^*} + 1/t^*$. (Note that $0 \in T_c^1$, and thus $t^* > 0$.)
4. For any facility $f \in \biguplus_{t \in T_c^1} F_{c,t}$, independently, perform *augmentation of* y_f, setting

$$y_f \leftarrow \left(1 + \frac{1}{\mathsf{cost}(f)}\right) \cdot y_f + \frac{1}{|F| \cdot \mathsf{cost}(f)}.$$

5. Update the set of saturated distances, setting $T_c^1 \leftarrow \{t \in T : x_{c,t} \geq 1\}$.

We now argue that if variable y_f is augmented in Step 4, then $\mathsf{cost}(f) > 0$ (i.e., Step 4 is well defined). Let $\tau = \mathsf{cost}(c, f)$. As y_f is augmented, the distance τ is saturated ($x_{c,\tau} = 1$). If $\mathsf{cost}(f) = 0$, then y_f would have been initialized to 1, and then $y(F_{c,\tau}) \geq 1$, in which case the serving constraint for c would be already satisfied.

Sidenote about T. For the sake of coherence and more streamlined analysis, Frac increases also connection variables $x_{c,t}$ to empty sets $F_{c,t}$, i.e., invests into distances to non-existing facilities. Fixing this overspending would not lead to asymptotic improvement of the performance.

2.3 Structural Properties

We focus on a single client c processed by Frac. We start with a property of connection variables $x_{c,t}$. The distances from T that are neither saturated nor active are called *inactive*. The following claim follows by an immediate induction on update operations performed by Frac.

▶ **Lemma 2.** *At all times when a client c is considered, $x_{c,t} \in [0,1]$ for any $t \in T$. In particular, $x_{c,t} = 1$ for any saturated distance $t \in T^1_c$. Furthermore,*

1. *either all distances are saturated,*
2. *or there exists an active distance $t^* > 0$, such that (i) all smaller distances are saturated, and (ii) all larger distances are inactive and the corresponding variables $x_{c,t}$ are equal to zero.*

Augmentation is performed on variables y_f corresponding to facilities whose distance from c is saturated.

▶ **Lemma 3.** *On any input $(G = (F, C, E, \mathsf{cost}), A)$, Frac returns a feasible solution and runs in time $\mathrm{poly}(|G|, |A|, \max_{e\in E} \mathsf{cost}(e), \max_{f\in F} \mathsf{cost}(f))$.*

Proof. Fix any client $c \in A$. By the definition of Frac, it takes t update operations to increase value $x_{c,t}$ from 0 to 1. Hence, after $\sum_{t\in T} t < 2\cdot\max_{e\in E} \mathsf{cost}(e)$ update operations, all connection variables are equal to 1. From that point on, all variables y_f for $f \in \biguplus_{t\in T} F_{c,t}$ are augmented in each update operation. Each variable y_f can be augmented at most $|F| \cdot \mathsf{cost}(f)$ times till it reaches or exceeds 1. That is, after at most $2\cdot\max_{e\in E} \mathsf{cost}(e)+|F|\cdot\max_{f\in F} \mathsf{cost}(f)$ update operations, the serving constraint is satisfied, i.e., the generated solution is feasible. ◀

The following lemma shows the crucial property of Frac. Namely for any client c, there exist a "good" distance τ, such that the collection of clusters $F_{c,t}$ at distance $t \le \tau$ is together fractionally half-open and that Frac invested $\Omega(\tau)$ into connecting client c. For any client c and distance $t \in T$, we define a set $S_{c,t}$ to be a collection of clusters alluded to in the introduction.

$$S_{c,t} = \biguplus_{t'\in T \,:\, t' \le t} F_{c,t'}$$

▶ **Lemma 4.** *Once Frac finishes serving client c, there exists a distance $\tau \in T$, such that $y(S_{c,\tau}) \ge 1/2$ and $\sum_{t\in T} t \cdot x_{c,t} \ge \tau/2$.*

Proof. We consider the state of variables once Frac finishes serving client c. Let $t^* > 0$ be the largest distance from T for which $x_{c,t^*} > 0$. As the serving constraint for client c is satisfied, we have

$$1 \le \sum_{t\in T} \min\{x_{c,t},\, y(F_{c,t})\} = \min\{x_{c,t^*},\, y(F_{c,t^*})\} + \sum_{t\in T \,:\, t<t^*} \min\{x_{c,t},\, y(F_{c,t})\}\,. \tag{2}$$

We pick τ depending on the value of the last term of (2).

If $\min\{x_{c,t^*},\, y(F_{c,t^*})\} \ge 1/2$, we set $\tau = t^*$. Then, $y(S_{c,\tau}) \ge y(F_{c,\tau}) \ge \min\{x_{c,\tau},\, y(F_{c,\tau})\} \ge 1/2$, and the first condition of the lemma follows. Furthermore, $\sum_{t\in T} t\cdot x_{c,t} \ge \tau\cdot x_{c,\tau} \ge \tau/2$.

Otherwise, $\min\{x_{c,t^*},\, y(F_{c,t^*})\} < 1/2$, and then, by (2), $\sum_{t\in T \,:\, t<t^*} \min\{x_{c,t},\, y(F_{c,t})\} \ge 1/2$. In such case, we choose τ as the largest distance from T smaller than t^*. Then

$$y(S_{c,\tau}) = \sum_{t\in T \,:\, t\le\tau} y(F_{c,t}) \ge \sum_{t\in T \,:\, t\le\tau} \min\{x_{c,t},\, y(F_{c,t})\} \ge 1/2,$$

i.e., the first condition of the lemma holds. By Lemma 2, either t^* is active at the end of processing c or all distances become saturated and t^* is the largest distance from T. In either case, $x_{c,t} = 1$ for any distance $t < t^*$, and thus in particular $x_{c,\tau} = 1$. Hence, the second part of the lemma holds as $\sum_{t \in T} t \cdot x_{c,t} \geq \tau \cdot x_{c,\tau} = \tau$. ◀

2.4 Dual Solution is Almost Feasible

Using primal-dual analysis, we may show that the generated dual solution violates each constraint at most by a factor of $O(\log |F|)$.

▶ **Lemma 5.** *For any facility f, FRAC augments y_f at most $O(\log |F|) \cdot \mathsf{cost}(f)$ times.*

Proof. First, we observe that variable y_f can be augmented only if prior to augmentation it is smaller than 1. To show that, observe that the augmentation of y_f occurs only when FRAC processes an active client $c \in \mathsf{set}(f)$. Let $\tau = \mathsf{cost}(f, c)$, i.e., $f \in F_{c,\tau}$. As FRAC augments y_f, the distance τ must be saturated, i.e., $x_{c,\tau} = 1$. On the other hand, the serving constraint (1) is not satisfied when y_f is augmented, and thus $\min\{x_{c,\tau}, y(F_{c,\tau})\} < 1$ which implies that y_f must be strictly smaller than 1.

In particular, if $\mathsf{cost}(f) = 0$, then y_f is set to 1 immediately at the beginning, and hence no augmentation of y_f is ever performed, and the lemma follows trivially. As all non-zero costs are at least 1, below we assume $\mathsf{cost}(f) \geq 1$.

During the first $\mathsf{cost}(f)$ augmentations, the value of y_f increases from 0 to at least $1/|F|$ (due to additive increases). Next, during the subsequent $\lceil \log_{1+1/\mathsf{cost}(f)} |F| \rceil$ augmentations, the value of y_f reaches at least 1 (due to multiplicative increases), and hence it will not be augmented further. In total, the number of augmentations is upper-bounded by $\mathsf{cost}(f) + \lceil \log_{1+1/\mathsf{cost}(f)} |F| \rceil = O(\log |F|) \cdot \mathsf{cost}(f)$. In the last relation, we used $\mathsf{cost}(f) \geq 1$. ◀

▶ **Lemma 6.** *FRAC violates each dual constraint at most by a factor of $O(\log |F|)$.*

Proof. We show the claim for all types of constraints in the dual program.

1. Each dual constraint $\gamma_c \leq \alpha_{c,t} + \beta_{c,t}$ always holds with equality as together with γ_c, for each $t \in T$, FRAC increments either $\alpha_{c,t}$ or $\beta_{c,t}$.
2. Consider a constraint $\alpha_{c,t} \leq t$. Initially $\alpha_{c,t} = 0$ when client c appears, and it is incremented in an update operation only if distance t is not saturated. Distances are processed from the smallest to the largest, and it takes exactly t' update operations for a distance $t' \in T$ to become saturated. Therefore, $\alpha_{c,t}$ can be incremented at most $\sum_{t' \in T : t' \leq t} t'$ times. If $t = 0$, then $\alpha_{c,t} = 0$ trivially. Otherwise, we use the fact that $T \setminus \{0\}$ contains only powers of 2, and hence $\alpha_{c,t} \leq \sum_{t' \in T : t' \leq t} t' < 2 \cdot t$.
3. Finally, fix any facility $f^* \in F$ and consider the constraint $\sum_{c \in \mathsf{set}(f^*) \cap A_k} \beta_{c, \mathsf{cost}(f^*, c)} \leq \mathsf{cost}(f^*)$. We want to show that this constraint is violated at most by a factor of $O(\log |F|)$, i.e., that
$$\sum_{c \in \mathsf{set}(f^*) \cap A_k} \beta_{c, \mathsf{cost}(f^*, c)} \leq O(\log |F|) \cdot \mathsf{cost}(f^*). \tag{3}$$
The left-hand side of (3) is initially 0 and it is incremented only when FRAC processes some active client $c^* \in \mathsf{set}(f^*)$. In a single update operation, FRAC may increment multiple β variables, but only one of them, namely $\beta_{c^*, \mathsf{cost}(f^*, c^*)}$, contributes to the growth of the left-hand side of (3). If variable $\beta_{c^*, \mathsf{cost}(f^*, c^*)}$ is incremented, it means that the distance $\tau = \mathsf{cost}(f^*, c^*)$ is already saturated, i.e., $\tau \in T^1_{c^*}$. Thus, in the same update operation, FRAC augments all variables y_f for $f \in \biguplus_{t \in T^1_{c^*}} F_{c^*, t}$. This set of facilities includes cluster $F_{c^*, \tau}$ and thus also facility f^*. By Lemma 5, the augmentation of y_{f^*} may happen at most $O(\log |F|) \cdot \mathsf{cost}(f^*)$ times, which implies our claim. ◀

2.5 Competitive Ratio of FRAC

Finally, we show that in each update operation the growth of the primal cost is at most constant times the growth of the dual cost. This will imply the competitive ratio of FRAC.

▶ **Lemma 7.** *For any step k, the value of the solution to $\mathcal{P}_k$ computed by* FRAC *is at most* 3 *times the value of its solution to $\mathcal{D}_k$.*

Proof. As the values of both solutions are initially zero, it suffices to analyze the growth of the primal and dual objectives for a single update operation. The value of the dual solution grows by 1 as γ_c is incremented only for the requested client c. Thus, it is sufficient to show that the primal solution increases at most by 3.

By y_f, $x_{c,t}$ and T_c^1, we understand the values of these variables before an update operation. Let $F_1 = \biguplus_{t \in T_c^1} F_{c,t}$. As the serving constraint for client c is not satisfied at that point,

$$1 > \sum_{t \in T} \min\{x_{c,t},\, y(F_{c,t})\} \geq \sum_{t \in T_c^1} \min\{x_{c,t},\, y(F_{c,t})\} \geq \sum_{t \in T_c^1} y(F_{c,t}) = y(F_1). \tag{4}$$

In the last inequality we used that (by Lemma 2), $T_c^1 = \{t \in T : x_{c,t} = 1\}$. The last equality follows as sets $F_{c,t}$ are disjoint for different t.

Within a single update operation, let $\Delta x_{c,t}$ and Δy_f be the increases of variables $x_{c,t}$ and y_f, respectively. By Lemma 2, FRAC increases one connection variable x_{c,t^*} for an active distance t^* (and no connection variable if there is no active distance) and performs augmentations of y_f for all $f \in F_1$. The increase of the primal value is then

$$\begin{aligned}\Delta \mathcal{P} &= \sum_{t \in T} t \cdot \Delta x_{c,t} + \sum_{f \in F_1} \mathsf{cost}(f) \cdot \Delta y_f \leq 1 + \sum_{f \in F_1} \mathsf{cost}(f) \cdot \left(\frac{y_f}{\mathsf{cost}(f)} + \frac{1}{|F| \cdot \mathsf{cost}(f)} \right) \\ &= 1 + y(F_1) + \frac{|F_1|}{|F|} < 3,\end{aligned}$$

where the last inequality follows by (4). ◀

▶ **Lemma 8.** *For any input $(G = (F, C, E, \mathsf{cost}), A)$, it holds that* $\text{FRAC}(G, A) \leq O(\log |F|) \cdot \text{OPT}(G, A)$.

Proof. Let k be the total number of active clients in A, and let $\mathsf{val}(\mathcal{P}_k)$ and $\mathsf{val}(\mathcal{D}_k)$ be the values of the final primal and dual solutions generated by FRAC. Then,

$$\begin{aligned}\text{FRAC}(G, A) = \mathsf{val}(\mathcal{P}_k) &\leq 3 \cdot \mathsf{val}(\mathcal{D}_k) && \text{(by Lemma 7)} \\ &\leq O(\log |F|) \cdot \text{OPT}(G, A) && \text{(by Lemma 6 and weak duality).}\end{aligned}$$ ◀

3 Deterministic Rounding

Now we define our deterministic algorithm INT, which rounds the fractional solution computed by FRAC. For a client $c \in A$, INT observes the actions of FRAC while processing c and on this basis makes its own decisions. First, INT processes augmentations of variables y_f performed by FRAC, and purchases some facilities. Once FRAC finishes handling client c, INT connects c to the closest open facility. (We show below that such facility exists.)

3.1 Purchasing Facilities: Properties of INTFAC

Purchasing facilities by Int is based solely on graph G and on updates of variables y_f produced by Frac. In particular, it neglects whether a given client is active or not. We use integral variables $\hat{y}_f \in \{0,1\}$ to denote whether Int opened facility f. Furthermore, for any set F' we use $\hat{y}(F')$ as a shorthand for $\sum_{f \in F'} \hat{y}_f$.

The following lemma is an adaptation of the deterministic rounding routine for the set cover problem by Alon et al. [3] and its proof is postponed to Subsection 3.3.

▶ **Lemma 9.** *Fix any input $(G = (F, C, E, \mathsf{cost}), A)$. Initially, $\hat{y}_f = y_f = 0$ for any $f \in F$. There exists a deterministic polynomial-time online algorithm IntFac that transforms increments of fractional variables y_f to increments of integral variables $\hat{y}_f \in \{0,1\}$, so that*

- *condition $y(S_{c,t}) \geq 1/2$ implies $\hat{y}(S_{c,t}) \geq 1$ for any client $c \in C$ (active or inactive) and any $t \in T$,*
- *$\sum_{f \in F} \mathsf{cost}(f) \cdot \hat{y}_f \leq O(\log |C \times T|) \cdot \sum_{f \in F} \mathsf{cost}(f) \cdot y_f + 2 \cdot \max_{f \in F} \mathsf{cost}(f)$.*

3.2 Connecting Clients

Once Int purchases facilities using deterministic routine IntFac (cf. Lemma 9), it connects client c to the closest open facility. Now we show that such a facility indeed exists and we bound the competitive ratio of Int.

▶ **Lemma 10.** *On any input (G, A), the solution generated by Int is feasible and the total cost of connecting clients by Int is at most $2 \cdot \textsc{Frac}(G, A)$.*

Proof. Fix any client $c \in A$. By Lemma 4, there exists a distance $\tau \in T$ such that $y(S_{c,\tau}) \geq 1/2$ and $\sum_{t \in T} t \cdot x_{c,t} \geq \tau/2$. By Lemma 9, once Int purchases facilities, it holds that $\hat{y}(S_{c,\tau}) \geq 1$. It means that at least one facility is opened in set $S_{c,\tau}$, i.e., at distance at most τ from c.

Therefore, Int is feasible and by connecting client c to the closest open facility, it ensures that the connection cost is at most $\tau \leq 2 \cdot \sum_{t \in T} t \cdot x_{c,t}$. The proof is concluded by observing that $\sum_{t \in T} t \cdot x_{c,t}$ is the connection cost of Frac that can be attributed solely to the connection of client c. ◀

▶ **Lemma 11.** *For any input $(G = (F, C, E, \mathsf{cost}), A)$, it holds that $\textsc{Int}(G, A) \leq q \cdot \log |F| \cdot (\log |C| + \log \log \Delta_G) \cdot \textsc{Opt}(G, A) + 2 \cdot \max_{f \in F} \mathsf{cost}(f)$, where q is a universal constant not depending on G or A. Furthermore, Int runs in time polynomial in $|G|$, $|A|$, $\max_{e \in E} \mathsf{cost}(e)$, and $\max_{f \in F} \mathsf{cost}(f)$.*

Proof. Let $\rho = \max_{f \in F} \mathsf{cost}(f)$. Then,

$$\begin{aligned} \textsc{Int}(G, A) &\leq \sum_{f \in F} \mathsf{cost}(f) \cdot \hat{y}_f + 2 \cdot \textsc{Frac}(G, A) && \text{(by Lemma 10)} \\ &\leq O(\log |C \times T|) \cdot \textsc{Frac}(G, A) + 2 \cdot \rho && \text{(by Lemma 9)} \\ &= O((\log |C| + \log |T|) \cdot \log |F|) \cdot \textsc{Opt}(G, A) + 2 \cdot \rho && \text{(by Lemma 8).} \end{aligned}$$

The bound on the cost of Int is concluded by using $|T| \leq 2 + \log \Delta_G$.

By Lemma 3, Frac running time is $\text{poly}(|G|, |A|, \max_{e \in E} \mathsf{cost}(e), \max_{f \in F} \mathsf{cost}(f))$. On top of that, Int adds its own computations (in particular the rounding scheme of IntFac), whose runtime is polynomial in $|G|$ and $|A|$. This implies the second part of the lemma (the running time of Int). ◀

3.3 Purchasing Facilities: Algorithm INTFAC

We start with a technical claim and later we define our rounding procedure IntFac.

▶ **Lemma 12.** *Fix any $q \in [0, 1/2]$ and any $r \geq 0$. Let X be a binary variable being 0 with probability $p > 0$. Then, $\mathbf{E}[\exp(q \cdot X)] \leq \exp(-(3/2) \cdot q \cdot \ln p)$.*

Proof. Using the definition of X, we have

$$\begin{aligned} \mathbf{E}[\exp(q \cdot X)] &= p \cdot \mathrm{e}^0 + (1-p) \cdot \mathrm{e}^q = \exp(\ln p) + (1 - \exp(\ln p)) \cdot \mathrm{e}^q \\ &\leq 1 + \ln p - e^q \cdot \ln p = 1 - \ln p \cdot (\mathrm{e}^q - 1) \\ &\leq 1 - (3/2) \cdot q \cdot \ln p \\ &\leq \exp(-(3/2) \cdot q \cdot \ln p). \end{aligned}$$

In the first inequality, we used that $e^x \cdot 1 + (1 - e^x) \cdot z \leq (1 + x) \cdot 1 + (-x) \cdot z$ for any $x \leq 0$ and $z \geq 1$ and in the second one, we used that $e^x - 1 \leq 3x/2$ for any $x \in [0, 1/2]$. ◀

Algorithm Description. As we mentioned earlier, our routine IntFac for rounding facilities is an adaptation of the deterministic rounding procedure for the set cover problem by Alon et al. [3]. On the basis of the facility-client graph G, we define the set $C \times T$ of *elements*. Intuitively, our solution Frac "covers" an element $(c, t) \in C \times T$ by fractionally opening facilities from $S_{c,t}$. The routine IntFac deterministically rounds these covering choices.

Let $\ell = |C \times T|$, $\rho = \max_{f \in F} \mathsf{cost}(F)$ and $b = 6 \cdot \ln \ell = O(\log |C \times T|)$. We consider the potential function $\Phi = \Phi_1 + \Phi_2$, where

$$\Phi_1 = \sum_{(c,t) \,:\, \hat{y}(S_{c,t}) = 0} \ell^{\,4 \cdot y(S_{c,t})} \qquad \text{and} \qquad \Phi_2 = \ell \cdot \exp\left(\sum_{f \in F} \frac{\mathsf{cost}(f)}{2\rho} \cdot (\hat{y}_f - b \cdot y_f) \right).$$

Assume that Frac augmented variable y_f. Then our algorithm IntFac chooses whether to set $\hat{y}_f$ to 1 or not (purchase f or not), so that the potential Φ does not increase. (We again emphasize that this choice neglects the current set of active clients.)

Correctness and Performance. In the lemma below, we show that IntFac is well defined, i.e., it is possible to fix variable $\hat{y}_f$, so that the potential Φ does not increase. This implies that both Φ_1 and Φ_2 remain upper-bounded, which can be in turn used to show properties of Lemma 9.

▶ **Lemma 13.** *Assume y_{f^*} is increased by δ. If $\hat{y}_{f^*} = 1$, then Φ does not increase. Otherwise, there is a choice to either set $\hat{y}_{f^*}$ to 1 or not, so that Φ does not increase.*

Proof. By y_f and $\hat{y}_f$, we mean the values of these variables before an update operation of Frac.

First, we assume $\hat{y}_{f^*} = 1$. Increasing variable y_{f^*} affects values of $y(S_{c,t})$ for $f^* \in S_{c,t}$: all such $y(S_{c,t})$ increase by δ. However, for any element (c, t), such that $f^* \in S_{c,t}$, it holds that $\hat{y}(S_{c,t}) \geq \hat{y}_{f*} = 1$, i.e., element (c, t) is not counted in the sum occurring in Φ_1. Thus, increasing variable y_{f^*} does not affect Φ_1. Furthermore, increasing y_{f^*} and keeping $\hat{y}_{f^*}$ unchanged can only decrease Φ_2. Thus, $\Phi = \Phi_1 + \Phi_2$ does not increase when $\hat{y}_{f^*} = 1$.

Second, we consider the case $\hat{y}_{f^*} = 0$. To show that either setting $\hat{y}_{f^*}$ to 1 or leaving it at 0 does not increase the potential, we use the probabilistic method and show that if we pick such action randomly (setting $\hat{y}_{f^*} = 1$ with probability $1 - \ell^{-4 \cdot \delta}$), then, in expectation, neither Φ_1 nor Φ_2 increases.

- As observed above, only elements (c,t) for which $S_{c,t}$ contain f^* are affected by the increase of y_{f^*} and possible change of $\hat{y}_{f^*}$. Let $Q = \{(c,t) : f^* \in S_{c,t} \text{ and } \hat{y}(S_{c,t}) = 0\}$ be the set of such elements contributing to Φ_1.
  Fix any element $(c,t) \in Q$. Its initial contribution towards Φ_1 is $\ell^{4 \cdot y(S_{c,t})}$ and when y_{f^*} increases, the contribution grows to $\ell^{4 \cdot (y(S_{c,t}) + \delta)}$. However, with probability $1 - \ell^{-4 \cdot \delta}$, variable $\hat{y}_{f^*}$ is set to 1, thus $\hat{y}(S_{c,t})$ grows from 0 to 1, and in effect element (c,t) stops contributing to Φ_1. Hence, the expected final contribution of element (c,t) towards Φ_1 is $\ell^{4 \cdot (y(S_{c,t}) + \delta)} \cdot \ell^{-4 \cdot \delta} + 0 \cdot (1 - \ell^{-4 \cdot \delta}) = \ell^{4 \cdot y(S_{c,t})}$, i.e., is equal to its initial contribution. Therefore, in expectation, the value of Φ_1 is unchanged.
- It remains to bound the expected value of Φ_2. Let $\hat{Y}$ be the random variable equal to the value of $\hat{y}_{f^*}$ after the random choice (i.e., $\hat{Y} = 1$ with probability $1 - \ell^{-4 \cdot \delta}$) and Φ'_2 denote the value of Φ_2 after increasing y_{f^*} and after the random choice. Using $y_{f^*} = 0$, we obtain

$$\begin{aligned}\Phi'_2 &= \ell \cdot \exp\left(\sum_{f \in F} \frac{\mathsf{cost}(f)}{2\rho} \cdot (\hat{y}_f - b \cdot y_f) + \frac{\mathsf{cost}(f^*)}{2\rho} \cdot \hat{Y} - \frac{b \cdot \mathsf{cost}(f^*)}{2\rho} \cdot \delta \right) \\ &= \Phi_2 \cdot \exp\left(\frac{\mathsf{cost}(f^*)}{2\rho} \cdot \hat{Y} \right) \cdot \exp\left(- \frac{b \cdot \mathsf{cost}(f^*)}{2\rho} \cdot \delta \right).\end{aligned}$$

  To estimate $\mathbf{E}[\Phi'_2]$, we upper-bound the expected value of expression $\exp(\hat{Y} \cdot \mathsf{cost}(f^*)/(2\rho))$, using Lemma 12 with $q = \mathsf{cost}(f^*)/(2\rho) \le 1/2$ and $p = \ell^{-4 \cdot \delta}$, obtaining that

$$\mathbf{E}\left[\exp\left(\frac{\mathsf{cost}(f^*)}{2\rho} \cdot \hat{Y} \right) \right] \le \exp\left(- \frac{(3/2) \cdot \mathsf{cost}(f^*)}{2\rho} \cdot \ln p \right) = \exp\left(\frac{6 \cdot \ln \ell \cdot \mathsf{cost}(f^*)}{2\rho} \cdot \delta \right).$$

  Therefore, $\mathbf{E}[\Phi'_2] \le \Phi_2$ and the lemma follows. ◀

Proof of Lemma 9. Initially, all variables y_f and $\hat{y}_f$ are zero, and thus $\Phi = \sum_{(c,t) \in C \times T} \ell^0 + \ell \cdot \exp(0) = 2 \cdot \ell$. By Lemma 13, the potential never increases. Since Φ_2 is non-negative, any summand of Φ_1 is always at most $2 \cdot \ell \le \ell^2$. Therefore, $4 \cdot y(S_{c,t}) \ge 2$ always implies $\hat{y}(S_{c,t}) > 0$, i.e., the first part of the lemma follows.

To show the second part, we again use that $\Phi = \Phi_1 + \Phi_2 \le 2 \cdot \ell$ at any time. As Φ_1 is non-negative, $\Phi_2 \le 2 \cdot \ell$. Substituting the definition of Φ_2, dividing by ℓ, and taking natural logarithm of both sides yields

$$\frac{1}{2\rho} \cdot \sum_{f \in F} (\hat{y}_f \cdot \mathsf{cost}(f) - b \cdot y_f \cdot \mathsf{cost}(f)) \le \ln(2) < 1.$$

Therefore, $\sum_{f \in F} \hat{y}_f \cdot \mathsf{cost}(f) \le 2\rho + b \cdot \sum_{f \in F} y_f \cdot \mathsf{cost}(f)$. ◀

4 Handling Large Aspect Ratios

The guarantee of Lemma 11 has two deficiencies: (i) the bound on the competitive ratio of Int depends on the aspect ratio of G and on the cost of the most expensive facility, (ii) the running time of Int depends on the maximal cost in graph G (which can be exponentially large in the input description). We show how to use cost doubling and edge pruning to handle these issues, creating our final deterministic solution Det and proving the main theorem (restated below).

▶ **Theorem 1.** *There exists a deterministic polynomial-time $O(\log |F| \cdot (\log |C| + \log \log |F|))$-competitive algorithm for the online non-metric facility location problem on set F of facilities and set C of clients.*

Proof. Fix facility-client graph $G = (F, C, E, \mathsf{cost})$ for the non-metric facility location problem. Recall that we assumed that all non-zero costs and distances in G are powers of 2 and are at least 1. Let $R = \log |F| \cdot (\log |C| + \log\log(|F| \cdot |C|))$.

We now construct a deterministic algorithm Det which is $O(R)$-competitive on an input (G, A). Let q be the constant from Lemma 11. Det operates in phases, numbered from 0. In phase j, it executes the following operations.

1. Det *pre-purchases* all facilities and edges of G whose cost is smaller than $2^j/(|F| \cdot |C|)$.
2. Det creates an auxiliary facility-client graph $\tilde{G}_j$ applying the following modifications to G.
   - First, Det creates graph G_j containing only edges and facilities from G whose individual cost is at most 2^j. It also removes connections to facilities that have been removed in this process.
   - Second, the costs of all facilities and edges that have been pre-purchased by Det are set to zero in G_j. In a result, G_j is a sub-graph of G with adjusted distances and costs of facilities, has the same set of clients, its set of facilities is a subset of F, and $\Delta_{G_j} \leq |F| \cdot |C|$.
   - Third, $\tilde{G}_j$ is the modified version of G_j, where all costs have been scaled down, so that the smallest positive cost is equal to 1. We denote the scaling factor by $h_j \leq 1$.
3. Det simulates algorithm Int on input $(\tilde{G}_j, A)$. That is, for a client $c \in A$, Det verifies whether the overall cost of Int (including serving c) remains at most $h_j \cdot (q \cdot R + 2) \cdot 2^j$. In such case, Det outputs the choices of Int for client c as its own. We emphasize that Int is run also on clients that have been already served in the previous phases; in effect, Det may purchase the same facilities or connections multiple times.
4. Eventually, either the sequence A of active clients ends and the total cost of Int on $(\tilde{G}_j, A)$ is at most $h_j \cdot (q \cdot R+2) \cdot 2^j$ (in which case Det terminates as well) or the purchases made by Int, while handling a client $c \in A$, caused its cost to exceed $h_j \cdot (q \cdot R + 2) \cdot 2^j$. (This includes the special case where c is disconnected from all facilities in $\tilde{G}_j$, because all edges incident to c in G were either more expensive than 2^j or were leading to facilities more expensive than 2^j.) In the case of exceeded cost, Det disregards the decisions of Int for client c, terminates Int, and starts phase $j + 1$, processing also all clients that were already served in phase j.

We now analyze the performance of Det. Let $k = \lceil \log(\mathrm{Opt}(G, A)) \rceil \geq 0$. We show that Det terminates latest in phase k. Assume that Det has not finished within phases $0, 1, \ldots, k-1$. In phase k, Det creates auxiliary graphs G_k and $\tilde{G}_k$, and runs Int on graph $\tilde{G}_k$. Graph G_k contains all edges of G of cost at most 2^k; their cost in G_k is the same or reset to zero. As $\mathrm{Opt}(G, A) \leq 2^k$, $\mathrm{Opt}(G, A)$ purchases only edges that are in G_k, and thus $\mathrm{Opt}(G, A)$ is also a feasible solution to instance (G_k, A). Thus, $\mathrm{Opt}(G_k, A) \leq \mathrm{Opt}(G, A) \leq 2^k$. As $\tilde{G}_k$ is the scaled-down copy of G_k, $\mathrm{Opt}(\tilde{G}_k, A) = h_k \cdot \mathrm{Opt}(G_k, A) \leq h_k \cdot 2^k$.

Let $\tilde{F}_k$ be the set of facilities of graph $\tilde{G}_k$ and $\tilde{\mathsf{cost}}_k(f)$ is the cost of opening facility f in graph $\tilde{G}_k$. Clearly, $|\tilde{F}_k| \leq |F|$ and $\tilde{\mathsf{cost}}_k(f) \leq h_k \cdot \mathsf{cost}(f)$ for any $f \in F$. By our construction, $\Delta_{\tilde{G}_k} = \Delta_{G_k} \leq |F| \cdot |C|$. Hence, Lemma 11 implies that

$$\begin{aligned}
\mathrm{Int}(\tilde{G}_k, A) &\leq q \cdot \log |F_k| \cdot \left(\log |C| + \log\log \Delta_{\tilde{G}_k}\right) \cdot \mathrm{Opt}(\tilde{G}_k, A) + 2 \cdot \max_{f \in \tilde{F}_k} \tilde{\mathsf{cost}}_k(f) \\
&\leq h_k \cdot q \cdot \log |F| \cdot (\log |C| + \log\log(|F| \cdot |C|)) \cdot 2^k + 2 \cdot h_k \cdot 2^k \\
&= h_k \cdot (q \cdot R + 2) \cdot 2^k.
\end{aligned}$$

Therefore, Int is not terminated prematurely within phase k because of high cost and it finishes the entire sequence A. This implies the feasibility of Int: it serves all clients latest in phase k.

To bound the total cost of Det, recall that at the beginning of phase j, Det purchases at most $|F|\cdot|C|$ edges and at most $|F|$ facilities, each of cost at most $2^j/(|F|\cdot|C|)$. The associated overall cost is at most $2\cdot 2^j$. The cost of the subsequent execution of algorithm Int on $\tilde{G}_j$ is, by our termination rule, at most $h_j \cdot (q \cdot R + 2) \cdot 2^j$, and thus the cost incurred by repeating Int's actions on G is at most $(q\cdot R+2)\cdot 2^j$. The overall cost is then $\text{Det}(G, A) \leq \sum_{j=0}^{k} (q\cdot R+4)\cdot 2^j = O(R) \cdot 2^k = O(R) \cdot \text{Opt}(G, A) = O(\log |F| \cdot (\log |C| + \log\log |F|)) \cdot \text{Opt}(G, A)$.

For the running time of Det, we note that in phase j, Int is run on a graph $\tilde{G}_j$ whose smallest cost is 1, and hence the largest cost is at most $\Delta_{\tilde{G}_j} = \Delta_{G_j} \leq |F| \cdot |C|$. Thus, by Lemma 11, the running time of Int in a single phase is polynomial in $|G|$ and $|A|$, and the number of phases is logarithmic in the maximum cost occurring in G, and thus also polynomial in $|G|$. ◀

5 Application to Online Node-Weighted Steiner Tree

Our result for the non-metric FL problem has an immediate application for the online node-weighted Steiner tree (NWST) problem, where the graph consists of ℓ nodes and an online algorithm is given k terminals to be connected. Namely, the randomized solution for the online NWST problem by Naor et al. [26] is in fact a deterministic polynomial-time “wrapper” around randomized routine solving the non-metric FL problem. To solve an instance of the NWST problem, their algorithm constructs a sub-instance of non-metric FL with $O(\ell)$ facilities, $O(\ell)$ potential clients, and $O(k)$ active clients. Such instance can be solved by the randomized algorithm of Alon et al. [2] with the competitive ratio of $O(\log k \cdot \log \ell)$. The wrapper adds another $O(\log k)$ factor in the ratio, resulting in an $O(\log^2 k \cdot \log \ell)$-competitive algorithm.

Our deterministic algorithm, when applied to this setting would be $O(\log^2 \ell)$-competitive on the constructed non-metric FL sub-instance. Therefore, by replacing the randomized algorithm by Alon et al. [2] with our deterministic one, we immediately obtain the first online deterministic solution for online NWST.

▶ **Corollary 14.** *There exists a polynomial-time deterministic online algorithm for the node-weighted Steiner tree problem, which is $O(\log k \cdot \log^2 \ell)$-competitive on graphs with ℓ nodes and k terminals.*

We note that the currently best solution for the node-weighted Steiner tree is randomized and achieves the ratio of $O(\log^2 \ell)$ [16, 15] and the best known lower bound for deterministic algorithms is $\Omega(\log \ell \cdot \log k/(\log\log \ell + \log\log k))$ [26, 3].

6 Final Remarks

We presented a deterministic solution to the non-metric facility location problem, whose performance nearly matches that of the best randomized one. By clustering facilities, we encoded dependencies between facilities and clients, which allowed us later to apply the rounding scheme to facilities only, neglecting the actual active clients. It would be however interesting and useful to have an online deterministic rounding routine able to handle such dependencies internally (e.g., by creating a pessimistic estimator that can be computed and handled in an online manner), as it is the case for the set cover problem or throughput-competitive virtual circuit routing [8].

That said, we believe that our distance clustering techniques can be extended to other network design problems for which only randomized algorithms existed so far, e.g., online multicast problems on trees [2], online group Steiner problem on trees [2], or variants of the facility location problem that are used as building blocks for solutions to other node-weighted Steiner problems [15, 16]. (For these problems there are no known direct reductions to the set cover problem). Finally, another open problem is whether these techniques could be also applied more directly for the node-weighted Steiner tree, resulting in a better deterministic competitive ratio.

References

1 Karen Aardal, Jaroslaw Byrka, and Mohammad Mahdian. Facility location. In *Encyclopedia of Algorithms*, pages 717–724. Springer, 2016. doi:10.1007/978-1-4939-2864-4_139.

2 Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. A general approach to online network optimization problems. *ACM Transactions on Algorithms*, 2(4):640–660, 2006. doi:10.1145/1198513.1198522.

3 Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The online set cover problem. *SIAM Journal on Computing*, 39(2):361–370, 2009. doi:10.1137/060661946.

4 Aris Anagnostopoulos, Russell Bent, Eli Upfal, and Pascal Van Hentenryck. A simple and deterministic competitive algorithm for online facility location. *Information and Computation*, 194(2):175–202, 2004. doi:10.1016/j.ic.2004.06.002.

5 Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing Systems*, 8(1):121–164, 2012. doi:10.4086/toc.2012.v008a006.

6 Baruch Awerbuch, Yossi Azar, and Serge A. Plotkin. Throughput-competitive on-line routing. In *Proc. 34th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 32–40, 1993. doi:10.1109/SFCS.1993.366884.

7 Niv Buchbinder and Joseph Naor. The design of competitive online algorithms via a primal-dual approach. *Foundations and Trends in Theoretical Computer Science*, 3(2–3):93–263, 2009. doi:10.1561/0400000024.

8 Niv Buchbinder and Joseph Naor. Online primal-dual algorithms for covering and packing. *Mathematics of Operations Research*, 34(2):270–286, 2009. doi:10.1287/moor.1080.0363.

9 Jaroslaw Byrka and Karen Aardal. An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. *SIAM Journal on Computing*, 39(6):2212–2231, 2010. doi:10.1137/070708901.

10 Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for facility location problems. *SIAM Journal on Computing*, 34(4):803–824, 2005. doi:10.1137/S0097539701398594.

11 Fabián A. Chudak and David B. Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. *SIAM Journal on Computing*, 33(1):1–25, 2003. doi:10.1137/S0097539703405754.

12 Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

13 Dimitris Fotakis. A primal-dual algorithm for online non-uniform facility location. *Journal of Discrete Algorithms*, 5(1):141–148, 2007. doi:10.1016/j.jda.2006.03.001.

14 Dimitris Fotakis. On the competitive ratio for online facility location. *Algorithmica*, 50(1):1–57, 2008. doi:10.1007/s00453-007-9049-y.

15 MohammadTaghi Hajiaghayi, Vahid Liaghat, and Debmalya Panigrahi. Near-optimal online algorithms for prize-collecting steiner problems. In *Proc. 41st Int. Colloq. on Automata, Languages and Programming (ICALP)*, pages 576–587, 2014. doi:10.1007/978-3-662-43948-7_48.

16 MohammadTaghi Hajiaghayi, Vahid Liaghat, and Debmalya Panigrahi. Online node-weighted steiner forest and extensions via disk paintings. *SIAM Journal on Computing*, 46(3):911–935, 2017. doi:10.1137/14098692X.

17 Dorit S. Hochbaum. Heuristics for the fixed cost median problem. *Mathematical Programming*, 22(1):148–162, 1982. doi:10.1007/BF01581035.

18 Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM*, 50(6):795–824, 2003. doi:10.1145/950620.950621.

19 Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In *Proc. 34th ACM Symp. on Theory of Computing (STOC)*, pages 731–740, 2002. doi:10.1145/509907.510012.

20 Antoon Kolen and Arie Tamir. Covering problems. In P.B. Mirchandani and R.L. Francis, editors, *Discrete Location Theory*, Wiley Series in Discrete Mathematics and Optimization. Wiley, 1990.

21 Simon Korman. On the use of randomization in the online set cover problem. Master's thesis, The Weizmann Institute of Science, 2004.

22 Madhukar R. Korupolu, C. Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. *Journal of Algorithms*, 37(1):146–188, 2000. doi:10.1006/jagm.2000.1100.

23 Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Information and Computation*, 222:45–58, 2013. doi:10.1016/j.ic.2012.01.007.

24 Mohammad Mahdian, Yinyu Ye, and Jiawei Zhang. Approximation algorithms for metric facility location problems. *SIAM Journal on Computing*, 36(2):411–432, 2006. doi:10.1137/S0097539703435716.

25 Adam Meyerson. Online facility location. In *Proc. 42nd IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 426–431, 2001. doi:10.1109/SFCS.2001.959917.

26 Joseph Naor, Debmalya Panigrahi, and Mohit Singh. Online node-weighted steiner tree and related problems. In *Proc. 52nd IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 210–219, 2011. doi:10.1109/FOCS.2011.65.

27 Prabhakar Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130–143, 1988. doi:10.1016/0022-0000(88)90003-7.

28 David B. Shmoys. Approximation algorithms for facility location problems. In *Proc. 3rd Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 27–33, 2000. doi:10.1007/3-540-44436-X_4.

29 David B. Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems (extended abstract). In *Proc. 29th ACM Symp. on Theory of Computing (STOC)*, pages 265–274, 1997. doi:10.1145/258533.258600.

30 Neal E. Young. Randomized rounding without solving the linear program. In *Proc. 6th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 170–178, 1995.

An Asymptotically Fast Polynomial Space Algorithm for Hamiltonicity Detection in Sparse Directed Graphs

Andreas Björklund
Private, Lund, Sweden [1]

Abstract

We present a polynomial space Monte Carlo algorithm that given a directed graph on n vertices and average outdegree δ, detects if the graph has a Hamiltonian cycle in $2^{n-\Omega(\frac{n}{\delta})}$ time. This asymptotic scaling of the savings in the running time matches the fastest known exponential space algorithm by Björklund and Williams ICALP 2019. By comparison, the previously best polynomial space algorithm by Kowalik and Majewski IPEC 2020 guarantees a $2^{n-\Omega(\frac{n}{2^\delta})}$ time bound.

Our algorithm combines for the first time the idea of obtaining a fingerprint of the presence of a Hamiltonian cycle through an inclusion–exclusion summation over the Laplacian of the graph from Björklund, Kaski, and Koutis ICALP 2017, with the idea of sieving for the non-zero terms in an inclusion–exclusion summation by listing solutions to systems of linear equations over $\mathbb{Z}_2$ from Björklund and Husfeldt FOCS 2013.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Hamiltonian cycle, directed graph, worst case analysis algorithm

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.15

Related Version *Previous Version*: `https://arxiv.org/abs/2009.11780`

Acknowledgements We are very grateful to an anonymous reviewer who pointed out a serious flaw in an earlier proof attempt of Lemma 6. We also thank Łukasz Kowalik for comments on an earlier version of the paper.

1 Introduction

Given a directed graph $G = (V, A)$ on $n = |V|$ vertices, we consider the problem of detecting if G has a Hamiltonian cycle, a directed cycle through G using a subset of the arcs A, visiting each vertex of V exactly once. We call this the *Hamiltonicity* problem. Deciding Hamiltonicity in a directed graph is one of Karp's original NP-complete problems [15]. For a very long time, the best worst case algorithm known for this problem was based on Bellman's [3] and Held and Karp's [13] dynamic programming across all vertex subsets from the early 1960's running in $2^n \operatorname{poly}(n)$ time. Much of recent research has focused on when one can improve over $O^*(2^n)$ time, confer the related work section below. A recent result by Björklund and Williams [11], building on Björklund, Kaski, and Koutis [9], describes a deterministic, exponential space, $2^{n-\Omega(\frac{n}{\delta})}$ time algorithm that counts the number of Hamiltonian cycles where $\delta = \frac{|A|}{n}$ is the average outdegree. A natural follow-up question is whether this speedup intrinsically comes at the cost of exponential space usage. In this paper, we give a partial negative answer to that question. We show that this requirement of an exponentially sized space resource can be reduced to a polynomially sized one, when we are only interested in detecting if the graph has a Hamiltonian cycle, and are content with a randomised algorithm. We prove

▶ **Theorem 1.** *There is a polynomial space Monte Carlo algorithm that given an n-vertex directed graph of average outdegree δ, detects w.h.p. if the graph has a Hamiltonian cycle in $2^{n-\Omega(\frac{n}{\delta})}$ time, without any false positives.*

[1] This work was done while being employed by Ericsson Research.

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 15; pp. 15:1–15:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Our algorithm builds on the algorithms by Björklund, Kaski, and Koutis [9] and its successor by Björklund and Williams [11]. At the core of the algorithms in [9] and [11], are efficient methods to list contributing terms to a sum evaluating to the number of Hamiltonian cycles. They use split, tabulate, and list procedures that seem to require exponential space. In more detail, in [11], they reduce the problem to listing pairs of dissimilar vectors from two exponential size sets of short vectors, where dissimilar means different in each coordinate. They further present two efficient algorithms to solve this latter problem that build on tabulation, one explicitly on one of the two sets, and the other indirectly as subresults of a large fast matrix multiplication. Our overall algorithm use a similar idea of listing contributing terms to a sum, but we use a different approach of obtaining the terms. In particular, we do not directly reduce to the problem of listing dissimilar vectors. Our main insight is that another technique previously used to compute the parity of the number of Hamiltonian cycles by Björklund and Husfeldt [7], by a careful design, can replace the tabulation for enumeration of solutions to a linear equation system over $\mathbb{Z}_2$. This latter task is well-known to be possible to do in polynomial space. Our way of combining the above two techniques is our main technical novelty.

Polynomial space algorithms improving over $O^*(2^n)$ time in sparse graphs were known before. We note that for the easier case of everywhere sparse graphs, i.e., graphs in which the sum of the in- and outdegree at every vertex is bounded by d, Björklund *et al.* [8] implicitly showed that you can decide Hamiltonicity in $2^{n-\Omega(\frac{dn}{2^d})}$ time using polynomial space. Their paper considered TSP in undirected graphs, but it is not difficult to see that their proof of Theorem 1.3 could also be used for directed Hamiltonicity. Very recently, Kowalik and Majewski [18] presented a polynomial space, $2^{n-\Omega(\frac{n}{2^\delta})}$ time algorithm for directed Hamiltonicity on n-vertex graphs of average outdegree δ. It builds on the algorithm by Björklund [5] which is a $(2-2^{1-\delta})^{n/2}\,\mathrm{poly}(n)$ time polynomial space algorithm in undirected bipartite graphs of average degree δ using techniques similar in spirit to our algorithm design here. Note though that in our design, the speedup is exponential in n/δ, whereas the speedup in Kowalik and Majewski [18] is exponential in $n/2^\delta$.

It should be noted in passing, that for many hard combinatorial problems the best known worst case algorithms use exponential space. In fact, in some cases the only known algorithms that improve over a straight-forward brute-force algorithm testing all possibilities use exponential space, are deterministic, and are also able to count the solutions. To give just one example that is also on Karp's list [15], this holds presently for MaxCut: compute a bipartition of the vertices that maximises the number of edges between the two parts. It has a $O^*(1.731^n)$ time counting algorithm where n is the number of vertices [21], but no polynomial space algorithm improving substantially over the brute-force $O^*(2^n)$ time algorithm is known, even if we only consider detection and use randomisation.

One reason to get rid of exponential space usage from a practical point of view, is to offer better implementability on parallel computing devices. This seems to be the case also with the present algorithm compared to the exponential space algorithms in [11]. In particular, the computationally heavy steps in our algorithm can easily be scheduled to compute different parts of the sum that may run obliviously of each other on different processors, only adding up their final partial sums in the end.

1.1 Related work

The fastest known (exponential space) algorithm for directed Hamiltonicity as far as we know is the $2^{n-\Omega(\sqrt{n/\log\log n})}$ time algorithm by Björklund, Kaski, and Williams [10]. The fastest known polynomial space algorithm is the $2^n \operatorname{poly}(n)$ time one based on counting closed walks via adjacency matrix powering and inclusion–exclusion, discovered at least four times [17, 16, 2, 1], the oldest by Kohn, Gottlieb, and Kohn [17] dates back to 1977. Much faster algorithms exist for special cases, also apart from the $2^{n-\Omega(\frac{n}{\delta})}$ time exponential space algorithm in average outdegree δ directed graphs by Björklund and Williams [11]. In *bipartite* directed graphs, there is a $O^*(1.732^n)$ time, polynomial space, algorithm by Björklund, Kaski, and Koutis [9]. There is also an earlier $O^*(1.888^n)$ time, exponential space, algorithm by Cygan, Kratsch, and Nederlof [12] based on a different technique. Björklund and Husfeldt [7] show a $O^*(1.619^n)$ time, polynomial space, algorithm that computes the *parity* of the number of Hamiltonian cycles in a directed graph. Somewhat perplexingly, this algorithm does not seem to be useful for the detection problem in general. However, counting modulo powers of small primes can be used for detection when the number of Hamiltonian cycles are less than c^n for some constant c: Björklund, Kaski, and Koutis [9], improving over a partial result in Björklund, Dell, and Husfeldt [6], show how to find a Hamiltonian cycle in $O((2-\epsilon_c)^n)$ time, with $\epsilon_c > 0$ being another constant depending only on c. In undirected graphs, there is a $O^*(1.657^n)$ time, polynomial space, algorithm, and in bipartite undirected graphs, there is a $O^*(1.415^n)$ time algorithm, both by Björklund [4]. Despite the partial positive results above, it is a major open question in the area of exact exponential time algorithms, whether or not a $O(c^n)$ time algorithm for any $c < 2$ exists for detecting Hamiltonian cycles in general directed graphs.

1.2 Methodology

Our algorithm is based on algebraic fingerprinting for Hamiltonian cycles, following a long line of works [4, 7, 6, 9, 5, 11]. The idea is to define a multivariate polynomial P over a ring, along with an efficient algorithm for its evaluation, with the property that P is non-zero only if the graph has a Hamiltonian cycle. That polynomial can then be used to detect Hamiltonicity, by testing if P is identically zero by evaluating P at a random point using the efficient algorithm (Polynomial identity testing, PIT). The choice of ring is important and a somewhat delicate matter. The basic observation is that using a larger ring increases the chance of making P non-zero on many points, whereas a smaller ring typically makes it easier to come up with an efficient evaluation algorithm. We will use a large ring for the polynomial, but our sample space will only take values from a small subring on a large subset of the variables. Our algorithm for evaluating P follows a construction by [9] based on an exponential sum of weighted Laplacians of the graph, that in itself already describes a $2^n \operatorname{poly}(n)$ time algorithm. To get a running time below that, we take measures in designing our sample space so that many summands will be zero for a trivial reason, and we can find out which are not by solving a linear equation system over $\mathbb{Z}_2$. This is inspired by the algorithm in [7] that lists solutions to a quadratic equation system over $\mathbb{Z}_2$ to sieve for the contributing terms. We list the solutions to a linear equation system by generating one solution from a Gaussian elimination followed by taking linear combinations of that solution with the null space (also found by the Gaussian elimination). This way we can list a superset of the summands that are non-zero and compute the sum to obtain the value of P at our random point.

2 The Algorithm

2.1 The Hamiltonicity Polynomial

We begin by describing the polynomial P we will be using. Following [11], we will work on a slightly modified version $G = (V, A)$ of the n-vertex input graph $G_{\text{in}} = (V_{\text{in}}, A_{\text{in}})$. We pick an arbitrary vertex $u \in V_{\text{in}}$, and replace u with two new vertices s and t, where s retains all outgoing arcs from u, and t retains all incoming arcs to u. Note that the Hamiltonian paths from s to t in this modified G are in one-to-one correspondence with the Hamiltonian cycles in the original graph G_{in}, and that the average degree is not increased. In the following, we consider the problem of detecting a s-t Hamiltonian path in the modified $n + 1$ vertex graph G.

Fix a (commutative) ring R, and introduce a variable $z_{uv} \in R$ for each arc $uv \in A$. Let $\mathcal{H}(G)$ be the set of Hamiltonian s-t paths in G, and consider the *Hamiltonicity polynomial* P_G as

$$P_G(z) = \sum_{H \in \mathcal{H}(G)} \prod_{uv \in H} z_{uv}. \tag{1}$$

Our algorithm is based on an efficient way of evaluating $P_G(z)$ in a carefully chosen random point z over a particular ring. Note that we will write z_{uv} in formulas to refer both to the formal variable and its value in R according to a specific assignment $z : A \to R$. In our analysis we will sometimes think of $P_G(z)$ as a formal polynomial in z with coefficients from R, but in the algorithm itself, we always mean $P_G(z)$ to be an evaluation over R of the polynomial P_G in a specific point z.

For now, note that an evaluation of $P_G(z)$ in a random point z can potentially be used as a fingerprint of existence of a Hamiltonian cycle in the original input graph: On one hand, the polynomial always evaluates to zero if the input graph has no Hamiltonian cycles (thus there are no false positives). On the other hand, by evaluating it in a random point, we will obtain a non-zero result if there is a Hamiltonian cycle in the input graph G_{in}, unless we are unlucky and the monomials happen to cancel each other. As mentioned above, there are two conflicting aspects to consider for a successful fingerprint design:

1. We want the ring and the sample space to be large enough so we can argue that the result is non-zero w.h.p. if the graph G_{in} is Hamiltonian.
2. We want the ring and the sample space to have some structure that we can use to derive an efficient evaluation algorithm.

The rest of our paper describes one way of balancing these aspects without having to resort to exponential size tabulation to enable a fast evaluation algorithm, by combining the graph Laplacian machinery in Björklund, Kaski, and Koutis [9] with the linear equation system modulo two listing idea from Björklund and Husfeldt [7]. In Section 2.3 we will address the first aspect of balancing the fingerprint which is the major novel part. In Sections 2.4 and 2.5 we describe the basis of the algorithm in [9] (and subsequently in [11]) that we will use, and in 2.6 and 2.7 we will address the second aspect of balancing the fingerprint by describing a linear algebra enumeration algorithm inspired by the algorithm in [7]. Finally, in Section 2.8 we put the parts together into an algorithm for Theorem 1. We begin by defining the ring.

2.2 The Choice of Ring

We will work over the polynomial ring $R = \mathbb{Z}_{2^k}[x]/(x^m)$, i.e., polynomials in one variable truncated at degree m with integer coefficients counted modulo 2^k. With foresight, both $k = k_R$ and $m = m_R$ will be poly(n), and hence an element in R is described by poly(n) bits, and the arithmetic operations of addition and multiplication can both be done in poly(n) time. To compute a determinant of a matrix in $R^{n\times n}$, as we will need later, we may use Kaltofen's division-free algorithm [14], that uses $O(n^{3.5}\log n\log\log n)$ ring operations. Altogether, the computation of the determinant of an $n\times n$ matrix over the ring R is a poly(n) time task.

2.3 The Sample Space

In this section, we describe the sample space over which we choose our point z for polynomial identity testing. We will also argue that a randomly chosen point from the sample space has $P_G(z)\neq 0$ with non-zero constant probability when the graph G has a Hamiltonian s-t path. As we primarily are interested in the asymptotic form of the running time scaling, we will set the parameters somewhat arbitrarily for ease of calculations. The sample space is parameterised by two positive integers τ and ℓ to be defined later in our analysis. The process to choose the point z is given below:

Procedure 1 SamplePoint (Returns a set T and an assignment z to be used for PIT of P_G).

```
1. Sample a subset $T \subseteq V \setminus \{s\}$ of size $\tau$ uniformly at random.
2. For every arc $uv, v \in T$, set $z_{uv} = 1$.
3. For every arc $uv, v \notin T$, set $z_{uv} = x^{w(uv)}$, where $w(uv) \in \{1, \cdots, \ell\}$ is a uniformly and
   independently randomly chosen integer.
```

We next turn to proving that the choice of z is good for PIT of P_G. We will first look at the Hamiltonicity polynomial after assigning $z_{uv}=1$ for all $v\in T$, but for now still treat all other z-variables unassigned (left as formal variables). We call these remaining variables $\tilde{z}$ with $\tilde{z}_{uv}=z_{uv}$ and consider the associated T-truncated polynomial $P_{G,T}(\tilde{z})$ obtained from P_G after the variable substitution. Define $\mathcal{H}_T(G)$ as the arc subsets of Hamiltonian s-t paths in $\mathcal{H}(G)$ after the removal of any arc ending in T, i.e.,

$$\mathcal{H}_T(G)=\{\cup_{uv\in H,v\notin T}uv : H\in\mathcal{H}(G)\}.$$

We call these the T-truncated Hamiltonian paths. We can write

$$P_{G,T}(\tilde{z})=\sum_{H'\in\mathcal{H}_T(G)}e_{H'}\cdot\prod_{uv\in H'}\tilde{z}_{uv},$$

where $e_{H'}$ counts the number of Hamiltonian cycles in G with T-truncation H'.

We first need to prove that $P_{G,T}(\tilde{z})$ is not the zero-polynomial with high enough probability, when G_{in} has a Hamiltonian cycle. Note that it may equal the zero-polynomial even in the presence of Hamiltonian cycles in G_{in}, when $e_{H'}$ is a multiple of 2^k for all $H'\in\mathcal{H}_T(G)$, as this would result in an annihilation in our ring R, where $k=k_R$ is the ring parameter in Section 2.2. We will first argue that this doesn't happen with too large a probability.

To prove this will not happen with some non-zero constant probability, let H be *any* fixed Hamiltonian s-t path in G. In particular, our arbitrary choice of H is independent of T. Let $H^T\in\mathcal{H}_T(G)$ be the T-truncation of H. We will upper bound the expectation of e_{H^T}, the number of Hamiltonian s-t paths in G whose T-truncation matches H^T. For every T, define $S=S(T,H)\subseteq V$ to be the set of in-neighbors of T along H, i.e.,

$$S=\{u : v\in T, uv\in H\}. \tag{2}$$

Note that S and T are not necessarily disjoint, but of the same size τ. We consider the following bipartite graph B_H obtained from an induced subgraph of G as follows. B_H has two parts, one representing the vertices in S, and one representing the vertices in T. Every arc in B_H connects a vertex in the first part representing S to a vertex in the second part representing T. There is an arc from a vertex u in the first part to a vertex v in the second part, iff uv is an arc in the induced subgraph $G[S \cup T]$. The following lemma tells us that the graph B_H in expectation is not too dense.

▶ **Lemma 2.** *The expected number of arcs in B_H is not larger than*

$$|S|\left(1+|T|\frac{\delta}{n}\right).$$

Proof. Let d_v denote the outdegree of vertex $v \in V$. Consider a vertex $u \in S$. The arc from u to the next vertex on H is always present in B_H. The number of other arcs though, is in expectation $(d_u - 1)\frac{|T|-1}{n}$ since the other vertices on T apart from u's out-neighbor on H are uniformly distributed. Hence, by the linearity of expectation, using that each vertex in $V \setminus \{t\}$ is included in S with probability $|S|/n$, the expected number of arcs in B_H is

$$\sum_{u \in V\setminus\{t\}} \frac{|S|}{n}\left(1+(d_u-1)\frac{|T|-1}{n}\right) \leq |S|\left(1+|T|\frac{\delta}{n}\right). \qquad ◀$$

This means that if we choose the fixed size $\tau = |T| = |S| = \frac{n}{c\delta}$, for some $c > 1$ to be set later, we get expected average outdegree from the vertices in the part representing S in B_H bounded by $1 + c^{-1}$. By Markov's inequality for a non-negative random variable X,

$$\Pr[X \geq \lambda \mathbb{E}[X]] \leq \frac{1}{\lambda},$$

we can bound the probability that the average degree is not much larger:

▶ **Corollary 3.** *The probability that the average outdegree of a vertex in S in B_H is at most*

$$\left(1+\frac{1}{49}\right)\left(1+\frac{1}{c}\right),$$

is at least $1/50$.

We next observe that all Hamiltonian s-t paths whose T-truncation is H^T defines the same set S. This also means that every Hamiltonian s-t path in G whose T-truncation is H^T must use some arc for each vertex in S in B_H. The product of the outdegrees of vertices in S is an upper bound on their number e_{H^T}. Hence, by the above corollary with probability at least $1/50$ there will be at most $((1+1/49)(1+c^{-1}))^\tau$ of them by the arithmetic mean-geometric mean inequality. By setting k_R in Section 2.2 large enough so that

$$2^{k_R} > \left(\left(1+\frac{1}{49}\right)\left(1+\frac{1}{c}\right)\right)^\tau,$$

we will get a monomial with non-zero coefficient in $P_{G,T}(\tilde{z})$ with probability at least $1/50$. We note that it suffices to set

$$k_R > \tau \log_2\left(\left(1+\frac{1}{49}\right)\left(1+\frac{1}{c}\right)\right) = \frac{n}{c\delta}\log_2\left(\left(1+\frac{1}{49}\right)\left(1+\frac{1}{c}\right)\right).$$

We will need k_R to be much smaller than τ in the evaluation algorithm described in the next sections in order to evaluate $P_G(z)$ fast. We will use this in Section 2.6 to prove Lemma 8.

Setting $c = 20$, say, we can thus use $k_R = \frac{\tau}{10}$ and conclude that $e_{H^T} < 2^{k_R}$ with large enough probability, and hence that $P_{G,T}(\tilde{z})$ has at least one monomial. To summarise, we have that

▶ **Lemma 4.** *With $\tau = \frac{n}{20\delta}$ and $k_R = \frac{n}{200\delta}$ (i.e., the parameters set as above),* **SamplePoint** *returns T so that the formal polynomial*

$$P_{G,T}(\tilde{z}) \neq 0,$$

when the input graph has a Hamiltonian cycle, with probability at least $1/50$.

We next turn to arguing that $P_G(z) \neq 0$ (over the ring R) with high enough probability for the assignment z returned by **SamplePoint**. The next famous lemma by Mulmuley, Vazirani, and Vazirani [19] shows that with ℓ large enough, we will be able to isolate a T-truncated Hamiltonian path in $\mathcal{H}_T(G)$ that is represented by a monomial in $P_{G,T}(\tilde{z})$.

▶ **Lemma 5** (Isolation Lemma, Mulmuley, Vazirani, and Vazirani [19])**.** *Let $m < M$ be two positive integers and let $\mathcal{F}$ be a nonempty family of subsets of $\{1, \cdots, m\}$. Suppose each element $x \in \{1, \cdots, m\}$ receives a weight $w(x) \in \{1, \cdots, M\}$ independently and uniformly at random. Define the weight of a set S in $\mathcal{F}$ as $w(S) = \sum_{x \in S} w(x)$. Then, with probability at least $1 - \frac{m}{M}$, there is a unique set in $\mathcal{F}$ of minimum weight.*

We apply the above lemma, on the family $\mathcal{F}$ equal to $\mathcal{H}_T(G)$ *further restricted* to those T-truncated Hamiltonian paths that are represented by a monomial in $P_{G,T}(\tilde{z})$, i.e., those H' that have $2^{k_R} \nmid e_{H'}$. We use the weights $w(uv)$ set as in step 3 of **SamplePoint** above to obtain z, with $\ell = 100|A|$, where A is the set of arcs in G. We have with probability at least $1 - 1/100$ that a monomial exists with some unique weight μ. In particular, with high enough probability, there is a Hamiltonian s-t path whose T-truncation H' is in $\mathcal{F}$ that will be isolated and contribute the value $e_{H'}x^{\mu}$ to $P_G(z)$.

By setting the ring parameter $m_R > n\ell$ in Section 2.2, we observe that this monomial in the polynomial ring is possible to detect.

▶ **Lemma 6.** *With $\tau = \frac{n}{20\delta}$,$k_R = \frac{n}{200\delta}$, $m_R > n\ell$, and $\ell = 100|A|$ (i.e., all the parameters set as above),* **SamplePoint** *returns z so that the polynomial*

$$P_G(z) \neq 0,$$

when the input graph has a Hamiltonian cycle, with probability at least $1/100$.

The probability bound comes from the probability of T being a good choice in Lemma 4 to guarantee that there exists a H' with $e_{H'} < 2^{k_R}$ (1/50) after subtracting the probability that the Isolation Lemma was not successful in its isolation (1/100).

2.4 The Laplacian

Björklund, Kaski, and Koutis [9] observed that the number of Hamiltonian cycles in a directed graph can be evaluated as an inclusion–exclusion summation over a determinant of a polynomial matrix representing the graph. We will use their construction, not over the integers, but over the particular ring R defined in Section 2.2. This means we will lose the ability to count the Hamiltonian cycles, but it will also enable a faster evaluation of the inclusion–exclusion formula as we will demonstrate. We reiterate their construction here for the sake of completeness and easy reference.

The weighted Laplacian of the graph G, is a $(n+1) \times (n+1)$ polynomial matrix $L = L_G(y, z)$ with rows and columns indexed by the vertices V, in the variables y_v for $v \in V \setminus \{t\}$, and variables z_{uv} for $uv \in A$:

$$L_{u,v} = \begin{cases} \sum_{wv \in A} z_{wv} y_w & \text{if } u = v \\ -z_{uv} y_u & \text{if } uv \in A \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

The Laplacian *punctured at the start vertex s*, is the matrix L_s obtained by removing row and column s from L. In [9](their Theorem 5), it was observed that Tutte's directed version of the Matrix-Tree theorem of Kirchhoff [20], where $\det(L_s)$ is a polynomial in which each term corresponds to a directed spanning out-branching rooted at s, could be used to compute the Hamiltonicity polynomial. By the principle of inclusion–exclusion, letting $|y|$ denote the number of vertices v for which $y_v = 1$, we have

▶ **Lemma 7** (Paraphrasing Equation (7) in Björklund, Kaski, and Koutis [9]).

$$P_G(z) = \sum_{y:(V\setminus\{t\})\to\{0,1\}} (-1)^{n-|y|} \det\left(L_s(y,z)\right). \tag{4}$$

The summation is over all 2^n assignments $y : V \setminus \{t\} \to \{0,1\}$. Hence, with the formula in Lemma (7), we now have a way to evaluate $P_G(z)$ in a particular point z in $2^n \operatorname{poly}(n)$ time. We will next see how we can speed-up the evaluation for a z from our sample space.

2.5 Random perturbations at T

Following [9] and [11], we perturb the Laplacian matrices, without affecting the determinant, so that in expectation many summands in the above formula Eq. 4 are zeroed-out. We introduce new random variables $q_v \in \{0,1\}$ for $v \in T$, sampled uniformly and independently, where T is the sampled set from **SamplePoint** and define the *q-perturbed Laplacian* of G as

$$L^q_{u,v} = \begin{cases} \sum_{wv\in A} z_{wv}y_w & \text{if } u = v, v \notin T \\ \sum_{wv\in A} z_{wv}y_w - q_v & \text{if } u = v, v \in T \\ -z_{uv}y_u & \text{if } uv \in A \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Comparing this to Eq. 3, we have only added a term on some of the diagonal entries in the rows indexed by our sampled set T. Note that these extra q_v variables do not affect the final inclusion–exclusion sum, as only the monomials representing Hamiltonian paths from s to t are counted, in particular only monomials with all n y_u-variables for $u \in V \setminus \{t\}$, confer [9] for a proof. Hence, irrespective of q, we can still compute the Hamiltonicity polynomial as:

$$P_G(z) = \sum_{y:(V\setminus\{t\})\to\{0,1\}} (-1)^{n-|y|} \det\left(L^q_s(y,z)\right). \tag{6}$$

What we have gained by doing this, is that the probability that a row indexed by $u \in T$ has its diagonal entry divisible by two, is 1/2, independently of other rows. We will next see how we can use this.

2.6 Efficient Evaluation of $P_G(z)$ given T

The basic idea is the same underlying the speed-ups in [7, 9, 11]. We make sure that in expectation, many summands in Eq. (6) will be trivially zero. Then, to evaluate the formula it suffices to list only the summands that are not trivially zero, so-called *contributing* terms, and sum up their contributions. Here, with "trivially zero", we will mean matrices that have at least $k = k_R$ rows of the matrix among the rows indexed by a vertex in the sampled set T from **SamplePoint** with all elements having all coefficients even. To see that such a term is zero, we merely have to recall Leibniz's determinant expansion of a matrix $M = \{m_{i,j}\}$:

$$\det(M) = \sum_{\sigma\in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} m_{i,\sigma(i)},$$

where S_n is the set of all permutations on n elements. Note in particular that in every term there is one element from each row. Hence, if the matrix has k rows in which every monomial ax^b in a ring element has a even, the product (over $\mathbb{Z}$) must be divisible by 2^k and hence cancel in the ring R.

Our algorithm to compute $P_G(z)$ will list the terms in Eq. 6 that have at least one odd coefficient in some ring element in at least $\tau - k + 1$ of the rows of L_s representing vertices in T. This is what is required to be a contributing terms. The algorithm outline is postponed to the next section.

We begin by arguing that, in expectation over the random q values, there are not too many contributing terms. Recalling Eq. 3, and inspecting any such row in the matrix $L_s(y)$ for a vertex $u \in T$, we see that

1. Off-diagonal entries are zero if $y_u = 0$ or $u = t$,
2. The diagonal entry is divisible by two if
$$\sum_{wu \in A} y_w = q_u (\bmod 2),$$
remembering that $z_{wu} = 1$ for all $wu \in A$ with $u \in T$.

Fix an assignment $y : V \setminus \{t\} \to \{0,1\}$, and let $Z_y \subseteq T$ be the vertices u for which the assignment sets $y_u = 0$, along with t if $t \in T$. From the above, the probability over the random q values, of the event ε_y that a fixed assignment y does *not* result in a trivially zero term in Eq. 6, is

$$\Pr_q[\varepsilon_y] = \left(\frac{1}{2}\right)^{|Z_y|} \sum_{i=0}^{k-1} \binom{|Z_y|}{i}. \tag{7}$$

Here we use that the diagonal entry of a row indexed by a vertex in Z_y is even with probability 1/2 independently of other rows as argued in Section 2.5. Let Y be the random variable equal to the number of assignments that are contributing. Then, in expectation

$$\mathbb{E}[Y] = \sum_{y \in V \setminus \{t\} \to \{0,1\}^n} \Pr_q[\varepsilon_y]. \tag{8}$$

We can bound the expectation as

▶ **Lemma 8.**

$$\mathbb{E}[Y] \in 2^{n - \Omega(\frac{n}{\delta})}.$$

Proof. From Eq. 7 and Eq. 8 we have

$$\mathbb{E}[Y] \leq \sum_{\substack{y \in V \setminus \{t\} \to \{0,1\}^n \\ |Z_y| < \frac{\tau}{3}}} 1 + \sum_{\substack{y \in V \setminus \{t\} \to \{0,1\}^n \\ |Z_y| \geq \frac{\tau}{3}}} \left(\frac{1}{2}\right)^{|Z_y|} \sum_{i=0}^{k-1} \binom{|Z_y|}{i}. \tag{9}$$

The left term in Eq. 9 is

$$2^{n-\tau} \left(\sum_{i=0}^{\tau/3-1} \binom{\tau}{i} \right) \in 2^{n - \Omega(\tau)},$$

and the right term in Eq. 9 is less than

$$2^n \left(\frac{1}{2^{\tau/3}} \sum_{i=0}^{k-1} \binom{\tau/3}{i} \right) \in 2^{n-\Omega(\tau)},$$

after remembering $k < \tau/10$ and $\tau = \frac{n}{20\delta}$, and noting that

$$\frac{1}{2^\gamma} \sum_{i=0}^{k-1} \binom{\gamma}{i},$$

for $\gamma \in \{\tau/3, \cdots \tau\}$ is maximised for $\gamma = \tau/3$. The stated bound in the lemma follows. ◀

2.7 Listing Contributing Terms

We finally describe how to list the contributing term assignments $y : V \setminus \{t\} \to \{0,1\}$ needed to compute $P_G(z)$ via Eq. 6 for a fixed q. The idea is to test for each partial assignment $y^* : T \setminus \{t\} \to \{0,1\}$ with the interpretation that $y_v = y^*_v$ for $v \in T \setminus \{t\}$, and each way of assigning parities $p : Z_{y^*} \to \{0,1\}$ to the diagonal entries of the vertices in Z_y that is consistent with a not trivially zero assignment, i.e., p takes the value 0 on at most $k-1$ rows. We then notice that these diagonal entries in Z_y describe a linear equation system over the variables in y outside of T. The equation system $E(y^*, p)$ consists of the equations (modulo two)

$$\sum_{wv \in A} y_w + q_v = p_v,$$

for each $v \in Z_y$, where we replace each variable y_w with $w \in T$ for its value y^*_w. We can list all solutions to this equation system by Gaussian elimination. We first solve for one solution and a null space basis. We next can enumerate all solutions by taking all linear combinations of the null space basis vectors with the solution. In summary our streaming procedure that generates all contributing terms' assignments is (we will think of it as a background process generating the solutions one-by-one):

■ **Procedure 2 ListingTerms** (outputs contributing assignments $y : V \setminus \{t\} \to \{0,1\}$ needed for Eq. 6).

```
1. For each $y^* : T \setminus \{t\} \to \{0,1\}$,
2.     For each $p : Z_{y^*} \to \{0,1\}$ with $|p| > |Z_{y^*}| - k$,
3.         Report every solution $y$ to $E(y^*, p)$.
```

Note that this lists every contributing term's assignment once since each y has precisely one restriction y^* on T and matches one of the tested p's. To bound the running time, we know from Lemma 8 that the output number of y assignments are at most $2^{n-\Omega(\frac{n}{\delta})}$ in expectation. This dominates the running time, since the number of equation systems considered, each of which can be solved in polynomial time in the sense of providing a parameterisation of the solution space as one solution vector along with the null space, is at most $3^\tau \in 2^{O(n/\delta)}$.

2.8 High-Level Algorithm

Putting the parts of the previous sections together, we are ready to give the high-level description of our algorithm in Theorem 1 as

Algorithm 3 DecideHamiltonicity (answers whether input G_{in} has a Hamiltonian cycle).

```
1. Repeat for $100 \log n$ times:
2.     Call **SamplePoint** to obtain point $z$ and subset $T$.
3.     Pick a $q$ uniformly at random.
4.     While there are still contributing terms:
5.         Get next $y$ from background process **ListingTerms**.
6.         If the number of generated terms is too big, continue to next outer repetition.
7.         Add $y$'s contribution to $P_G(z)$.
8.     If $P_G(z) \neq 0$ break and output "Yes".
9. Output "No".
```

In particular, there is no need to store the list of y assignments explicitly, but rather we use them one by one as they are generated to update the sum in Eq. 6. From Lemma 6 we know that a false negative happens with probability $1 - 1/100$. Since we pick $100 \log n$ sample points z, independently of each other, we will be unsuccessful in all of them with probability $(1 - 1/100)^{100 \log n} < n^{-1}$. From Section 2.7 we know the number of contributing term assignments and the running time of **ListingTerms** is $2^{n-\Omega(\frac{n}{\delta})}$ in expectation. If the number of generated terms are more than n times the expected value, we abort this z, T, q-value repetition at step 6 of the algorithm. This also happens only with probability n^{-1} by Markov's inequality. Altogether, the probability of a false negative is at most $\frac{2}{n}$. This concludes the proof of Theorem 1.

References

1 Alexander I. Barvinok. Two algorithmic results for the traveling salesman problem. *Math. Oper. Res.*, 21(1):65–84, 1996. doi:10.1287/moor.21.1.65.

2 Eric T. Bax. Inclusion and exclusion algorithm for the Hamiltonian path problem. *Inf. Process. Lett.*, 47(4):203–207, 1993.

3 Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63, January 1962. doi:10.1145/321105.321111.

4 Andreas Björklund. Determinant sums for undirected Hamiltonicity. *SIAM J. Comput.*, 43(1):280–299, 2014.

5 Andreas Björklund. Exploiting sparsity for bipartite Hamiltonicity. In Wen-Lian Hsu, Der-Tsai Lee, and Chung-Shou Liao, editors, *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*, volume 123 of *LIPIcs*, pages 3:1–3:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ISAAC.2018.3.

6 Andreas Björklund, Holger Dell, and Thore Husfeldt. The parity of set systems under random restrictions with applications to exponential time problems. 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 I*, volume 9134 of *Lecture Notes in Computer Science*, pages 231–242. Springer, 2015. doi:10.1007/978-3-662-47672-7_19.

7 Andreas Björklund and Thore Husfeldt. The parity of directed Hamiltonian cycles. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 727–735, 2013.

8 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. The traveling salesman problem in bounded degree graphs. *ACM Trans. Algorithms*, 8(2):18:1–18:13, 2012. doi:10.1145/2151171.2151181.

9 Andreas Björklund, Petteri Kaski, and Ioannis Koutis. Directed Hamiltonicity and out-branchings via generalized Laplacians. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 91:1–91:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.91.

10 Andreas Björklund, Petteri Kaski, and Ryan Williams. Generalized Kakeya sets for polynomial evaluation and faster computation of fermionants. *Algorithmica*, 81(10):4010–4028, 2019. doi:10.1007/s00453-018-0513-7.

11 Andreas Björklund and Ryan Williams. Computing permanents and counting Hamiltonian cycles by listing dissimilar vectors. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 25:1–25:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.25.

12 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast Hamiltonicity checking via bases of perfect matchings. *J. ACM*, 65(3):12:1–12:46, 2018. doi:10.1145/3148227.

13 Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal for the Society for Industrial and Applied Mathematics*, pages 1–10, 1962.

14 Erich Kaltofen. On computing determinants of matrices without divisions. In Paul S. Wang, editor, *Proceedings of the 1992 International Symposium on Symbolic and Algebraic Computation, ISSAC '92, Berkeley, CA, USA, July 27-29, 1992*, pages 342–349. ACM, 1992.

15 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

16 Richard M. Karp. Dynamic programming meets the principle of inclusion and exclusion. *Operations Research Letters*, 1(2):49–51, 1982.

17 Samuel Kohn, Allan Gottlieb, and Meryle Kohn. A generating function approach to the traveling salesman problem. In *Proceedings of the 1977 Annual Conference*, ACM '77, page 294–300. Association for Computing Machinery, 1977.

18 Łukasz Kowalik and Konrad Majewski. The asymmetric travelling salesman problem in sparse digraphs. In *15th International Symposium on Parameterized and Exact Computation (IPEC 2020)*, volume 180 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2020.23.

19 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.

20 W. T. Tutte. The dissection of equilateral triangles into equilateral triangles. *Mathematical Proceedings of the Cambridge Philosophical Society*, 44(4):463–482, 1948.

21 Ryan Williams. A new algorithm for optimal constraint satisfaction and its implications. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, volume 3142 of *Lecture Notes in Computer Science*, pages 1227–1237. Springer, 2004. doi:10.1007/978-3-540-27836-8_101.

Online Simple Knapsack with Reservation Costs

Hans-Joachim Böckenhauer
Department of Computer Science, ETH Zürich, Switzerland

Elisabet Burjons
Department of Computer Science, RWTH Aachen, Germany

Juraj Hromkovič
Department of Computer Science, ETH Zürich, Switzerland

Henri Lotze
Department of Computer Science, RWTH Aachen, Germany

Peter Rossmanith
Department of Computer Science, RWTH Aachen, Germany

Abstract

In the Online Simple Knapsack Problem we are given a knapsack of unit size 1. Items of size smaller or equal to 1 are presented in an iterative fashion and an algorithm has to decide whether to permanently reject or include each item into the knapsack without any knowledge about the rest of the instance. The goal is then to pack the knapsack as full as possible. In this work, we introduce a third option additional to those of packing and rejecting an item, namely that of reserving an item for the cost of a fixed fraction α of its size. An algorithm may pay this fraction in order to postpone its decision on whether to include or reject the item until after the last item of the instance was presented.

While the classical Online Simple Knapsack Problem does not admit any constantly bounded competitive ratio in the deterministic setting, we find that adding the possibility of reservation makes the problem constantly competitive, with varying competitive ratios depending on the value of α. We give upper and lower bounds for the whole range of reservation costs, with tight bounds for costs up to $1/6$ – an area that is strictly 2-competitive – , for costs between $\sqrt{2}-1$ and 1 – an area that is strictly $(2+\alpha)$-competitive up to $\phi-1$, and strictly $1/(1-\alpha)$-competitive above $\phi-1$, where ϕ is the golden ratio.

With our analysis, we find a counterintuitive characteristic of the problem: Intuitively, one would expect that the possibility of rejecting items becomes more and more helpful for an online algorithm with growing reservation costs. However, for higher reservation costs above $\sqrt{2}-1$, an algorithm that is unable to reject any items tightly matches the lower bound and is thus the best possible. On the other hand, for any positive reservation cost smaller than $1/6$, any algorithm that is unable to reject any items performs considerably worse than one that is able to reject.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online problem, Simple knapsack, Reservation costs

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.16

Related Version *Full Version*: `https://arxiv.org/abs/2009.14043`

Acknowledgements We want to thank the anonymous reviewers for pointing out imprecise formulations and helping to improve the overall quality of this work.

1 Introduction

Online algorithms can be characterized by receiving their input in an iterative fashion and having to act in an irrevocable way on each piece of the input, e. g., by deciding whether to include an element into the solution set or not. This has to be done with no additional knowledge about the contents or even the length of the rest of the instance that is still

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 16; pp. 16:1–16:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to be revealed. The goal, as with regular offline algorithms, is to optimize some objective function. In order to measure the performance of an online algorithm, it is compared to that of an optimal offline algorithm on the same instance. The worst-case ratio between the performances of these algorithms over all instances is then called the *strict competitive ratio* of an online algorithm, which was introduced by Sleator and Tarjan [22].

One of the arguably most basic and famous problems of online computation is called the *ski rental problem* [19]. In this problem, someone is going on a skiing holiday of not yet fixed duration without owning the necessary equipment. On each day, she decides, based solely on that day's short-term weather report, whether skiing is possible on that day or not. On each day with suitable weather, she can either buy the equipment or rent it for that day for a fixed percentage of the cost of buying a pair of ski. Arguably, the only interesting instances are those in which a selected number of days are suitable for skiing, followed by the rest of the days at which she is unable to go skiing anymore. This is simply due to the fact that, as long as she has not decided to buy a pair of ski yet, a day at which no skiing is possible requires no buy-or-rent decision. Thus, the problem can be simplified as follows: The input is a string of some markers that represent days suitable for skiing, and as soon as the instance ends, skiing is no longer possible.

This notion of delaying a decision for a fixed percentage of the cost is the model that we want to study in this work. Note that, while the ski-rental problem in its above-mentioned form only models a single buy-or-rent decision (buying or renting a single commodity), iterated versions of it have been discussed in the literature, with important applications, e. g., to energy-efficient computing [15, 1, 4]. In the following, we investigate the power of delaying decisions for another more involved problem, namely the online knapsack problem.

The *knapsack problem* is defined as follows. Given a set of items I, a size function $w\colon I \to \mathbf{R}$ and a gain function $g\colon I \to \mathbf{R}$, find a subset $S \subseteq I$ such that the sum of sizes of S is lower or equal to the so-called *knapsack capacity* (which we assume to be normalized to 1 in this paper) and the sum of the gains is maximized. The online variant of this problem reveals the items of I piece by piece, with an algorithm having to immediately decide whether to pack a revealed item or to discard it.

The knapsack problem is a classical hard optimization problem, with the decision variant being shown to be NP-complete as one of Karp's 21 NP-complete problems [18]. The offline variant of this problem was studied extensively in the past, showing for example that it admits a fully polynomial time approximation scheme [14].

A variant of this problem in which the gains of all items coincide with their respective sizes is called the *simple knapsack problem.* Both variants do not admit a constantly bounded competitive ratio [20]. In this paper, we focus on the online version of the latter variant, which we simply refer to as the *online knapsack problem*, short OnlineKP, if not explicitly stated otherwise.

We propose a rather natural generalization of the online knapsack problem. Classically, whenever an item of the instance is presented, an online algorithm has to irrevocably decide whether to include it into its solution (i. e., pack it into the knapsack) or to discard it. In our model, the algorithm is given a third option, which is to reserve an item for the cost of a fixed percentage $0 \leq \alpha \leq 1$ of its value. The algorithm may reserve an arbitrary number of items of the instance and decide to pack or discard them at any later point. Philosophically speaking, the algorithm may pay a prize in order to (partially) "go offline." It is easy to see that, for $\alpha = 0$, the complete instance can be learned before making a decision without any disadvantages, essentially making the problem an offline problem, while, for $\alpha = 1$, reserving an item is not better than discarding it because packing a reserved item does not add anything to the gain.

One of the key properties of our reservation model applied to the simple knapsack problem is its unintuitive behavior with respect to rejecting items. It shows some sharp thresholds for the competitive ratio at seemingly arbitrary points: For some interval of low reservation fees, the competitive ratio is not affected by the charge at all, on the next interval, it depends linearly on the charge, and on the last interval, the competitive ratio grows even faster. This behavior is further discussed in the following subsection.

Moreover, this extension to the knapsack problem is arguably quite natural: Consider somebody trying to book a flight with several intermediate stops. Since flight prices are subject to quick price changes, it might be necessary to invest some reservation costs even for some flights not ending up in the final journey. The knapsack problem with reservations might also be seen as a simple model for investing in a volatile stock market, where a specific type of deviates is available (at some cost) that allows the investor to fix the price of some stock for a limited period of time.

The OnlineKP has been extensively studied under many other variations, including buffers of constant size in which items may be stored before deciding whether to pack them, studied by Han et al. [12]. Here, the authors allow for a buffer of size at least the knapsack capacity into which the items that are presented may or may not be packed and from which a subset is then packed into the actual knapsack in the last step. They extensively study the case in which items may be irrevocably removed from the buffer. Our model can be understood as having an optional buffer of infinite size, with the caveat that each buffered item induces a cost. A variant without an additional buffer, but with the option to remove items from the knapsack at a later point was studied by Iwama et al. [16]. The same model with costs for each removal that are proportional to a fraction f of each item from the knapsack was researched by Han et al. [11], which is closer to our model. Their model allows an algorithm to remove items that were already packed into the knapsack, for a proportion of the value of these items. However, we do not know of any simple reduction from one model to the other, which is supported by the considerably different behavior of the competitive ratio relative to the reservation cost. For OnlineKP, Han et al. show that the problem is 2-competitive for a removal cost factor $f \leq 0.5$ and becomes $(1 + f + \sqrt{f^2 + 2f + 5})/2$-competitive for $f > 0.5$.

Other models of the (simple) online knapsack problem have been studied as well, such as the behavior of the resource-augmentation model, where the knapsack of the online algorithm is slightly larger than the one of the offline algorithm on the same instances, studied by Iwama et al. [17]. When randomization is allowed, the OnlineKP becomes 2-competitive using only a single random bit and does not increase its competitive ratio with any additional bits of randomization [6]. A relatively young measure of the informational complexity of a problem is that of advice complexity, introduced by Dobrev, Královič, and Pardubská [9], revised by Hromkovič et al. [13] and refined by Böckenhauer et al. [5]. In the standard model, an online algorithm is given an advice tape of bits that an oracle may use to convey some bits of information to an online algorithm about the instance in order to improve its performance. Not surprisingly, a single advice bit also improves the competitive ratio to 2, but interestingly, any advice in the size of $o(\log n)$ advice bits does not improve this ratio [6].

1.1 Our Contributions

We study the behavior of the knapsack problem with reservation costs for a reservation factor $0 < \alpha < 1$. We analyze several subintervals for α separately with borders at $1/6$, $\alpha_0 = 1 - \frac{\sqrt{2}}{2} \approx 0.293$, $\alpha_1 \approx 0.318$, $\alpha_2 \approx 0.365$, $\alpha_3 = \sqrt{2} - 1 \approx 0.414$, and $\phi - 1 \approx 0.618$,

Table 1 Competitive ratios proven in this work.

α	Lower bound		Upper bound	
$0 < \alpha \leq \frac{1}{6}$	2	Thm 3	2	Thm 1
$\frac{1}{6} < \alpha \leq 1-\frac{\sqrt{2}}{2}$	2	Thm 3	$2+\alpha$	Thm 5
$1-\frac{\sqrt{2}}{2} < \alpha \leq \approx 0.318$	$\frac{1}{(1-\alpha)^2}$	Thm 4	$2+\alpha$	Thm 5
$\approx 0.318 < \alpha \leq \approx 0.365$	≈ 2.15	Thm 4	$2+\alpha$	Thm 5
$\approx 0.365 < \alpha \leq \sqrt{2}-1$	$\frac{1+\alpha}{1-\alpha}$	Thm 4	$2+\alpha$	Thm 5
$\sqrt{2}-1 < \alpha < \phi-1$	$2+\alpha$	Thm 12	$2+\alpha$	Thm 5, 6
$\phi-1 \leq \alpha < 1$	$\frac{1}{1-\alpha}$	Thm 15	$\frac{1}{1-\alpha}$	Thm 14

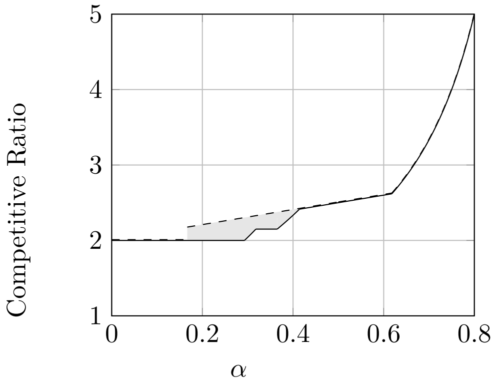


Figure 1 A schematic plot of the results proven in this paper. The full lines represent lower bounds while the dashed lines represent upper bounds.

where ϕ is the golden ratio. The bounds that we are providing, which are also illustrated in Figure 1, can be found in Table 1. We prove tight competitive ratios for $\alpha \leq \frac{1}{6}$ as well as for $\alpha \geq \sqrt{2}-1$.

We also take a look at a subclass of algorithms for this problem that never discard presented items up to a point where they stop processing the rest of the input, with arguably paradox results: One would expect that, for very small reservation costs, reserving items instead of rejecting them right away is more helpful than for large reservation costs and that rejecting becomes more and more helpful with increasing reservation costs. But we prove that, while, for small values of α, every algorithm that is unable to reject an item is strictly dominated by algorithms that are able to reject items, for larger values of α this does not appear to be the case, with our algorithms used for $\alpha \geq \sqrt{2}-1$ being nonrejecting and matching their lower bounds.

We cannot give a definitive answer on why this behavior can be observed. Our intuition is that for higher reservation costs, the behavior is more similar to the model without reservations. In this classical model, any errors that occur by ultimately not packing an item are punished severely, so while reserving everything is costly, it is not worse than rejecting any items.

The remainder of this paper is structured as follows: In Section 2, we present tight upper and lower bounds for small values of α as well as almost tight lower bounds for $\frac{1}{6} < \alpha < \sqrt{2}-1$. In Section 3, we look at values of α up to $\phi-1$, providing tight bounds for the complete interval. We conclude the competitive analysis with lower and upper bounds for larger values of α in Section 4. Section 5 is devoted to a discussion of algorithms that are unable to reject items. We conclude the paper in Section 6. A full version that contains all the proofs is available via arXiv [3].

1.2 Preliminaries

Our model can be defined as follows. Consider a knapsack of size 1 and a *reservation factor* α, with $0 \leq \alpha \leq 1$. A *request sequence* I is a list of item sizes $x_1, x_2, \ldots, x_n$ with $x_i \leq 1$ for $1 \leq i \leq n$, which arrive sequentially. At time step i, the knapsack is filled up to some size $t_i \leq 1$ and the reserved items add up to size r_i. When an item with size x_i arrives, an online algorithm may *pack* this item into the knapsack if $x_i + t_i \leq 1$, it may also *reject* the item, or *reserve* the item at cost $\alpha \cdot x_i$.

At step $n+1$, no new items arrive and the knapsack contains all items that were taken with total size t_n and the reserved items have size $r_n = R$. An algorithm can additionally pack any of the reserved items which still fit into the knapsack, up to some size $t \leq 1$. The *gain* of an algorithm $\mathtt{A}$ solving ReserveKP with a reservation factor α on an instance I is $\text{gain}_{\mathtt{A}}(I) = t - \alpha \cdot R$.

The *strict competitive ratio* of an algorithm $\mathtt{A}$ on an instance I is, given a solution with optimal gain $\text{gain}_{\mathtt{OPT}}(I)$ on this instance, $\rho_{\mathtt{A}}(I) = \text{gain}_{\mathtt{OPT}}(I)/\text{gain}_{\mathtt{A}}(I)$. The general strict competitive ratio of an algorithm $\mathtt{A}$ is taken as the worst case over all possible instances, $\rho_{\mathtt{A}} = \max_I\{\rho_{\mathtt{A}}(I)\}$.

The strict competitive ratio as defined above is a special case of the well-known *competitive ratio* which relaxes the above definition by allowing for a constant additive term in the definition. Note that this generalized definition only makes sense for online problems in which the optimal solution has unbounded gain. Since the gain of an optimal solution for ReserveKP is bounded by the knapsack capacity, we only work with strict competitive ratios in this paper and will simply call it *competitive ratio* from now on. For a thorough introduction to competitive analysis of online algorithms, see the textbooks by Borodin and El-Yaniv [7] and by Komm [19].

2 Small Reservation Costs

In this section, we analyze the case of reservation costs below $\frac{1}{6}$. Additionally, we provide lower bounds for values of α up to $\sqrt{2}-1$. The upper bounds left out in this section are a byproduct of the upper bounds of the next section for medium reservation costs and can be found there.

2.1 Upper Bound for $0 < \alpha \leq \frac{1}{6}$

▶ **Theorem 1.** *ReserveKP has a competitive ratio of at most 2 if $0 < \alpha \leq \frac{1}{6}$.*

To prove Theorem 1, we need to start with some technical considerations. First, we define a threshold μ that will be used prominently in our online algorithm in the proof of Theorem 1. We choose μ exactly so large, that packing items of size μ yields a competitive ratio of at most 2, even if we have to pay reservation costs for all those items. We define $\mu = 1/(2(1-\alpha))$.

▶ **Lemma 2.** *Let $\alpha \leq \frac{1}{2}$. If an algorithm has reserved items of total size $R \leq \mu$, then filling the knapsack up to μ is sufficient to achieve a competitive ratio of at most 2.*

Algorithm 1 The case $0 < \alpha \leq \frac{1}{6}$.

```
 1: $R := 0$;
 2: for $k = 1, \dots, n$ do
 3:     if $x_k \geq \mu$ then pack $x_k$ and stop
 4:     else if $\mu \leq x_k + R \leq 1$ then pack $x_k$ and all reserved items and stop
 5:     else if $R \leq 1 - \mu$ then
 6:         if $x_k \geq \frac{1}{2}$ then pack $x_k$ and all reserved items and stop
 7:         else reserve $x_k$ and let $R := R + x_k$
 8:     else if $x_k + R < \mu$ then reserve $x_k$ and let $R := R + x_k$
 9:     else
10:         if $x_k \geq \frac{1}{2}$ then
11:             if $\mathrm{gain}_{\mathtt{OPT}}(x_k, R) \geq \mu$ then pack optimally and stop
12:             else reject $x_k$
13:         else pack $x_k$ and reserved objects optimally and stop
14: pack the reserved items optimally
```

For small reservation costs, we design Algorithm 1, which unfortunately has a lot of case distinctions, most of which serve "easy" cases. The analysis of Algorithm 1 will prove Theorem 1. Let us look at some easy examples how the algorithm proceeds. If we feed a stream of small items of identical size $\varepsilon \ll \alpha$ into the algorithm, it will reserve all of them until their total size reaches μ, when all items are put into the knapsack and the algorithm stops in line 4. The gain of the algorithm is then at least $\mu - \alpha\mu$, which yields a competitive ratio of at least $\frac{1}{2}$.

In the next example, the reservation is still small, smaller than $1 - \mu$, and a request comes for an item that is larger than $\frac{1}{2}$. If the item is larger than μ, it will be packed in line 3, and achieve the desired competitive ratio due to Lemma 2. Otherwise, if it is smaller than μ, but it reaches μ together with the reservation, it will be packed in line 4, and will still achieve the desired competitive ratio, due to the same lemma, otherwise, it will be packed in line 6. Because the reservation is smaller than $1 - \mu$, the item fits in with the reserved items, the only concern is that it does not reach up to μ and we cannot apply the lemma, however, the desired competitive ratio is still achieved as the item is packed without being reserved first, making the total $\mathrm{gain}_{\mathtt{A}}(I)$ larger than $\frac{1}{2}$ as a result.

As a last example, we consider what happens when a large item (larger than $\frac{1}{2}$) arrives after the reservation size is already larger than $1 - \mu$, in this case, the only possibility that the item gets rejected is if the condition in line 11 is not satisfied. But literally this condition only requires that the optimal packing of the reserved items and the new item achieves the desired competitive ratio. Thus, we will have to make sure that the algorithm does not perform too badly by rejecting objects if the end of the request sequence is achieved and the algorithm is forced to pack in line 14.

Proof of Theorem 1. There are six ways how the algorithm might terminate. We have to prove for all six cases that the competitive ratio is at most 2, when the algorithm terminates at this point. These six possibilities correspond to lines 3, 4, 6, 11, 13, and 14.

Given a request x_k larger than μ, we know by Lemma 2 that we achieve the desired competitive ratio by packing x_k alone, which is done in line 3. If our request is smaller than μ, but $\mu \leq x_k + R \leq 1$, we again achieve the desired competitive ratio by packing all of the reserved objects together with x_k, thus satisfying the conditions for Lemma 2, which is carried out in line 4. Observe, for these two cases, that we do not reserve anything that would make R larger than μ.

Now, in line 5, if $R \leq 1-\mu$ and $x_k \geq \frac{1}{2}$, we know, since line 3 did not trigger the packing, that $x_k < \mu$ and all reserved items fit in the knapsack together with x_k. We only need to ensure that they achieve the desired competitive ratio, when the algorithm stops in line 6:

$$\frac{1}{x_k + (1-\alpha)R} \leq \frac{1}{1/2 + (1-\alpha)R} \leq 2\ .$$

If line 9 of the algorithm is reached, i. e., $x_k + R > 1$ and $R > 1-\mu$, we can have the case that $x_k \geq \frac{1}{2}$. In this case, line 11 will ensure that the algorithm only stops and packs when the desired competitive ratio is achieved.

On the other hand, if $x_k + R > 1$, $R > 1-\mu$, and $x_k < \frac{1}{2}$, i. e., if the algorithm has reached line 13, we know, by inspecting the algorithm, that every object in R is smaller than $\frac{1}{2}$, so each of them fits into the knapsack together with x_k. Indeed, otherwise one of the items $x_1, \ldots, x_{k-1}$ would have triggered the packing in line 4 or 6. If we were not able to fill the knapsack up to μ with x_k and the reserved objects, it would mean that there is a reserved object $x_i > 1-\mu$ that does not fit in the end. However, this object fits into the knapsack together with x_k. We distinguish two cases. If $x_k \geq 1-\mu$, we put these two objects into the knapsack and achieve a gain of $x_k + x_i \geq 1-\mu+1-\mu = 2-2\mu$, and we know that $2-2\mu \geq \mu$ for every $\alpha \leq \frac{1}{4}$, so we achieve a competitive ratio of 2 by Lemma 2 when the algorithm terminates in line 13. Otherwise, if $x_k < 1-\mu$, and packing x_i and x_k into the knapsack, together with some other available reserved items still does not fill the knapsack up to μ, there must be yet another object x_j in the reservation $x_j > 1-\mu$, and by the same analysis $x_i + x_j > \mu$ for $\alpha \leq \frac{1}{4}$, achieving yet again the desired competitive ratio.

Now we are left with only one case to analyze: The algorithm reaches the end of the request sequence I without having stopped. Then it packs an optimal subset of the reserved items and ends in line 14. If there is an optimal solution that does not contain any items that were rejected by the algorithm, then our algorithm can pack the same items as in the optimal solution and the gain is $(1-\alpha)\text{gain}_{\texttt{OPT}}(I)$, which leads to a competitive ratio of

$$\frac{\text{gain}_{\texttt{OPT}}(I)}{(1-\alpha)\text{gain}_{\texttt{OPT}}(I)} = \frac{1}{1-\alpha} \leq 2 \qquad \text{for } \alpha \leq \tfrac{1}{2}.$$

The optimal solution might include one of the rejected items, but only one, as all rejected items are larger than $\frac{1}{2}$. In this case, we know that the optimal solution contains the rejected item x_{rej} and some other items smaller than $\frac{1}{2}$ that will necessarily be reserved, that is, the size of an optimal solution is $x_{rej} + R'$ where $R' \subseteq R$. However, we also know that the rejected item did not fit with all reserved items when it appeared. This means that there is one reserved object x_i, of size $1-\mu < x_i < \frac{1}{2}$, and this object is not in R', as it does not fit with x_{rej}: otherwise it would have been taken by the algorithm, which then would stop, as two objects greater than $1-\mu$ that fit into the knapsack are enough to achieve a competitive ratio of 2 for $\alpha \leq \frac{1}{4}$ as we just saw in the previous case. Hence, the algorithm can pack at least $x_i + R'$, as $x_i < \frac{1}{2}$ and $x_i \notin R'$. Thus, the competitive ratio in this case is

$$\rho_{\texttt{A}} \leq \frac{x_{rej} + R'}{x_i + R' - \alpha R} \leq \frac{\mu + R'}{1-\mu+R'-\alpha\mu}\ ,$$

where the last inequality follows since $x_{rej} < \mu$ (otherwise, the algorithm would have terminated in line 3) and $R < \mu$ (otherwise, the algorithm would have stopped in line 4). We know that $0 \leq R' \leq 1-\mu$, but for the larger values of R' we know that, if at some point $x_i + R' \geq \mu$, the algorithm would have terminated sooner and not reached the end of the sequence. Therefore $R' \leq \mu - x_i \leq \mu - (1-\mu) \leq 2\mu - 1$ and $0 \leq R' \leq 2\mu - 1$. Then,

$$\frac{\mu + R'}{1-\mu+R'-\alpha\mu} \leq \frac{\mu}{1-\mu-\alpha\mu} \leq 2\ ,$$

for $0 \leq \alpha \leq \frac{1}{6}$, as can be shown with standard methods from calculus. ◀

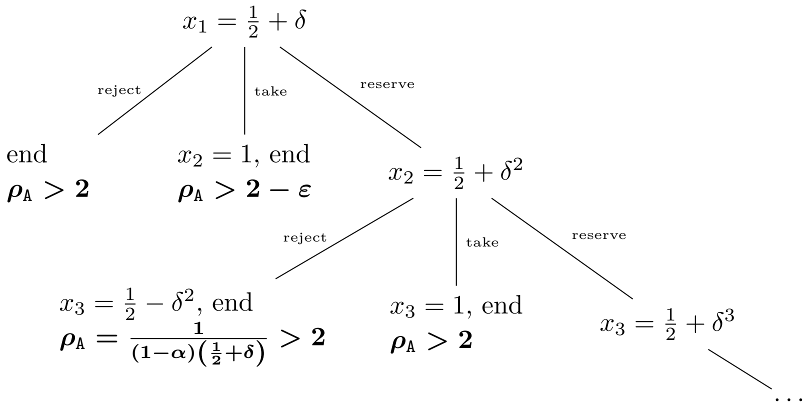

$x_1 = \frac{1}{2} + \delta$
reject
take
reserve
end
$\rho_A > 2$
$x_2 = 1$, end
$\rho_A > 2 - \varepsilon$
$x_2 = \frac{1}{2} + \delta^2$
reject
take
reserve
$x_3 = \frac{1}{2} - \delta^2$, end
$\rho_A = \frac{1}{(1-\alpha)(\frac{1}{2}+\delta)} > 2$
$x_3 = 1$, end
$\rho_A > 2$
$x_3 = \frac{1}{2} + \delta^3$
...

Figure 2 Sketch of the adversarial strategy that is used in the proof of Theorem 3.

2.2 Lower Bound for $0 < \alpha \leq \sqrt{2} - 1$

First we present an adversarial strategy that works for all values of α. Then we proceed to analyze the case where only three objects are presented as a generic adversarial strategy and find improved lower bounds for some values of α.

▶ **Theorem 3.** *For $\alpha > 0$ there exists no algorithm for reservation knapsack achieving a competitive ratio better than 2.*

Proof. Consider the following set of adversarial instances depicted in Figure 2. Given any $\varepsilon > 0$, the adversary presents first an object of size $\frac{1}{2} + \delta$ with $0 < \delta \ll \varepsilon$. If an algorithm takes this object, an object of size 1 will follow, making its competitive ratio $1/(\frac{1}{2}+\delta) > 2-\varepsilon$. If an algorithm rejects this object, no more objects will follow and it will not be competitive. If an algorithm reserves this object, then an object of size $\frac{1}{2} + \delta^2$ will be presented. Observe, that these two objects do not fit together into the knapsack. If an algorithm takes this object, an object of size 1 will be presented, and again the algorithm will achieve a competitive ratio worse than $2 - \varepsilon$. If an algorithm rejects this object, then an object of size $\frac{1}{2} - \delta^2$ will be presented. This object does not fit in the knapsack with the first one, thus the algorithm can only pack the first object, obtaining a competitive ratio worse than $2 - \varepsilon$. If an algorithm reserves it instead, an object of size $\frac{1}{2} + \delta^3$ will be presented. The adversary can follow this procedure on and on, and in each step the competitive ratios for algorithms that accept or reject the offered item only get worse due to the additional reservation costs.

The adversary can stop offering items as soon as the reservation costs are such that filling the knapsack will only result in a competitive ratio worse than 2. This shows that, for every $\varepsilon > 0$, the competitive ratio is at least $2 - \varepsilon$, so the best competitive ratio is at least 2. ◀

The adversarial strategy depicted in Figure 2 works for every positive value of α. Observe that if an algorithm continues to reserve items, the cumulated reservation cost will eventually exceed any remaining gain.

This strategy provides us with a lower bound that does not match the upper bound for $\alpha > \frac{1}{6}$. We improve the lower bound for larger α by designing a generic adversarial strategy with three items shown in Figure 3, which shows the competitive ratios for all possible outcomes. It is therefore bounded by

$$\rho_A \geq \min\left\{\frac{1}{s}, \frac{1}{t - \alpha s}, \frac{t}{(1-\alpha)t - \alpha s}, \frac{t}{s - \alpha s}\right\}. \quad (1)$$

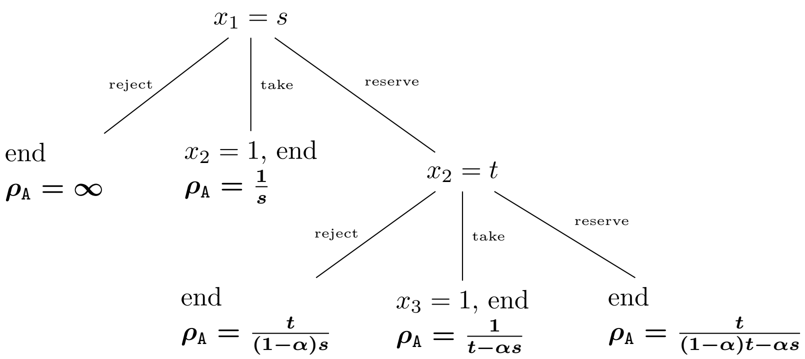


Figure 3 Diagram of a generic adversarial strategy with 3 items.

To prove a lower bound for every $0 < \alpha < 1$, we can choose s and t in order to make (1) as large as possible. Standard calculus leads to the bounds in the following theorem.

▶ **Theorem 4.** *The competitive ratio of* ReserveKP *is at least*
(a) $1/(1-\alpha)^2 \approx 2 \ldots 2.15$, *for* $0.293 \leq \alpha < 0.318$*;*
(b) $(2-\alpha_1)/(1-\alpha_1+\alpha_1^2) \approx 2.15$, *for* $0.318 \leq \alpha < 0.365$ *(recall that* $\alpha_1 \approx 0.318$*);*
(c) $(1+\alpha)/(1-\alpha) \approx 2.15 \ldots 2.41$, *for* $0.365 \leq \alpha < 0.414$*; and*
(d) $2+\alpha \approx 2.41 \ldots 3$, *for* $0.414 \leq \alpha \leq 1$.

3 Medium Reservation Costs

We now consider the case of medium reservation costs, i. e., $\sqrt{2}-1 \leq \alpha < \phi - 1 = \frac{\sqrt{5}}{2} - \frac{1}{2}$. We will provide an upper bound of $2+\alpha$ for $\alpha < \phi - 1$ in Theorems 5 and 6. We complement these upper bounds by a tight lower bound of $2+\alpha$ for the whole range of medium-size reservation cost.

3.1 Upper Bound for $\frac{1}{6} < \alpha < \phi - 1$

We prove now a general upper bound for $\alpha < \phi - 1$, but with a competitive ratio that depends on the parameter α. We split the proof, into two pieces: Theorem 5 handles the case for values of α up to $\frac{1}{2}$, . Theorem 6 contains an induction over the number of large elements in an instance, which proves the upper bound for the rest of the interval up to $\phi - 1$.

▶ **Theorem 5.** ReserveKP *has a competitive ratio of at most* $2+\alpha$ *if* $0 < \alpha \leq \frac{1}{2}$.

▶ **Theorem 6.** ReserveKP *has a competitive ratio of at most* $2+\alpha$ *if* $\frac{1}{2} < \alpha < \phi - 1$.

We consider Algorithm 2, which, unlike Algorithm 1, does not reject any offered item until it stops processing the rest of the input. In Section 5, we further discuss this class of algorithms and when they are optimal. We need the following technical lemmas.

▶ **Lemma 7.** *The size of* R *in Algorithm 2 is never larger than* $1/(2+\alpha)(1-\alpha)$.

Note, that for every considered value of α the upper bound on R is positive, that is, $(2+\alpha)(1-\alpha) \geq 0$ if α is smaller than 1. With Lemma 7 we can prove the following claim.

▶ **Lemma 8.** *If Algorithm 2 packs at least* $1/(2+\alpha)(1-\alpha)$ *into the knapsack, then its competitive ratio is at most* $2+\alpha$.

Algorithm 2 Competitive ratio $2 + \alpha$ for $0 < \alpha \leq \phi - 1$.

```
$R := 0$;
for $k = 1, \dots, n$ do
    if $x_k + (1 - \alpha)R \geq 1/(2 + \alpha)$ then
        pack $x_1, \dots, x_k$ optimally;
        stop
    else
        reserve $x_k$;
        $R := R + x_k$
pack $x_1, \dots, x_n$ optimally
```

The next lemma allows us to restrict our attention to ordered sequences of items. It proves that, if an instance violated the upper bound for Algorithm 2, we could rearrange the first $k - 1$ items in decreasing order and get another counterexample.

▶ **Lemma 9.** *If $x_1, \dots, x_n$ is an instance for Algorithm 2, whose packing is triggered by x_k, with a competitive ratio larger than $2+\alpha$, then there is another instance $x_{i_1}, x_{i_2}, x_{i_3}, \dots, x_{i_{k-1}}$, $x_k, \dots, x_n$ where $(i_1, i_2, \dots, i_{k-1})$ is a permutation of $(1, \dots, k - 1)$ and $x_{i_1} \geq x_{i_2} \geq \dots \geq x_{i_{k-1}}$, that also has a competitive ratio larger than $2 + \alpha$.*

From Lemma 8, we know that, if Algorithm 2 packs at least $1/(2 + \alpha)(1 - \alpha)$, this guarantees a competitive ratio of at most $2 + \alpha$. Thus, if we have enough elements that are smaller than the gap of size $1 - 1/(2 + \alpha)(1 - \alpha)$, those elements can be packed greedily and always achieve the desired competitive ratio. Let us call *small items* those of size smaller than

$$1 - \frac{1}{(2 + \alpha)(1 - \alpha)} = \frac{(2 + \alpha)(1 - \alpha) - 1}{(2 + \alpha)(1 - \alpha)} = \frac{1 - \alpha - \alpha^2}{(2 + \alpha)(1 - \alpha)} \tag{2}$$

This definition is only valid when (2) is positive, that is, when $1 - \alpha - \alpha^2 \geq 0$, which is the case if $0 < \alpha \leq \phi - 1$, including our desired range. We call *large items* those of larger size. Let α_4 be the unique positive real root of the polynomial $1 - 2\alpha - \alpha^2 + \alpha^3$, i. e.,

$$\alpha_4 = \frac{1}{3} + \frac{2\sqrt{7}}{3} \cos\left(\frac{1}{3} \arccos\left(-\frac{1}{2\sqrt{7}}\right) - \frac{2\pi}{3}\right) \approx 0.445 .$$

▶ **Lemma 10.** *Given any request sequence $x_1 \geq x_2 \geq \dots \geq x_{k-1}, x_k, \dots, x_n$, where x_k triggers the packing in Algorithm 2, there is at most one large item if $0 < \alpha \leq \alpha_4$ and there are at most two large items if $\alpha_4 < \alpha \leq 0.5$.*

Proof. Assume by contradiction that there exists a request sequence where $x_1 \geq x_2 \geq \dots \geq x_i \geq \frac{1-\alpha-\alpha^2}{(2+\alpha)(1-\alpha)}$, with $k > i$. Any item x_j with $j \leq i$ satisfies

$$x_j \leq \frac{1}{2 + \alpha} - (1 - \alpha)(x_1 + x_2 + \dots + x_{j-1}) ,$$

in particular,

$$x_i \leq \frac{1}{2 + \alpha} - (1 - \alpha)(x_1 + x_2 + \dots + x_{i-1}) . \tag{3}$$

All of the contributions of previous requests are negative, so in order to obtain a maximal value for x_i, we need that $x_1, \dots, x_{i-1}$ are minimal, but by construction, still greater or equal than x_i. Thus, the maximal value is obtained when $x_1 = x_2 = \dots = x_i$. In this case, we obtain from (3) the following upper bound on the value of x_i, $x_i \leq \frac{1}{2+\alpha} - (1-\alpha)(i-1)x_i$ which we solve for x_i and obtain

$$x_i \leq \frac{1}{(2+\alpha)(i(1-\alpha)+\alpha)} \ . \tag{4}$$

We also know that x_i is a large item, thus we can also state the following lower bound on x_i

$$x_i \geq \frac{1-\alpha-\alpha^2}{(2+\alpha)(1-\alpha)} \ . \tag{5}$$

If we take into account both (4) and (5) we get $\frac{1}{(2+\alpha)(i(1-\alpha)+\alpha)} \geq \frac{1-\alpha-\alpha^2}{(2+\alpha)(1-\alpha)}$ which we solve for i to obtain

$$i \leq 1 + \frac{\alpha^2}{(1-\alpha)(1-\alpha-\alpha^2)} \ .$$

In particular, for $i = 2$ we get $2(1-\alpha)(1-\alpha-\alpha^2) \leq (1-\alpha)(1-\alpha-\alpha^2) + \alpha^2$ which is equivalent to $(1-\alpha)(1-\alpha-\alpha^2) \leq \alpha^2$ and thus to $1 - 2\alpha - \alpha^2 + \alpha^3 \leq 0$, which means that the number of large items is strictly smaller than 2 for $\alpha \leq \alpha_4$.

For $i = 3$, we get $3(1-\alpha)(1-\alpha-\alpha^2) \leq (1-\alpha)(1-\alpha-\alpha^2) + \alpha^2$ or equivalently $2(1-2\alpha+\alpha^3) \leq \alpha^2$. Hence, $2 - 4\alpha - \alpha^2 + 2\alpha^3 \leq 0$, which means that the number of large items is strictly smaller than 3 for $\alpha \leq 0.5$, since 0.5 is the unique positive real root of the left-hand-side polynomial. ◀

Now we are ready to prove the claimed competitive ratio of Algorithm 2.

Proof sketch of Theorem 5. Let us consider first the case where $\alpha \leq \alpha_4$, which means that only one large element can appear in the request sequence without triggering the packing.

We assume that there is a shortest contradictory sequence containing at least two elements, which at some point, namely with request x_k, triggers Algorithm 2 to pack. If there is no such sequence, it is rather easy to see that the claimed competitive ratio can be achieved. We consider several cases according to the sizes of the items which leads to the desired result.

In the case where $\alpha \leq 0.5$, we can use similar arguments to prove the claim. ◀

We continue with proving an upper bound for the rest of the interval, which is an induction over the number of large elements. Before we start our proof, we provide a lemma that allows us to ignore possible small elements during the proof of Theorem 6.

▶ **Lemma 11.** *Given a request sequence without small elements for which Algorithm 2 does not achieve a competitive ratio of $2+\alpha$, adding small elements to it will only improve its competitive ratio.*

Proof. Let us consider a request sequence containing only large elements $x_1 \geq \ldots \geq x_{k-1}, x_k, \ldots, x_n$, where x_k is the element triggering the packing for Algorithm 2, and the achieved competitive ratio is larger than $2+\alpha$. This means, by Lemma 8, that the packed knapsack size is smaller than $1/(2+\alpha)(1-\alpha)$. By definition, if we add enough small elements to the request sequence before x_k, the small elements can be packed greedily until the knapsack is filled up to $1/(2+\alpha)(1-\alpha)$, achieving the desired competitive ratio. If not enough of them are added before x_k, the small elements requested before x_k will be reserved but will still be able to be packed, so they will never contribute negatively to the total packing gain. ◀

Proof of Theorem 6. We prove by induction that, for any finite number of large elements before the packing, Algorithm 2 achieves the desired competitive ratio for $\alpha < \phi - 1$. The base case for zero large elements is trivial, as we can greedily reserve and later pack all small elements to get the desired competitive ratio. Let us assume that Algorithm 2 achieves the desired competitive ratio for any request sequence with less than $k-1$ large elements before the algorithm packs and stops.

If a request sequence has $k-1$ large elements we can assume by Lemma 9, that the smallest of those is x_{k-1}, and we can also assume that x_k triggers the packing by Lemma 11. We distinguish two cases.

1. When the packing is triggered, x_{k-1} is not part of an optimal packing.
   In this case, $\sum_{j \text{ is packed}} x_j + x_{k-1} > 1$. Then the following bounds hold.

$$x_{k-1} + (1-\alpha)\left(\sum_{j<k-1} x_j\right) < \frac{1}{2+\alpha}\ ,$$

   $x_{k-1} \leq x_1 < 1/(2+\alpha)$, and $\frac{\alpha}{1-\alpha} < 1-\alpha$ for $\alpha < \phi - 1$.
   Thus, the gain that Algorithm 2 achieves is

$$\begin{aligned}
&\sum_{j \text{ is packed}} x_j - \alpha R \\
&\qquad \geq 1 - x_{k-1} - \alpha\left(\sum_{j\leq k-1} x_j\right) \\
&\qquad = 1 - (1+\alpha)x_{k-1} - \alpha\left(\sum_{j<k-1} x_j\right) \\
&\qquad = 1 - 2\alpha x_{k-1} - (1-\alpha)\left(x_{k-1} + \frac{\alpha}{1-\alpha}\left(\sum_{j<k-1} x_j\right)\right) \\
&\qquad \geq 1 - 2\alpha x_{k-1} - (1-\alpha)\left(x_{k-1} + (1-\alpha)\left(\sum_{j<k-1} x_j\right)\right) \\
&\qquad \geq 1 - 2\alpha x_{k-1} - (1-\alpha)\left(\frac{1}{2+\alpha}\right) \\
&\qquad \geq 1 - \frac{2\alpha}{2+\alpha} - \frac{1-\alpha}{2+\alpha} \\
&\qquad \geq \frac{1}{2+\alpha}\ ,
\end{aligned}$$

   as we wanted.

2. When the packing is triggered, x_{k-1} is part of an optimal packing. We consider two subcases.
   a. Taking x_{k-1} out of the request sequence still triggers the packing.
      This means that $x_k + (1-\alpha)(R - x_{k-1}) \geq \frac{1}{2+\alpha}$ holds. We can thus consider the sequence $x_1, \ldots, x_{k-2}, x_k$. This sequence has $k-2$ large elements, and x_{k-1} cannot be part of its optimal solution, thus its competitive ratio is at most $2+\alpha$ by the induction hypothesis, and the competitive ratio after adding x_{k-1} can only get better.
   b. Taking x_{k-1} out of the request sequence does not trigger the packing.

This means that $x_k + (1-\alpha)(R - x_{k-1}) < \frac{1}{2+\alpha}$, but also

$$x_k + (1-\alpha)R \geq \frac{1}{2+\alpha} ,$$

and if we let x_j be the smallest element that does not get packed, $x_j \geq x_{k-1}$ holds. Also, because of the optimality of the packing $x_k \geq x_j$, (otherwise one can take all of the reserved elements as the packing and obtain a better bound) and

$$\sum_{t \text{ is packed}} x_t - x_{k-1} + x_j > 1$$

holds. Moreover, we can bound x_j by $x_j \leq x_1 \leq \frac{1}{2+\alpha}$. With these bounds, the gain incurred by the algorithm is at least

$$\begin{aligned}
&\sum_{t \text{ is packed}} x_t - \alpha R \\
&\geq 1 - x_j + x_{k-1} - \alpha R \\
&\geq 1 - x_j + \frac{x_k}{1-\alpha} + R - \frac{1}{(1-\alpha)(2+\alpha)} - \alpha R \\
&= 1 - x_j + \frac{x_k}{1-\alpha} + (1-\alpha)R - \frac{1}{(1-\alpha)(2+\alpha)} \\
&\geq 1 - x_j + \frac{\alpha x_k}{1-\alpha} + \frac{1}{2+\alpha} - \frac{1}{(1-\alpha)(2+\alpha)} \\
&= \frac{1}{2+\alpha} + \frac{1-\alpha-\alpha^2}{(1-\alpha)(2+\alpha)} - x_j + \frac{\alpha x_k}{1-\alpha} \\
&\geq \frac{1}{2+\alpha} + \frac{1-\alpha-\alpha^2}{(1-\alpha)(2+\alpha)} + \frac{\alpha-(1-\alpha)}{1-\alpha} x_k \\
&\geq \frac{1}{2+\alpha} + \frac{1-\alpha-\alpha^2}{(1-\alpha)(2+\alpha)} + \frac{2\alpha-1}{1-\alpha} x_k \\
&\geq \frac{1}{2+\alpha} ,
\end{aligned}$$

where the last step is trivially true for any $\alpha \geq 1/2$. Thus we get the desired competitive ratio in all of the considered range for α.

This proves the induction step, and thus the desired upper bound on the competitive ratio. ◀

3.2 Lower Bound for $\sqrt{2} - 1 \leq \alpha \leq \phi - 1$

We now prove that, for the whole interval of medium reservation costs, no algorithm can achieve a better competitive ratio than $2 + \alpha$.

▶ **Theorem 12.** *Given an $\varepsilon > 0$ and an α such that $\sqrt{2}-1 \leq \alpha < 1$, there exists no algorithm for reservation knapsack achieving a competitive ratio of $2 + \alpha - \varepsilon$.*

Proof. Consider the following set of adversarial instances. First, the adversary presents an item of size $\frac{1}{2+\alpha}$. If an algorithm takes this item, the adversary presents an item of size 1, and the algorithm has a competitive ratio of $2+\alpha$ as claimed. If an algorithm rejects this item, the adversary will present no further items; thus, the algorithm will not be competitive at all. If the item is reserved, the adversary presents a second item of size $1 - \frac{1}{2+\alpha} + \delta$, where $\delta < \frac{\varepsilon}{(2+\alpha)(2+\alpha-\varepsilon)}$. Note that the second item is larger than the first item for all $\alpha > 0$ and that they do not fit together into the knapsack. If the item is taken, again the adversary presents an item of size 1 and ends the sequence. Thus, any algorithm reserving the first item and taking the second item has a competitive ratio of no better than

$$\frac{1}{1-\frac{1}{2+\alpha}+\delta-\frac{\alpha}{2+\alpha}} = \frac{1}{\frac{1}{2+\alpha}+\delta} > 2+\alpha-\varepsilon\ ,$$

where the last inequality follows from the choice of δ. If the second item is rejected, the adversary will present no further items and thus any algorithm following this strategy will have a competitive ratio of

$$\frac{1-\frac{1}{2+\alpha}+\delta}{\frac{1}{2+\alpha}-\frac{\alpha}{2+\alpha}} = \frac{\frac{1+\alpha}{2+\alpha}+\delta}{\frac{1-\alpha}{2+\alpha}} > \frac{1+\alpha}{1-\alpha} > 2+\alpha$$

where the last inequality follows from the fact that $\frac{1+\alpha}{1-\alpha} > 2+\alpha$, for all values of $\alpha \geq \sqrt{2}-1$. If the second item is reserved, no further items will be presented by the adversary. Thus, any algorithm following this strategy will have a competitive ratio of

$$\begin{aligned} \frac{1-\frac{1}{2+\alpha}+\delta}{1-\frac{1}{2+\alpha}+\delta-\frac{\alpha}{2+\alpha}-\alpha+\frac{\alpha}{2+\alpha}-\alpha\delta} &= \frac{\frac{1+\alpha}{2+\alpha}+\delta}{\frac{1-\alpha-\alpha^2}{2+\alpha}+\delta(1-\alpha)} \\ &> \frac{\frac{1+\alpha}{2+\alpha}}{\frac{1-\alpha-\alpha^2}{2+\alpha}+\delta} > \frac{1-\alpha}{\frac{1-\alpha-\alpha^2}{2+\alpha}+\delta}, \end{aligned}$$

where the last inequality again follows from the fact that $\frac{1+\alpha}{1-\alpha} > 2+\alpha$, for all values of $\alpha \geq \sqrt{2}-1$. We claim that

$$\frac{1-\alpha}{\frac{1-\alpha-\alpha^2}{2+\alpha}+\delta} > 2+\alpha\ .$$

Since $1-\alpha-\alpha^2$ is positive for all $\alpha < \phi - 1$, this is equivalent to $\frac{1-\alpha}{2+\alpha} > \frac{1-\alpha-\alpha^2}{2+\alpha} + \delta$ or equivalently $\alpha^2 > (2+\alpha)\delta$ which is true for all reasonable choices of δ. ◀

4 High Reservation Costs

In this section, we analyze the competitive ratio in the remaining interval of reservation costs between $\phi - 1$ and 1. We prove a tight bound of $\frac{1}{1-\alpha}$ on the competitive ratio.

4.1 Upper Bound for $\phi - 1 \leq \alpha < 1$

For proving an upper bound, we consider Algorithm 3 and first bound its reservation costs.

▶ **Lemma 13.** *For Algorithm 3, the reservation cost R is never larger than* 1.

We are now ready to prove the desired competitive ratio for Algorithm 3.

▶ **Theorem 14.** *Algorithm 3 is an online algorithm for* ReserveKP *achieving a competitive ratio of at most $\frac{1}{1-\alpha}$, for all $\phi - 1 \leq \alpha < 1$.*

Proof sketch. The full proof of this theorem is structurally very similar to that of Theorem 6. Let us first assume that we run Algorithm 3 on an instance $x_1, \ldots, x_n$, and no element triggers the packing. This means that all elements are reserved. But we know by Lemma 13

Algorithm 3 Algorithm for $\phi - 1 \leq \alpha < 1$.

```
$R := 0$;
for $k = 1, \ldots, n$ do
    if $x_k + (1-\alpha)R \geq 1 - \alpha$ then
        pack $x_1, \ldots, x_k$ optimally;
        stop
    else
        reserve $x_k$;
        $R := R + x_k$
pack $x_1, \ldots, x_n$ optimally
```

that the reservation never exceeds the capacity of the knapsack. This means that the optimal solution packs all offered elements. Thus, the algorithm achieves a competitive ratio of

$$\frac{\sum_{i=1}^{n} x_{k-1}}{(\sum_{i=1}^{n} x_{k-1}) - \alpha \sum_{i=1}^{n} x_{k-1}} = \frac{1}{1-\alpha} .$$

Now, it remains to analyze the case where an instance $x_1, \ldots, x_n$ triggers the packing, for some x_k. We do an induction on the value of k. If $k = 1$, the first item offered triggers the packing, thus it holds that $x_1 \geq 1 - \alpha$ and the gain of the algorithm is at least $1 - \alpha$ as we expected. Now, we assume that Algorithm 3 has a gain of at least $1 - \alpha$ on any request sequence triggering the algorithm to pack and stop before k elements are offered.

We proceed similarly to the proof of Theorem 6, making a case distinction over whether x_{k-1} is part of an optimal packing. ◀

4.2 Lower Bound for $\phi - 1 \leq \alpha < 1$

Now we present a lower bound of $\frac{1}{1-\alpha}$ for $\phi - 1 \leq \alpha < 1$. Observe that this lower bound works for all α. However, for smaller values of α, we see that $2 + \alpha > \frac{1}{1-\alpha}$, so Theorem 12 already gives a better lower bound for that range.

▶ **Theorem 15.** *For any $\varepsilon > 0$ and any α such that $\phi - 1 \leq \alpha < 1$, no online algorithm for* ReserveKP *can achieve a competitive ratio of $\frac{1}{1-\alpha} - \varepsilon$.*

Proof sketch. The proof of this theorem is structurally similar to that of Theorem 12. ◀

5 Nonrejecting Algorithms

We call an algorithm *nonrejecting* if it only chooses from one of the options *pack*, *reserve* or *stop and pack* for each item that it is given. Such an algorithm is thus unable to reject any items until the point where it discards the remaining elements of an instance.

A valid intuition for ReserveKP might be the following: If the cost of reservation is very small, rejecting an item should not be necessary, as even when an item cannot be packed, the cost of reserving it is negligible. On the other hand, when the reservation cost is rising, aggressively reserving items may seem like a very bad strategy, as the risk of not being able to utilize reserved items may come to mind. Interestingly, both of these intuitions turn out to be wrong, which we will show by first giving a lower bound for nonrejecting algorithms that exceeds the upper bound of a rejecting algorithm for small α and that tightly matches the upper bound of a nonrejecting algorithm for bigger α.

We first provide a lower bound on the competitive ratio for nonrejecting algorithms.

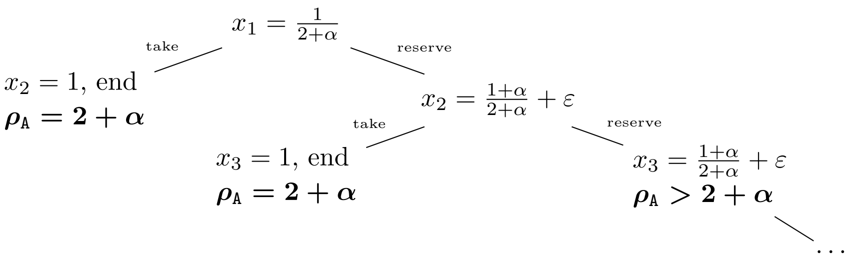

$x_1 = \frac{1}{2+\alpha}$
take
reserve
$x_2 = 1$, end
$\rho_A = 2 + \alpha$
$x_2 = \frac{1+\alpha}{2+\alpha} + \varepsilon$
take
reserve
$x_3 = 1$, end
$\rho_A = 2 + \alpha$
$x_3 = \frac{1+\alpha}{2+\alpha} + \varepsilon$
$\rho_A > 2 + \alpha$
...

Figure 4 Sketch of the adversarial strategy that is used in the proof of Theorem 16. Upon continued reservation, a new item of the same size is repeatedly presented.

▶ **Theorem 16.** *There exists no deterministic online algorithm for* ReserveKP *that does not reject any elements with a competitive ratio better than* $2 + \alpha$ *for any* $0 < \alpha < 1$.

Proof sketch. The proof of this theorem is structurally similar to the proofs of Theorems 12 and 15. A sketch of the adversarial strategy can be found in Figure 4. ◀

Combined with the upper bound given in Lemma 2 we see that an algorithm that is unable to reject items performs quite a bit worse than one that is able to reject items, such as Algorithm 1. Thus, an algorithm needs to be able to reject items to become 2-competitive for small values of α. On the other hand, the lower bound provided in Theorem 16 matches the upper bound of Theorem 5, which is based on the nonrejecting Algorithm 2. Thus, there are nonrejecting algorithms for every $\alpha \geq \sqrt{2} - 1$ that are at least as good as any other algorithms that are able to reject items.

6 Further Work

In this work, we give first bounds on this new model, but left a gap between the upper and lower bounds for $\frac{1}{6} < \alpha < \sqrt{2} - 1$. It would be of interest to us how the complete picture of all tight bounds looks like. We left the case open where the item costs are not proportional to their sizes, which seems also constantly competitive depending on α. We also leave the randomized and advice complexity of ReserveKP open.

Furthermore, one could consider a variant where reservation costs are refunded if the item is used or where one pays a fee proportional to the size of all reserved items in each step.

The concept of reservation may be applied to other online problems such as online call admission problems in networks [2, 7, 19] or problems of embedding guest graphs into a host graph. In the online path packing problem, one packs paths in a edge-disjoint way (sometimes node-disjointly) into a graph, which is a generalization of ReserveKP, thus inheriting all lower bounds. The offline version was studied on many types of graphs, with an incomplete selection being [10, 21, 23, 8].

References

1 Susanne Albers. Energy-efficient algorithms. *Commun. ACM*, 53(5):86–96, 2010. doi:10.1145/1735223.1735245.

2 Baruch Awerbuch, Yossi Azar, and Serge A. Plotkin. Throughput-competitive on-line routing. In *34th Annual Symposium on Foundations of Computer Science, Palo Alto, California, USA, 3-5 November 1993*, pages 32–40. IEEE Computer Society, 1993. doi:10.1109/SFCS.1993.366884.

3 Hans-Joachim Böckenhauer, Elisabet Burjons, Juraj Hromkovič, Henri Lotze, and Peter Rossmanith. Online simple knapsack with reservation costs. *CoRR*, abs/2009.14043, 2020. arXiv:2009.14043.

4 Hans-Joachim Böckenhauer, Richard Dobson, Sacha Krug, and Kathleen Steinhöfel. On energy-efficient computations with advice. In Dachuan Xu, Donglei Du, and Ding-Zhu Du, editors, *Computing and Combinatorics - 21st International Conference, COCOON 2015, Beijing, China, August 4-6, 2015, Proceedings*, volume 9198 of *Lecture Notes in Computer Science*, pages 747–758. Springer, 2015. doi:10.1007/978-3-319-21398-9_58.

5 Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. Online algorithms with advice: The tape model. *Inf. Comput.*, 254:59–83, 2017. doi:10.1016/j.ic.2017.03.001.

6 Hans-Joachim Böckenhauer, Dennis Komm, Richard Královič, and Peter Rossmanith. The online knapsack problem: Advice and randomization. *Theor. Comput. Sci.*, 527:61–72, 2014. doi:10.1016/j.tcs.2014.01.027.

7 Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

8 Darryn E. Bryant. Packing paths in complete graphs. *J. Comb. Theory, Ser. B*, 100(2):206–215, 2010. doi:10.1016/j.jctb.2009.08.004.

9 Stefan Dobrev, Rastislav Královič, and Dana Pardubská. Measuring the problem-relevant information in input. *ITA*, 43(3):585–613, 2009. doi:10.1051/ita/2009012.

10 András Frank. Packing paths in planar graphs. *Combinatorica*, 10(4):325–331, 1990. doi:10.1007/BF02128668.

11 Xin Han, Yasushi Kawase, and Kazuhisa Makino. Online unweighted knapsack problem with removal cost. *Algorithmica*, 70(1):76–91, 2014. doi:10.1007/s00453-013-9822-z.

12 Xin Han, Yasushi Kawase, Kazuhisa Makino, and Haruki Yokomaku. Online knapsack problems with a resource buffer. In Pinyan Lu and Guochuan Zhang, editors, *30th International Symposium on Algorithms and Computation, ISAAC 2019, December 8-11, 2019, Shanghai University of Finance and Economics, Shanghai, China*, volume 149 of *LIPIcs*, pages 28:1–28:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ISAAC.2019.28.

13 Juraj Hromkovič, Rastislav Královič, and Richard Královič. Information complexity of online problems. In Petr Hliněný and Antonín Kučera, editors, *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6281 of *Lecture Notes in Computer Science*, pages 24–36. Springer, 2010. doi:10.1007/978-3-642-15155-2_3.

14 Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, 1975. doi:10.1145/321906.321909.

15 Sandy Irani, Sandeep K. Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Trans. Algorithms*, 3(4):41, 2007. doi:10.1145/1290672.1290678.

16 Kazuo Iwama and Shiro Taketomi. Removable online knapsack problems. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan J. Eidenbenz, and Ricardo Conejo, editors, *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 293–305. Springer, 2002. doi:10.1007/3-540-45465-9_26.

17 Kazuo Iwama and Guochuan Zhang. Online knapsack with resource augmentation. *Inf. Process. Lett.*, 110(22):1016–1020, 2010. doi:10.1016/j.ipl.2010.08.013.

18 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

19 Dennis Komm. *An Introduction to Online Computation - Determinism, Randomization, Advice*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016. doi:10.1007/978-3-319-42749-2.

20 Alberto Marchetti-Spaccamela and Carlo Vercellis. Stochastic on-line knapsack problems. *Math. Program.*, 68:73–104, 1995. doi:10.1007/BF01585758.

21 Alexander Schrijver and Paul D. Seymour. Packing odd paths. *J. Comb. Theory, Ser. B*, 62(2):280–288, 1994. doi:10.1006/jctb.1994.1070.

22 Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985. doi:10.1145/2786.2793.

23 Natalia Vanetik. Path packing and a related optimization problem. *J. Comb. Optim.*, 17(2):192–205, 2009. doi:10.1007/s10878-007-9107-z.

Inapproximability of Diameter in Super-Linear Time: Beyond the 5/3 Ratio

Édouard Bonnet
Univ. Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Abstract

We show, assuming the Strong Exponential Time Hypothesis, that for every $\varepsilon > 0$, approximating directed Diameter on m-arc graphs within ratio $7/4 - \varepsilon$ requires $m^{4/3-o(1)}$ time. Our construction uses non-negative edge weights but even holds for sparse digraphs, i.e., for which the number of vertices n and the number of arcs m satisfy $m = \tilde{O}(n)$. This is the first result that conditionally rules out a near-linear time 5/3-approximation for a variant of Diameter.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Diameter, inapproximability, SETH lower bounds, k-Orthogonal Vectors

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.17

Related Version *Full Version*: https://arxiv.org/abs/2008.11315

Funding This work was supported by the grant from French National Agency under PRC program (project Digraphs, ANR-19-CE48-0013-01).

1 Introduction

The diameter of a graph is the largest length of a shortest path between two of its vertices. We denote by Diameter the algorithmic task of computing the diameter of an input graph. We will sometimes prefix Diameter by the adjectives *undirected/directed* specifying if edges can be oriented (i.e., if they can be arcs), and *unweighted/weighted* specifying if non-negative edge weights can be used. By default, we will assume that both edge orientations and edge weights are allowed. To be clear, the diameter in a digraph (or directed graph) is the maximum taken over all *ordered* pairs of vertices (u, v) of the distance from u to v. Note that it is very possible that the pair (u, v) realizes the distance of the diameter, while there is a much shorter path from v to u (perhaps just an arc).

There is an active line of work aiming to determine the best running time for an algorithm approximating (variants of) Diameter within a given ratio (see for instance the survey of Rubinstein and Vassilevska Williams [12]). We focus here on sparse graphs, for which the number of edges m and the number of vertices n verify $m = \tilde{O}(n)$, where $\tilde{O}$ suppresses the polylogarithmic factors.[1] There is an exact algorithm running in time $\tilde{O}(n^2)$ by computing n shortest-path trees from every vertex of the graph. There is also a folklore 2-approximation running in time $\tilde{O}(n)$ by computing a shortest-path tree from an arbitrary vertex and outputting the largest distance found. There are an $\tilde{O}(n^{3/2})$ time 3/2-approximation for weighted directed Diameter [1, 11, 6], and for every non-negative integer k, an $\tilde{O}(n^{1+\frac{1}{k+1}})$ time $(2 - 2^{-k})$-approximation for weighted *undirected* Diameter [4]. In dense graphs these four algorithms take time $\tilde{O}(mn)$, $\tilde{O}(m)$, $\tilde{O}(m^{3/2})$ and $\tilde{O}(mn^{\frac{1}{k+1}})$, respectively.

[1] Throughout the paper we adopt the convention that n denotes the number of vertices and m, the number of edges of a given graph.

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 17; pp. 17:1–17:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

There are two competing criteria: minimizing the approximation factor, which is in that case between 1 and 2, and minimizing the exponent of the running time, also a real number between 1 and 2. We now know that the points $(1, 2)$, $(\frac{3}{2}, \frac{3}{2})$, and $(2, 1)$ are feasible for the more general variant of Diameter. The question is whether these algorithms can be improved or if conditional lower bounds can be provided instead. The paper is about the latter, so we will now briefly present the relevant framework of fine-grained complexity, as well as its known consequences for Diameter.

Fine-grained complexity

The program of fine-grained complexity aims to match fine-grained algorithms (where the precise running time matters more than the membership to some robust complexity class) with tight conditional lower bounds under well-established assumptions. These assumptions are said *problem-centric*. They rely on the fact that we have been collectively unable to "meaningfully" improve over the brute-force or textbook algorithms for some important problems. Then perhaps such improvements are impossible, or at least they are currently out of reach. A fine-grained reduction from one of these central problems to our problem of interest Π tells us that improving on Π would result in a major breakthrough.

The three main hypotheses are based on SAT, 3-SUM, and All-Pairs Shortest-Paths. One might think that All-Pairs Shortest-Paths is a better starting point for a reduction to Diameter. Surprisingly it happens that SAT is. We will now focus on conditional lower bounds for Diameter, so we will only define the hypothesis based on SAT. For more on fine-grained complexity, we refer the interested reader to the survey of Vassilevska Williams [14].

The Strong Exponential Time Hypothesis (SETH, for short) asserts that for every $\varepsilon > 0$, there is an integer k such that k-SAT cannot be solved in time $(2-\varepsilon)^n$ on n-variable instances [8]. The first SETH-based lower bound for a polynomial-time solvable graph problem was precisely on unweighted undirected Diameter [11]. The authors show that, unless the SETH fails, any $(3/2-\delta)$-approximation for sparse unweighted undirected Diameter, with $\delta > 0$, requires time $n^{2-o(1)}$. Backurs et al. [2] show under the same assumption that, for every $k \geqslant 3$, any $((5k-7)/(3k-4)-\delta)$-approximation for sparse unweighted directed Diameter, with $\delta > 0$, requires time $n^{1+\frac{1}{k-1}-o(1)}$, and that any $(5/3-\delta)$-approximation for sparse weighted undirected Diameter requires time $n^{3/2-o(1)}$. Li [9] improves on these results showing that, unless the SETH fails, any $(5/3-\delta)$-approximation for unweighted undirected Diameter requires time $n^{3/2-o(1)}$.

Since a 5/3-approximation of Diameter running in near-linear time was consistent with the current knowledge, even in weighted directed graphs, Rubinstein and Vassilevska Williams [12] and Li [9] ask for such an algorithm or some lower bounds with a ratio closer to 2. We give an evidence that, at least for weighted directed graphs, no such algorithm is possible. More precisely, our contribution is the following.

▶ **Theorem 1.** *Unless the SETH fails, for any $\varepsilon > 0$, $(7/4-\varepsilon)$-approximating Diameter on directed n-vertex $\tilde{O}(n)$-edge graphs where all the edge weights are non-negative integers requires $n^{4/3-o(1)}$ time.*

Figure 1 summarizes what was known for directed Diameter and where Theorem 1 fits.

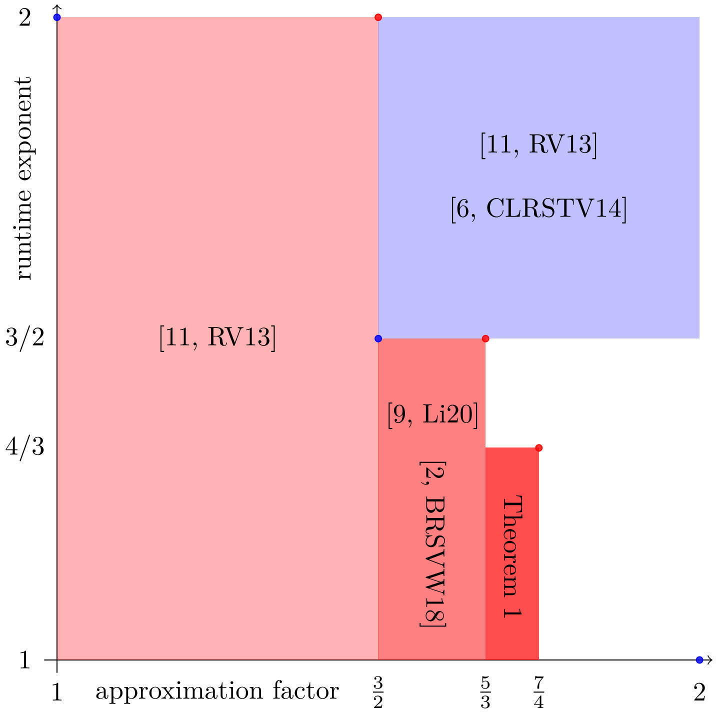


Figure 1 Approximability of sparse directed Diameter. The blue region is feasible, as witnessed by algorithms at the bottom-left corners (blue dots). The three algorithms support non-negative edge weights. The red regions would refute the SETH, as witnessed by reductions at top-right corners (red dots). The lower bounds in [11, 9] even hold for the sparse unweighted undirected Diameter, and the one in [2], for sparse weighted undirected Diameter, while Theorem 1 uses edge weights and orientations.

Techniques

Like all the Diameter lower bounds (see also [14, 12]), we reduce from k-Orthogonal Vectors. In this problem, given a set of N 0, 1-vectors of dimension ℓ, one looks for k vectors such that at every index, at least one of these k vectors has a 0 entry. Unless the SETH fails, k-Orthogonal Vectors requires time $N^{k-o(1)}$ [13], even when ℓ is polylogarithmic in N.

Here we will more precisely reduce from 4-Orthogonal Vectors (4-OV, for short). We want to build a digraph on $\tilde{O}(N^3)$ vertices and arcs, with diameter 7 if there is an orthogonal quadruple (that is, a solution to the 4-OV instance), and diameter 4 otherwise. Following a reduction to ST-Diameter[2] by Backurs et al. [2] (arguably also following [11]) all the reductions feature layers $L_0, L_1, \dots, L_{k-1}, L_k$, with only (forward) edges between two consecutive L_i. The vertices within the same layer share the same number of "vector attributes" and "index attributes". The interplay between vector and index attributes in defining the vertices and edges is adjusted so that if there are no k orthogonal vectors, then there are paths of "optimal" length k between every pair in $L_0 \times L_k$, while if there is set X of k orthogonal vectors, a pair (x, y) in $L_0 \times L_k$ jointly encoding X is far apart (usually and ideally at distance $2k - 1$).

We do not deviate too much from this strategy. Our construction is inspired from and pushes one step forward the elegant reduction of Li [9]. Here we rename L_0, L_1, L_2, L_3, L_4 by $\mathrm{ABC}, \mathrm{AB}, \mathrm{AD}_Y, \mathrm{DC}, \mathrm{DCB}$, respectively. Some pairs of vertices are too far apart on 4-OV

[2] where one seeks the length of a longest shortest path from a vertex of S to a vertex of T

NO-instances. We thus add two “gates” u and v and link them with weighted arcs to the rest of the graph. This puts many pairs of vertices at distance at most 4 regardless on whether the 4-OV instance is positive or negative.

Of course, we cannot do so for *all* the pairs outside $L_0 \times L_k$. For instance, we do not want the distance from every vertex in ABC to every vertex in AD_Y to be always at most 4. Indeed, that would make the longest path from ABC to DCB of length at most 6. Our main novel contribution is to add a vertex set AD_X (L_2') only linked to AD_Y (L_2) in a way that gives enough flexibility to make the remaining pairs sufficiently close for 4-OV NO-instances, while not decreasing the distance from x to y.

Recent developments

While the paper was under review, some exciting developments happened. Independently, Wein and Dalirrooyfard [7] and Li [10] showed that, under the SETH, for every integer $k \geqslant 2$ and real $\delta > 0$, any $(\frac{2k-1}{k} - \delta)$-approximation for sparse unweigthed directed Diameter requires time $n^{1+\frac{1}{k-1}-o(1)}$.

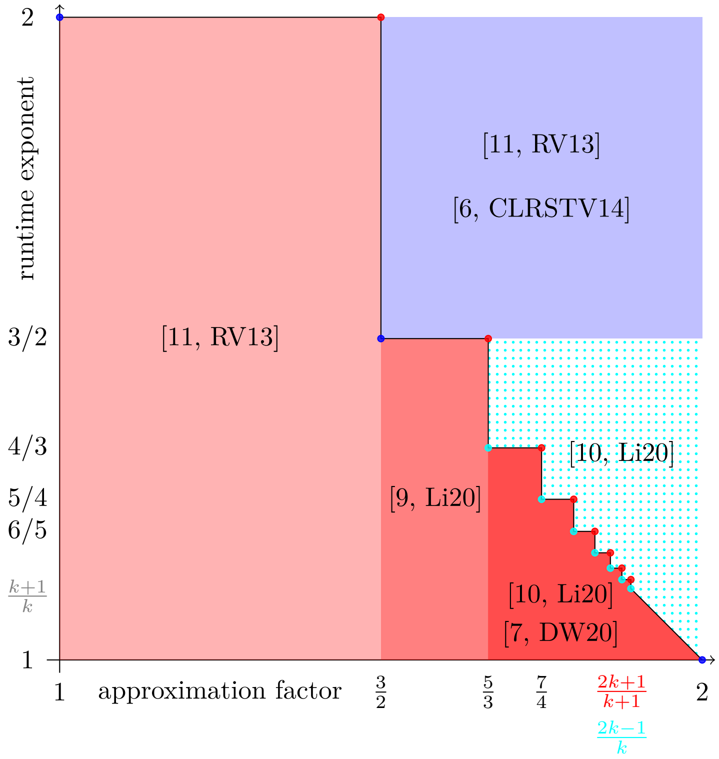


Figure 2 The new results for sparse unweigthed directed Diameter. The blue region is feasible, as witnessed by an algorithm at the bottom-left corner (blue dot). The red regions would refute the SETH, as witnessed by reductions at top-right corners (red dots). A red dot in the interior of the dotted cyan region would refute the NSETH. The lower bounds in [11, 9] even hold for the sparse unweighted undirected Diameter.

In the same paper, Li gives a piece of evidence that this could be as far as the hardness of unweigthed directed Diameter goes. This evidence is based on the NSETH (for Non-deterministic SETH), a strengthening of SETH introduced by Carmosino et al. [5]. NSETH asserts that for every $\varepsilon > 0$, there is an integer k such that the k-Taut problem cannot be solved in *non-deterministic* time $(2-\varepsilon)^n$, where k-Taut asks, given a k-DNF formula whether every truth assignment satisfies it. Li shows, for all four variants of Diameter *but* the weighted directed one that improving on any of these (deterministic) SETH lower bounds would refute the NSETH (see dotted cyan region in Figure 2).

The construction of Dalirrooyfard and Wein [7] works for $k \geqslant 5$. It makes a more intricate use of "parallel layers" (such as AD_X). For $k = 2$ and 3, the authors cite the existing lower bounds for unweighted undirected Diameter, and for $k = 4$ they tune the construction presented in this paper, in order to remove the edge weights. Namely, there is a simpler and weight-free way of connecting the "gates" u, v to the rest of the graph. The construction of Li [10] effectively combines *index-changing* "back" edges (which re-implements and extends the *skew edges* of this paper) with the usual *vector-changing* "forward" edges.

These generalizations crucially rely on edge orientations, while in the particular case of $k = 4$, our construction seems to only accidentally require arcs. In a subsequent work [3], we show how to obtain the lower bound for $k = 4$ in the most constrained case of unweigthed *undirected* Diameter. This makes a non-trivial use of additional sets of vertices without "vector fields".

Preliminaries

We use the standard graph-theoretic notations. If G is a graph, $V(G)$ denotes its vertex set. If $S \subseteq V(G)$, $G[S]$ denotes the subgraph of G induced by S, and $G - S$ is a short-hand for $G[V(G) \setminus S]$. For $u, v \in V(G)$, $d_G(u, v)$ denotes the distance from u to v in G, that is the length of a shortest path from u to v, or equivalently, the minimum sum of weights on the edges on a path from u to v. Note that, in a directed graph, $d_G(u, v)$ and $d_G(v, u)$ may well be different values. We drop the subscript, if the graph G is clear from the context. We denote by $\text{diam}(G)$ the diameter of G, that is, $\max_{u,v \in V(G)} d_G(u, v)$. Note that both the pairs (u, v) and (v, u) are considered in this maximum. If ℓ is positive integer, $[\ell]$ denotes the set $\{1, 2, \ldots, \ell\}$.

2 Reduction from 4-Orthogonal Vectors to 4 vs 7 Diameter

For every fixed positive integer k, the k-Orthogonal Vectors (k-OV for short) problem is as follows. It asks, given a set S of 0,1-vectors in $\{0, 1\}^\ell$, if there are k vectors $v_1, \ldots, v_k \in S$ such that for every $i \in [\ell]$, $\Pi_{h \in [k]} v_h[i] = 0$ or equivalently that $v_1[i] = v_2[i] = \cdots = v_k[i] = 1$ does not hold. Williams [13] showed that, assuming the SETH, k-OV requires $N^{k-o(1)}$ time with $N := |S|$. Here we will leverage this lower bound for $k = 4$. This is a usual opening step: for example, Roditty and Vassilevska Williams [11] uses this lower bound for $k = 2$, and Li [9] uses it for $k = 3$.

From any set S of N vectors in $\{0, 1\}^\ell$, we build a directed weighted graph $G := \rho(S)$ (without negatively-weighted arcs) with $O(N^3 + N^2\ell^3)$ vertices and $O(N^3\ell^3 + N^2\ell^6)$ arcs such that if S admits an orthogonal quadruple then the diameter of G is (at least) 7, whereas if S has no orthogonal quadruple then the diameter of G is (at most) 4. There is a large enough constant c such that 4-OV requires $N^{4-o(1)}$ time, unless the SETH fails, even when $\ell = c\lceil \log N \rceil$ [13]. In that case, the graph G has $O(N^3)$ vertices and $\tilde{O}(N^3)$ edges. Hence any algorithm approximating sparse, weighted, directed Diameter within ratio better than $7/4$ in time $n^{4/3-\delta}$, with $\delta > 0$, would refute the SETH.

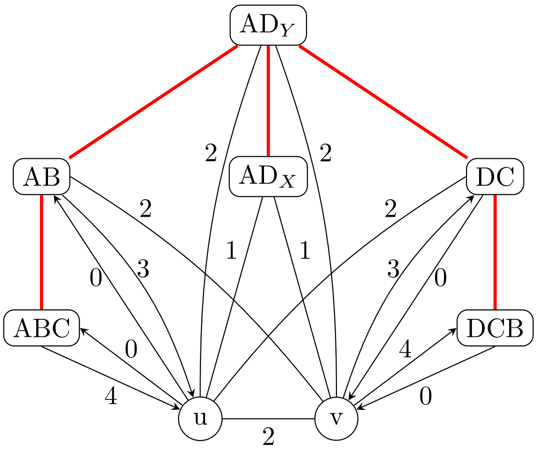


Figure 3 The part of the reduction *not* depending on the 4-OV instance. The edges represented without arrow are double-arcs with the indicated weight. The black arc between, say, ABC and u, symbolizes that every vertex of ABC is linked by an arc of weight 4 to vertex u. Thick red edges represent *some* double-arcs of weight 1. Not every double-arc (or edge) is present between two sets linked by a red edge. This will be specified in the rest of the construction.

2.1 Constant part

We start by describing the part of the construction which does *not* depend on the 4-OV instance. Its purpose is to make many pairs of vertices at distance at most 4 regardless on whether the 4-OV instance is positive or negative. The vertex set of the eventually-built graph G consists of two special vertices u and v, and six (disjoint) sets ABC, AB, AD$_X$, AD$_Y$, DC, and DCB. Vertices u and v are unconditionally linked to these sets (and to each other) by weighted arcs as specified in Figure 3. As we wrote in the introduction, there is a simpler way, that does not require edge weights, of realizing the constant part (see [7, Section 6]). We keep our construction for the sake of consistency but invite the reader to have a look at [7, Figure 9].

In this figure, a black arc between a vertex $x \in \{u, v\}$ and a set $Z \in \{\text{ABC}, \text{AB}, \text{AD}_X, \text{AD}_Y, \text{DC}, \text{DCB}\}$ (or vice versa) indicates that x is linked to *every vertex* of Z by such an arc. Note that edges represented without arrow are double-arcs. Double-arcs will sometimes simply be called *edges*. The only arcs *not* incident to $\{u, v\}$ are edges (double-arcs) of weight 1. These edges are only present between ABC and AB, AB and AD$_Y$, AD$_X$ and AD$_Y$, AD$_Y$ and DC, and finally DC and DCB. We will describe them later. At this point, one just needs to know that every vertex in ABC (resp. DCB) has at least one neighbor in AB (resp. DC), and that these edges have weight 1.

We check that many pairs of vertices are at distance at most 4 (even without the knowledge of the edges symbolized in red). For the sake of conciseness, when we write, say, "$\mathbf{u} \leftrightarrow \mathbf{DCB}$", we intend to provide paths from u to every vertex in DCB, and from every vertex in DCB to u. Similarly, the paragraph "$\mathbf{DCB} \rightarrow \mathbf{ABC}$" gives a path (of length 2) from every vertex of DCB to every vertex of ABC.

$\mathbf{u} \leftrightarrow \mathbf{v}$. There is an edge of weight 2 between u and v.

$\mathbf{u} \leftrightarrow \mathbf{Z} \in \{\text{ABC}, \text{AB}, \text{AD}_X, \text{AD}_Y, \text{DC}\}$. There are double-arcs with weight at most 4 between these pairs.

$\mathbf{u} \leftrightarrow \mathbf{DCB}$. There is a path of double-arcs of total weight 3: From a vertex of DCB, take any (weight-1) edge to DC, followed by the weight-2 edge to u. Recall that every vertex in DCB will have at least one neighbor in DC (via a weight-1 edge).

The next two cases are symmetric.

$\mathbf{v} \leftrightarrow \mathbf{Z} \in \{\mathrm{AB}, \mathrm{AD}_X, \mathrm{AD}_Y, \mathrm{DC}, \mathrm{DCB}\}$. There are double-arcs with weight at most 4 between these pairs.

$\mathbf{v} \leftrightarrow \mathbf{ABC}$. There is a path of double-arcs of total weight 3: From a vertex of ABC, take any (weight-1) edge to AB, followed by the weight-2 edge to v. Recall that every vertex in ABC will have at least one neighbor in AB (via a weight-1 edge).

So far, we have seen that for each $x \in \{u, v\}$ and $y \in V(G)$, $d(x, y) \leqslant 4$ and $d(y, x) \leqslant 4$.

For each $\mathbf{Z} \in \{\mathrm{ABC}, \mathrm{AB}, \mathrm{AD}_\mathbf{X}, \mathrm{AD}_\mathbf{Y}, \mathrm{DC}, \mathrm{DCB}\}$**,** $\mathbf{Z} \leftrightarrow \mathbf{Z}$. Each of these six sets Z has a double-arc to u or to v (or both) whose sum of weights is at most 4.

$\mathbf{AD_X} \rightarrow \mathbf{Z} \in \{\mathrm{ABC}, \mathrm{AB}, \mathrm{AD}_Y, \mathrm{DC}\}$. These pairs are at distance at most 3. There is an edge of weight 1 from every vertex of AD_X to u, and an arc of weight at most 2 from u to every vertex of $Z \in \{\mathrm{ABC}, \mathrm{AB}, \mathrm{AD}_Y, \mathrm{DC}\}$.

$\mathbf{AD_X} \rightarrow \mathbf{DCB}$. There is a path of length 4, via u and DC. Again recall that every vertex of DCB has at least one neighbor in DC (via a double-arc of weight 1).

The next two cases are symmetric.

$\mathbf{AD_X} \leftarrow \mathbf{Z} \in \{\mathrm{AB}, \mathrm{AD}_Y, \mathrm{DC}, \mathrm{DCB}\}$. These pairs are at distance at most 3. There is an arc of weight at most 2 from every vertex of $Z \in \{\mathrm{AB}, \mathrm{AD}_Y, \mathrm{DC}, \mathrm{DCB}\}$ to v, and an edge of weight 1 from v to every vertex of AD_X.

$\mathbf{AD_X} \leftarrow \mathbf{ABC}$. There is path of length 4, via AB and v.

We have now established that for every $x \in \mathrm{AD}_X$ and $y \in V(G)$, $d(x, y) \leqslant 4$ and $d(y, x) \leqslant 4$.

$\mathbf{AD_Y} \leftrightarrow \mathbf{AB}$. There is a path of two double-arcs of weight 2, via v.

$\mathbf{AD_Y} \leftrightarrow \mathbf{DC}$. There is a path of two double-arcs of weight 2, via u.

$\mathbf{AD_Y} \rightarrow \mathbf{ABC}$. There is a path of length 2 via u.

$\mathbf{AD_Y} \leftarrow \mathbf{DCB}$. There is a path of length 2 via v.

$\mathbf{ABC} \leftrightarrow \mathbf{AB}$. Via u, there is a path of length 4 from every vertex of ABC to every vertex of AB, and a path of length 3 from every vertex of AB to every vertex of ABC.

$\mathbf{DCB} \leftrightarrow \mathbf{DC}$. Via v, there is a path of length 3 from every vertex of DCB to every vertex of DC, and a path of length 4 from every vertex of DC to every vertex of DCB.

$\mathbf{DCB} \rightarrow \mathbf{ABC}$. There is a path of length 2 via v and u.

$\mathbf{DCB} \rightarrow \mathbf{AB}$. There is a path of length 2 via v.

$\mathbf{DC} \rightarrow \mathbf{Z} \in \{\mathrm{ABC}, \mathrm{AB}\}$. There is a path of length 2 via u.

To summarize, we have obtained that for every pair $x, y \in V(G)$, $d(x, y) > 4$ implies that $(x, y) \in P$ for some $P \in \{\mathrm{ABC} \times \mathrm{AD}_Y, \mathrm{ABC} \times \mathrm{DC}, \mathrm{ABC} \times \mathrm{DCB}, \mathrm{AB} \times \mathrm{DC}, \mathrm{AB} \times \mathrm{DCB}, \mathrm{AD}_Y \times \mathrm{DCB}\}$.

2.2 Variable part

We think of the vector set S as having four copies A, B, C, D with $S = A = B = C = D$. Equivalently, one looks for $a \in A$, $b \in B$, $c \in C$, and $d \in D$ such that a, b, c, d are orthogonal.

Vertex set

We first describe the vertex set $V(G) \setminus \{u, v\}$.

- For every $(a, b, c) \in A \times B \times C$, we add vertex $(a, b, c)_{\mathrm{ABC}}$ to ABC.
- Similarly for every $(d, c, b) \in D \times C \times B$, we add vertex $(d, c, b)_{\mathrm{DCB}}$ to DCB.
- For every $(a, b) \in A \times B$ and every triple $i, j, k \in [\ell]$ such that $a[i] = a[j] = a[k] = 1$ and b takes value 1 on at least two indices of $\{i, j, k\}$, we add vertex $(a, b, i, j, k)_{\mathrm{AB}}$ to AB.
- The set of vertices DC is defined analogously with D and C playing the roles of A and B (recall that actually $A = B = C = D$).
- For every $(a, d) \in A \times D$ and every triple of indices $i, j, k \in [\ell]$ such that $a[i] = a[j] = a[k] = 1$ and $d[i] = d[j] = d[k] = 1$, we add vertex $(a, d, i, j, k)_{\mathrm{AD}_Y}$ to AD_Y.
- For every $(a, d) \in A \times D$ and every $i, j, k \in [\ell]$ such that at most one of $a[i], a[j], a[k], d[i], d[j], d[k]$ is equal to 0, we add vertex $(a, d, i, j, k)_{\mathrm{AD}_X}$ to AD_X.

As observed in [7], the definition of AD_X could be simpler. We keep it as is, again for the sake of consistency.

Edge Set

We now describe the edge set on $V(G) \setminus \{u, v\}$. All these edges are double-arcs (we do not need to orient them) of weight 1 (we do not need to put weights).

- We add an edge (double-arc) of weight 1 between every pair $(a, b, c)_{\mathrm{ABC}}$ and $(a, b, i, j, k)_{\mathrm{AB}}$ if c takes value 1 on at least one index of $\{i, j, k\}$ where b also takes value 1. In our construction, the existence of an edge is implicitly conditional to the existence of both of its endpoints. The edge exists only if $(a, b, i, j, k)_{\mathrm{AB}}$ is indeed a vertex of AB. The edges between DCB and DC are defined similarly.
- We add every edge between $(a, b, i, j, k)_{\mathrm{AB}}$ and $(a, b, i', j', k')_{\mathrm{AB}}$, with $a \in A$, $b \in B$, and $i, j, k, i', j', k' \in [\ell]$. Similarly we add every edge between $(d, c, i, j, k)_{\mathrm{DC}}$ and $(d, c, i', j', k')_{\mathrm{DC}}$, with $d \in D$, $c \in C$, and $i, j, k, i', j', k' \in [\ell]$. We call these edges *index-switching*. AB and DC are the only two sets among $\{\mathrm{ABC}, \mathrm{AB}, \mathrm{AD}_X, \mathrm{AD}_Y, \mathrm{DC}, \mathrm{DCB}\}$ which are *not* independent sets.
- We link every pair $(a, b, i, j, k)_{\mathrm{AB}}$ and $(a, d, i, j, k)_{\mathrm{AD}_Y}$ by an edge, as well as every pair $(a, d, i, j, k)_{\mathrm{AD}_Y}$ and $(d, c, i, j, k)_{\mathrm{DC}}$.
- We add an edge between every pair $(a, d, i, j, k)_{\mathrm{AD}_X}$ and $(a, d', i, j, k)_{\mathrm{AD}_Y}$ (such that $a \in A$, $d \neq d' \in D$, and $i, j, k \in [\ell]$), and $(a, d, i, j, k)_{\mathrm{AD}_X}$ and $(a', d, i, j, k)_{\mathrm{AD}_Y}$ (such that $a \neq a' \in A$, $d \in D$, and $i, j, k \in [\ell]$).
- Finally we add an edge between every pair $(a, d, i, j, k)_{\mathrm{AD}_X}$ and $(a, d, i', j', k')_{\mathrm{AD}_Y}$ with $a \in A$, $d \in D$, and $i, j, k, i', j', k' \in [\ell]$. We call this type of edge *skew*. Skew edges are the only way to change the indices while also moving from one set to another (it is not internal to AB or to DC).

All the edges defined in this section that are not index-switching or skew are called *regular*. This ends the construction of $G = \rho(S)$. See Figure 4 for an illustration.

Simplified vertex notations, vector fields, and index fields

Henceforth we will drop the indices in the vertex labels. The set a vertex belongs to will be implicit by the choice of the variable labels. For instance (a, b', i, j, k) is in AB, and (d', c, b') is in DCB. This does not allow us to distinguish vertices of AD_X and AD_Y. We will denote

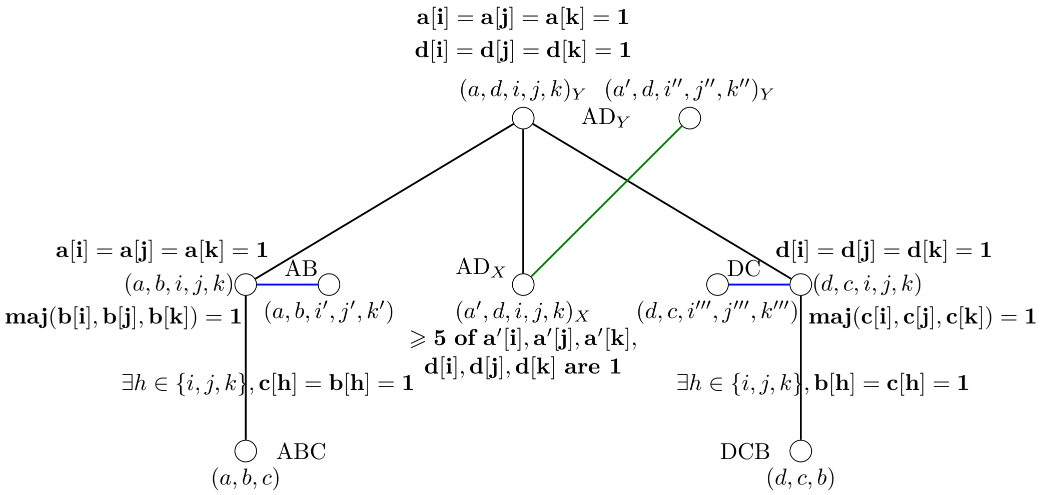

$\mathbf{a[i] = a[j] = a[k] = 1}$
$\mathbf{d[i] = d[j] = d[k] = 1}$
$(a, d, i, j, k)_Y$
$(a', d, i'', j'', k'')_Y$
AD_Y
$\mathbf{a[i] = a[j] = a[k] = 1}$
(a, b, i, j, k)
AB
(a, b, i', j', k')
$\mathbf{maj(b[i], b[j], b[k]) = 1}$
AD_X
$(a', d, i, j, k)_X$
$\mathbf{\geqslant 5\ of\ a'[i], a'[j], a'[k], d[i], d[j], d[k]\ are\ 1}$
DC
$\mathbf{d[i] = d[j] = d[k] = 1}$
(d, c, i, j, k)
(d, c, i''', j''', k''')
$\mathbf{maj(c[i], c[j], c[k]) = 1}$
$\exists h \in \{i, j, k\}, \mathbf{c[h] = b[h] = 1}$
$\exists h \in \{i, j, k\}, \mathbf{b[h] = c[h] = 1}$
ABC
(a, b, c)
DCB
(d, c, b)

Figure 4 The rules for the existence of vertices and edges in $G - \{u, v\}$. We removed the subscripts in the vertex labels as discussed in the second-to-last paragraph of Section 2.2. Conditions to the existence of a vertex appear in bold next to the vertex. Conditions to the existence of an edge appear in bold along the edge. All edges (double-arcs) have weight 1, so we omit their weight. Regular edges are represented in black. Index-switching edges are represented in blue (they are only present in AB and DC). Skew edges are represented in green (they are only present between AD_X and AD_Y. Note that the regular edge $(a, d, i, j, k)_Y (a', d, i, j, k)_X$ could also be of the form $(a, d, i, j, k)_Y (a, d', i, j, k)_X$.

by $(a, d, i, j, k)_Y = (a, d, i, j, k)_{\mathrm{AD}_Y}$ a vertex in AD_Y and by $(a, d, i, j, k)_X$ the "same" vertex in AD_X. Note that it is possible that $(a, d, i, j, k)_X$ exists but not $(a, d, i, j, k)_Y$, if exactly five of $a[i], a[j], a[k], d[i], d[j], d[k]$ are equal to 1.

We call *vector fields* the first three coordinates of every vertex in $\mathrm{ABC} \cup \mathrm{DCB}$, and the first two coordinates of every vertex in $\mathrm{AB} \cup \mathrm{AD}_X \cup \mathrm{AD}_Y \cup \mathrm{DC}$. We call *index fields* the last three coordinates of every vertex in $\mathrm{AB} \cup \mathrm{AD}_X \cup \mathrm{AD}_Y \cup \mathrm{DC}$. We can assume that the 4-OV instance does not have an orthogonal *triple* (this can be checked in time $\tilde{O}(N^3)$). Thus every vertex $(a, b, c) \in \mathrm{ABC}$ (resp. $(d, c, b) \in \mathrm{DCB}$) indeed has at least one neighbor in AB (resp. DC), namely (a, b, i, i, i) (resp. (d, c, i, i, i)) where $i \in [\ell]$ is such that $a[i] = b[i] = c[i] = 1$ (resp. $d[i] = c[i] = b[i] = 1$).

Vertex and edge count

Before tackling the correctness of the reduction, we check that G has $O(N^3)$ vertices and $\tilde{O}(N^3)$ arcs. The number of vertices of G is bounded by $2 + 2 \cdot N^3 + 4 \cdot N^2 \ell^3 = O(N^3)$ since $\ell = O(\log N)$. The number of arcs of G is bounded by $4 \cdot O(N^3) + 4 \cdot N^2 \ell^6 + 10 \cdot N^3 \ell^3 + 2 \cdot N^2 \ell^6 = \tilde{O}(N^3)$, where the first term accounts for the arcs incident with $\{u, v\}$, the second for the index-switching arcs, the third for the regular arcs, and the fourth for the skew arcs.

2.3 No orthogonal quadruple implies diameter at most 4

We exhibit in this section short paths (of length at most 4) between every pair of vertices of G. For that we extensively use that, as there is no orthogonal quadruple in S, for every $u, v, w, x \in S$, $\mathrm{ind}(u, v, w, x) := \min\{i \in [\ell] \mid u[i] = v[i] = w[i] = x[i] = 1\}$ is a well-defined index in $[\ell]$. We only take the minimum index to have a deterministic notation. There will not be anything particular with the minimum, and any index of the non-empty $\{i \in [\ell] \mid u[i] = v[i] = w[i] = x[i] = 1\}$ would work as well. We will also use $\text{ind}(u, v, w)$ as a short-hand for $\text{ind}(u, v, w, w)$.

In Section 2.1, we have reduced the task of showing that $\text{diam}(G) \leqslant 4$ to considering only six pairs of sets.

ABC → AD$_\mathbf{Y}$. Let (a, b, c) and $(a', d, i', j', k')_Y$ be two vertices in ABC and AD$_Y$ respectively. We define the indices $i := \text{ind}(a, b, c, d)$, $j := \text{ind}(a, a', b, d)$, and $k := \text{ind}(a, a', d)$. Then, $(a, b, c) \to (a, b, i, j, k) \to (a, d, i, j, k)_Y \to (a', d, i, j, k)_X \to (a', d, i', j', k')_Y$ is a path of length 4 in G.

We first justify the existence of the inner vertices of this path (i.e., all but the endpoints). Indeed the endpoints exist by assumption. Vertex $(a, b, i, j, k) \in \text{AB}$ is present in G since $a[i] = a[j] = a[k] = 1$, and $b[i] = b[j] = 1$. Vertex $(a, d, i, j, k)_Y \in \text{AD}_Y$ exists since $a[i] = a[j] = a[k] = 1 = d[i] = d[j] = d[k]$. Finally $(a', d, i, j, k)_X \in \text{AD}_X$ is indeed a vertex of G since $a'[j] = a'[k] = 1$ and $d[i] = d[j] = d[k] = 1$. Recall that in AD$_X$ (contrary to AD$_Y$) it is fine if at most one of the six values obtained by evaluating one of the two vectors at one of the three indices is 0.

We now justify the existence of the edges. The arc $(a, b, c) \to (a, b, i, j, k)$ exists since $c[i] = b[i] = 1$ (and both its endpoints exist). The arcs $(a, b, i, j, k) \to (a, d, i, j, k)_Y$ and $(a, d, i, j, k)_Y \to (a', d, i, j, k)_X$ are regular edges of G: one vector field and the three index fields remain unchanged. Finally the arc $(a', d, i, j, k)_X \to (a', d, i', j', k')_Y$ is a skew edge of G: it is between AD$_X$ and AD$_Y$, and both vector fields remain the same (while the indices are allowed to change).

ABC → DC. Let (a, b, c) and (d, c', i', j', k') be two vertices in ABC and DC respectively. We define the indices $i := \text{ind}(a, b, c, d)$, $j := \text{ind}(a, b, c', d)$, and $k := \text{ind}(a, c', d)$. Then, $(a, b, c) \to (a, b, i, j, k) \to (a, d, i, j, k)_Y \to (d, c', i, j, k) \to (d, c', i', j', k')$ is a path of length 4 in G.

As in the previous case, the existence of vertices $(a, b, i, j, k) \in \text{AB}$, $(a, d, i, j, k)_Y \in \text{AD}_Y$, $(d, c', i, j, k) \in \text{DC}$ is ensured by the fact that $a[i] = a[j] = a[k] = 1 = d[i] = d[j] = d[k]$ and $b[i] = b[j] = 1 = c'[j] = c'[k]$. The arc $(a, b, c) \to (a, b, i, j, k)$ exists since $c[i] = b[i] = 1$, and the arcs $(a, b, i, j, k) \to (a, d, i, j, k)_Y \to (d, c', i, j, k)$ are two (existing) regular arcs. Finally $(d, c', i, j, k) \to (d, c', i', j', k')$ is an index-switching arc internal to DC (note that the two vector fields remain the same, as they should).

ABC → DCB. Let (a, b, c) and (d, c', b') be two vertices in ABC and DCB respectively. We define the indices $i := \text{ind}(a, b, c, d)$, $j := \text{ind}(a, b, c', d)$, and $k := \text{ind}(a, b', c', d)$. Then, $(a, b, c) \to (a, b, i, j, k) \to (a, d, i, j, k)_Y \to (d, c', i, j, k) \to (d, c', b')$ is a path of length 4 in G.

The vertices $(a, b, i, j, k) \in \text{AB}$, $(a, d, i, j, k)_Y \in \text{AD}_Y$, $(d, c', i, j, k) \in \text{DC}$ exist since $a[i] = a[j] = a[k] = 1 = d[i] = d[j] = d[k]$ and $b[i] = b[j] = 1 = c'[j] = c'[k]$. The arc $(a, b, c) \to (a, b, i, j, k)$ is in G since $c[i] = b[i] = 1$. The arcs $(a, b, i, j, k) \to (a, d, i, j, k)_Y \to (d, c', i, j, k)$ are two regular arcs in G. The arc $(d, c', i, j, k) \to (d, c', b')$ exists since $b'[k] = c'[k] = 1$.

AB → DC. Let (a, b, i', j', k') and (d, c, i'', j'', k'') be two vertices in AB and DC respectively. We define the index $i := \text{ind}(a, b, c, d)$. Then, $(a, b, i', j', k') \to (a, b, i, i, i) \to (a, d, i, i, i)_Y \to (d, c, i, i, i) \to (d, c, i'', j'', k'')$ is a path of length 4 in G.

Vertices $(a, b, i, i, i) \in \text{AB}$, $(a, d, i, i, i)_Y \in \text{AD}_Y$, $(d, c, i, i, i) \in \text{DC}$ exist since $a[i] = b[i] = c[i] = d[i] = 1$. The arc $(a, b, i', j', k') \to (a, b, i, i, i)$ is a legal index-switching arc, internal to AB. The arcs $(a, b, i, i, i) \to (a, d, i, i, i)_Y \to (d, c, i, i, i)$ are two regular arcs in G. Finally the arc $(d, c, i, i, i) \to (d, c, i'', j'', k'')$ is an index-switching arc, internal to DC.

The next two cases are symmetric to $\mathbf{ABC} \to \mathbf{DC}$ and $\mathbf{ABC} \to \mathbf{AD_Y}$, respectively. We spell them out since it is not much longer that making the symmetry explicit.

AB → DCB. Let (a, b, i', j', k') and (d, c, b') be two vertices in AB and DCB respectively. We define the indices $i := \text{ind}(a, b, c, d)$, $j := \text{ind}(a, b, d)$, and $k := \text{ind}(a, b', c, d)$. Then, $(a, b, i', j', k') \to (a, b, i, j, k) \to (a, d, i, j, k)_Y \to (d, c, i, j, k) \to (d, c, b')$ is a path of length 4 in G.

Vertices $(a, b, i, j, k) \in \text{AB}$, $(a, d, i, j, k)_Y \in \text{AD}_Y$, $(d, c, i, j, k) \in \text{DC}$ exist since $a[i] = a[j] = a[k] = 1 = d[i] = d[j] = d[k]$ and $b[i] = b[j] = 1 = c[i] = c[k]$. The arc $(a, b, i', j', k') \to (a, b, i, j, k)$ is index-switching in AB. The arcs $(a, b, i, j, k) \to (a, d, i, j, k)_Y \to (d, c, i, j, k)$ are two regular arcs present in G. Finally the arc $(d, c, i, j, k) \to (d, c, b')$ exists since $b'[k] = c[k] = 1$.

AD$_\mathbf{Y}$ → DCB. Let $(a, d, i', j', k')_Y$ and (d', c, b) be two vertices in AD_Y and DCB respectively. We define the indices $i := \text{ind}(a, b, c, d')$, $j := \text{ind}(a, c, d, d')$, and $k := \text{ind}(a, d, d')$. Then, $(a, d, i', j', k')_Y \to (a, d, i, j, k)_X \to (a, d', i, j, k)_Y \to (d', c, i, j, k) \to (d', c, b)$ is a path of length 4 in G.

The vertices $(a, d, i, j, k)_X \in \text{AD}_X$, $(a, d', i, j, k)_Y \in \text{AD}_Y$, $(d', c, i, j, k) \in \text{DC}$ are present in G since $a[i] = a[j] = a[k] = 1 = d'[i] = d'[j] = d'[k]$ and $d[j] = d[k] = 1 = c[i] = c[j]$. Recall that $a[i] = a[j] = a[k] = 1 = d[j] = d[k]$ suffices for the existence of $(a, d, i, j, k)_X$ (but not for the one of $(a, d, i, j, k)_Y$). The arc $(a, d, i', j', k')_Y \to (a, d, i, j, k)_X$ is a skew arc: it is between AD_Y and AD_X, and the two vector fields remain unchanged. The arcs $(a, d, i, j, k)_X \to (a, d', i, j, k)_Y \to (d', c, i, j, k)$ are two regular arcs of G. Finally the arc $(d', c, i, j, k) \to (d', c, b)$ exists since $b[i] = c[i] = 1$.

We have proved that there is a path of length at most 4 between every (ordered) pair of vertices in G, when there is no orthogonal quadruple. Thus the diameter of G is then (at most) 4.

2.4 An orthogonal quadruple implies two vertices at distance at least 7

We now suppose that S admits at least one orthogonal quadruple, say, a, b, c, d. We show that G has diameter at least 7, by arguing that there is no path of length at most 6 from $(a, b, c) \in \text{ABC}$ to $(d, c, b) \in \text{DCB}$.

The first observation is that there is no path of length at most 6 from (a, b, c) to (d, c, b) intersecting $\{u, v\}$. Indeed one can check that $d((a, b, c), u) = 4$ and $d(u, (d, c, b)) = 3$, and that $d((a, b, c), v) = 3$ and $d(v, (d, c, b)) = 4$. We can now rule out the existence of a path P of length 6 from (a, b, c) and (d, c, b) in $G - \{u, v\}$.

We distinguish two cases:

- (a) P does not intersect AD_X, or
- (b) P intersects AD_X.

Case (a). We further distinguish two cases: either (a1) P contains no index-switching arc, or (a2) P contains at least one index-switching arc. In case (a1), the three index fields cannot change at all in P (recall that the skew edges are between AD_X and AD_Y). Thus the first and penultimate vertices of P are $(a, b, i, j, k) \in \text{AB}$ and $(d, c, i, j, k) \in \text{DC}$ for some $i, j, k \in [\ell]$. The existence of these vertices imply that $a[i] = a[j] = a[k] = 1 = d[i] = d[j] = d[k]$, and vectors b and c both take value 1 on at least two indices among $\{i, j, k\}$. Therefore there exists an index $h \in \{i, j, k\}$ such that $a[h] = b[h] = c[h] = d[h] = 1$. This contradicts the fact that a, b, c, d are orthogonal.

We now tackle case (a2). Since the removal of AD_Y separates $\mathrm{ABC} \cup \mathrm{AB}$ from $\mathrm{DCB} \cup \mathrm{DC}$ in $G - \{u, v\}$, $G[\mathrm{AD}_Y]$ is an independent set, and there are no edges between AD_Y and $\mathrm{ABC} \cup \mathrm{DCB}$, such a path P has to contain a subpath $x \to y \to z$ with $x \in \mathrm{AB}$, $y \in \mathrm{AD}_Y$, and $z \in \mathrm{DC}$. As P contains at least one index-switching arc, it cannot also contains a back-and-forth along $\mathrm{AB} \to \mathrm{ABC} \to \mathrm{AB}$, $\mathrm{AD}_Y \to \mathrm{AB} \to \mathrm{AD}_Y$, $\mathrm{DC} \to \mathrm{AD}_Y \to \mathrm{DC}$, or $\mathrm{DCB} \to \mathrm{DC} \to \mathrm{DCB}$. Indeed that would amount to at least three additional arcs (at least one index-switching plus two for the back-and-forth) to the mandatory four arcs $\mathrm{ABC} \to \mathrm{AB} \to \mathrm{AD}_Y \to \mathrm{DC} \to \mathrm{DCB}$, hence a path of length at least 7. Therefore, there are three indices $i, j, k \in [\ell]$ such that $x = (a, b, i, j, k)$, $y = (a, d, i, j, k)_Y$, and $z = (d, c, i, j, k)$. Indeed every path in $G[\mathrm{ABC} \cup \mathrm{AB}]$ and every path in $G[\mathrm{DCB} \cup \mathrm{DC}]$ preserve the first two vector fields. Again the existence of (a, b, i, j, k) (in AB) and (d, c, i, j, k) (in DC) contradicts that a, b, c, d are orthogonal.

Case (b). We can now assume that P intersects AD_X. Thus P has length exactly 6 and is of the form $(a, b, c) \in \mathrm{ABC} \to \mathrm{AB} \to \mathrm{AD}_Y \to \mathrm{AD}_X \to \mathrm{AD}_Y \to \mathrm{DC} \to \mathrm{DCB} \ni (d, c, b)$. In particular, P cannot contain an index-switching edge. If P contains no skew edge too, the index fields cannot change. So the second and sixth vertices of P are some $(a, b, i, j, k) \in \mathrm{AB}$ and $(d, c, i, j, k) \in \mathrm{DC}$, and we can conclude as in case (a).

Thus P has to contain at least one skew edge. Let us show that P has to contain exactly one skew edge. We recall that the skew edges are only present between AD_X and AD_Y.

We first argue that the third vertex of P is $(a, d', i, j, k)_Y$ for some $d' \neq d \in D$ and $i, j, k \in [\ell]$. The first vector field cannot change in a path of the form $\mathrm{ABC} \to \mathrm{AB} \to \mathrm{AD}_Y$, so we only have to show that d' cannot be equal to d. Indeed, otherwise $a[i] = a[j] = a[k] = 1 = d[i] = d[j] = d[k]$ by the existence of $(a, d, i, j, k)_Y$. Furthermore, the existence of the arc $(a, b, c) \to (a, b, i, j, k)$ (which has to be the first arc of P) implies that there is an index $h \in \{i, j, k\}$ such that $b[h] = c[h] = 1$. This index thus contradicts the orthogonality of a, b, c, d.

As the third vertex of P is $(a, d', i, j, k)_Y$ with $d' \neq d$, two skew edges $\mathrm{AD}_Y \to \mathrm{AD}_X \to \mathrm{AD}_Y$ would lead to a vertex $(a, d', i', j', k')_Y$. This latter vertex is linked in DC to vertices of the form (d', c', i', j', k') (where c' can be c). This cannot lead to (d, c, b) since the first vector field does not change in an arc from DC to DCB.

We have established that from vertex $(a, d', i, j, k)_Y$, P takes exactly one skew edge, either from AD_Y to AD_X, or from AD_X to AD_Y. In both cases, by the previous remark, the second vector field of the fifth vertex of P should be d. This implies that the fifth vertex of P is of the form $(a, d, i', j', k')_Y$ for some indices $i', j', k' \in [\ell]$. Indeed the skew edge of P in $\mathrm{AD}_Y \to \mathrm{AD}_X \to \mathrm{AD}_Y$ preserves both vector fields, whereas the regular edge of P in $\mathrm{AD}_Y \to \mathrm{AD}_X \to \mathrm{AD}_Y$ can only change one vector field, and has to change d' ($\neq d$) to d.

The end of P is thus $(a, d, i', j', k')_Y \to (d, c, i', j', k') \to (d, c, b)$, since the first two vector fields cannot be changed by an arc from DC to DCB. The existence of vertex $(a, d, i', j', k')_Y$ implies that $a[i'] = a[j'] = a[k'] = 1 = d[i'] = d[j'] = d[k']$. The existence of the arc $(d, c, i', j', k') \to (d, c, b)$ implies that there is an index $h \in \{i', j', k'\}$ such that $c[h] = b[h] = 1$. This yields $a[h] = b[h] = c[h] = d[h] = 1$, contradicting the orthogonality of a, b, c, d.

We have ruled out the existence of a path in G of length at most 6 between (a, b, c) and (d, c, b), when a, b, c, d are orthogonal. Hence the diameter of G is at least 7 when there is an orthogonal quadruple.

References

1 Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast Estimation of Diameter and Shortest Paths (Without Matrix Multiplication). *SIAM J. Comput.*, 28(4):1167–1181, 1999. doi:10.1137/S0097539796303421.

2 Arturs Backurs, Liam Roditty, Gilad Segal, Virginia Vassilevska Williams, and Nicole Wein. Towards tight approximation bounds for graph diameter and eccentricities. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 267–280. ACM, 2018. doi:10.1145/3188745.3188950.

3 Édouard Bonnet. 4 vs 7 sparse undirected unweighted Diameter is SETH-hard at time $n^{4/3}$, 2021. arXiv:2101.02312.

4 Massimo Cairo, Roberto Grossi, and Romeo Rizzi. New Bounds for Approximating Extremal Distances in Undirected Graphs. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 363–376. SIAM, 2016. doi:10.1137/1.9781611974331.ch27.

5 Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 261–270. ACM, 2016. doi:10.1145/2840728.2840746.

6 Shiri Chechik, Daniel H. Larkin, Liam Roditty, Grant Schoenebeck, Robert Endre Tarjan, and Virginia Vassilevska Williams. Better Approximation Algorithms for the Graph Diameter. 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 1041–1052. SIAM, 2014. doi:10.1137/1.9781611973402.78.

7 Mina Dalirrooyfard and Nicole Wein. Tight Conditional Lower Bounds for Approximating Diameter in Directed Graphs. *CoRR*, abs/2011.03892, 2020. arXiv:2011.03892.

8 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which Problems Have Strongly Exponential Complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

9 Ray Li. Improved SETH-hardness of unweighted Diameter. *CoRR*, abs/2008.05106, 2020. arXiv:2008.05106.

10 Ray Li. Settling SETH vs. Approximate Sparse Directed Unweighted Diameter (up to (NU)NSETH). *CoRR*, abs/2008.05106, 2020. arXiv:2008.05106.

11 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 515–524. ACM, 2013. doi:10.1145/2488608.2488673.

12 Aviad Rubinstein and Virginia Vassilevska Williams. SETH vs Approximation. *SIGACT News*, 50(4):57–76, 2019. doi:10.1145/3374857.3374870.

13 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.

14 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, volume 3, pages 3431–3472. World Scientific, 2018.

The Approximation Ratio of the 2-Opt Heuristic for the Euclidean Traveling Salesman Problem

Ulrich A. Brodowsky
Pontsheide 20, 52076 Aachen, Germany

Stefan Hougardy[1]
Research Institute for Discrete Mathematics, Universität Bonn, Germany

Abstract

The 2-Opt heuristic is a simple improvement heuristic for the Traveling Salesman Problem. It starts with an arbitrary tour and then repeatedly replaces two edges of the tour by two other edges, as long as this yields a shorter tour. We will prove that for Euclidean Traveling Salesman Problems with n cities the approximation ratio of the 2-Opt heuristic is $\Theta(\log n / \log\log n)$. This improves the upper bound of $O(\log n)$ given by Chandra, Karloff, and Tovey [3] in 1999.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases traveling salesman problem, metric TSP, Euclidean TSP, 2-Opt, approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.18

Funding *Stefan Hougardy*: funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2047/1 – 390685813.

Acknowledgements We thank the anonymous referees for several useful comments.

1 Introduction

The Traveling Salesman Problem (TSP) is one of the best studied problems in combinatorial optimization. Given n cities and their pairwise distances, the task is to find a shortest tour that visits each city exactly once. This problem is NP-hard [7] and it is even hard to approximate to a factor that is polynomial in n [13].

In the *Euclidean TSP*, the cities are points in $\mathbb{R}^2$ and the distance function is the Euclidean distance between the points. The Euclidean TSP is also NP-hard [12] but it allows a polynomial time approximation scheme [1, 11]. Euclidean Traveling Salesman Problems often appear in practice and they are usually solved using some heuristics. One of the simplest of these heuristics is the *2-Opt heuristic*. It starts with an arbitrary tour and then repeatedly replaces two edges of the tour by two other edges, as long as this yields a shorter tour. The 2-Opt heuristic stops when no further improvement can be made this way. A tour that the 2-Opt heuristic cannot improve is called *2-optimal*.

On real-world instances the 2-Opt heuristic achieves surprisingly good results (see e.g. Bentley [2]). Despite its simplicity the exact approximation ratio of the 2-Opt heuristic for Euclidean TSP is not known. In 1999, Chandra, Karloff, and Tovey [3] proved a lower bound of $c \cdot \frac{\log n}{\log\log n}$ for some constant $c > 0$ and an upper bound of $O(\log n)$ on the approximation ratio of the 2-Opt heuristic for Euclidean TSP. This leaves a gap of factor $O(\log\log n)$ between the best known upper and lower bound for the approximation ratio of the 2-Opt heuristic for Euclidean TSP. Our main result closes this gap up to a constant factor:

[1] corresponding author

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 18; pp. 18:1–18:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

▶ **Theorem 1.** *The approximation ratio of the 2-Opt heuristic for Euclidean TSP instances with n points is $\Theta(\log n/\log\log n)$.*

Related Results. On real-world Euclidean TSP instances it has been observed that the 2-Opt heuristic needs a sub-quadratic number of iterations until it reaches a local optimum [2]. However, there exist worst-case Euclidean TSP instances for which the 2-Opt heuristic may need an exponential number of iterations [5].

For n points embedded into the d-dimensional Euclidean space $\mathbb{R}^d$ for some constant $d > 2$ the approximation ratio of the 2-Opt heuristic is bounded by $O(\log n)$ from above [3] and by $\Omega(\log n/\log\log n)$ from below [15].

The Euclidean TSP is a special case of the *metric TSP*, i.e., the Traveling Salesman Problem where the distance function satisfies the triangle inequality. The well-known algorithm of Christofides [4] and Serdjukov [14] achieves an approximation ratio of $3/2$ for the metric TSP. If one allows randomization then the recent algorithm of Karlin, Klein, and Oveis Gharan [9] slightly improves on this. For the metric TSP the 2-Opt heuristic has approximation ratio exactly $\sqrt{n/2}$ [8]. A very special case of the metric TSP is the 1-2-TSP. In this version all edge lengths have to be 1 or 2. For the 1-2-TSP the approximation ratio of the 2-Opt heuristic is $3/2$ [10].

For a constant $k > 2$ the 2-Opt heuristic naturally extends to the so called *k-Opt heuristic* where in each iteration k edges of a tour are replaced by k other edges. For Euclidean TSP Zhong [15] has shown that $\Omega(\log n/\log\log n)$ is a lower bound for the k-Opt heuristic if k is constant. Therefore, Theorem 1 immediately implies:

▶ **Corollary 2.** *For constant k the approximation ratio of the k-Opt heuristic for Euclidean TSP instances with n points is $\Theta(\log n/\log\log n)$.*

Organization of the paper. The result of Chandra, Karloff, and Tovey [3, Theorem 4.4] mentioned above shows that $\Omega(\log n/\log\log n)$ is a lower bound for the 2-Opt heuristic for Euclidean TSP. To prove Theorem 1 it remains to prove the upper bound $O(\log n/\log\log n)$. We proceed as follows. First we will present in Section 2 some properties of Euclidean 2-optimal tours. In Section 2.1 we will prove Theorem 1 by reducing it to the special case where no intersections between the edges of an optimal tour and the edges of a 2-optimal tour exist. In this special case we will show that we can partition the edge set of a 2-optimal tour into five sets that are each in some sense orientation-preserving with respect to an optimal tour. The main step then is to prove that for each of these five sets we can bound the total edge length by $O(\log n/\log\log n)$ times the length of an optimal tour. To achieve this we will relate optimal tours and subsets of the edge set of a 2-optimal tour to some weighted arborescences. This relation is studied in Section 3. For weighted arborescences we will provide in Section 4 some bounds for the edge weights. These results then will allow us in Section 5 to finish the proof of Theorem 1.

2 Euclidean TSP and 2-Optimal Tours

An instance of the Euclidean TSP is a finite subset $V \subset \mathbb{R}^2$. The task is to find a polygon of shortest total edge length that contains all points of V. Note that by our definition a Euclidean TSP instance cannot contain the same point multiple times. In the following we will denote the cardinality of V by n.

For our purpose it is often more convenient to state the Euclidean Traveling Salesman Problem as a problem on graphs. For a given point set V of a Euclidean TSP instance we take a complete graph on the vertex set V, i.e., the graph $G = (V, E)$ where E is the set of

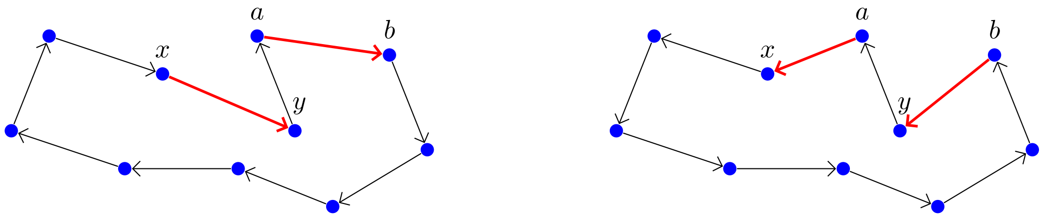


Figure 1 An oriented TSP tour (left) and the tour obtained after replacing the edges (a, b) and (x, y) with the edges (a, x) and (b, y) (right). The orientation of the tour segment between the vertices b and x has been reversed in the new tour.

all $\frac{1}{2}n(n-1)$ possible edges on V. We assign the Euclidean distance between the vertices in G by a function $c : E(G) \to \mathbb{R}_{>0}$. A *tour* in G is a cycle that contains all the vertices of G. The *length* of a tour T in G is defined as $c(T) := \sum_{e\in E(T)} c(e)$. An *optimal tour* is a tour of minimum length among the tours in G. Thus we can restate the Euclidean TSP as a problem in graphs: Given a complete graph $G = (V, E)$ on a point set $V \subset \mathbb{R}^2$ and a Euclidean distance function $c : E(G) \to \mathbb{R}_{>0}$, find an optimal tour in G. Throughout this paper we will use the geometric definition of the Euclidean TSP and the graph-theoretic version of the Euclidean TSP simultaneously. Thus, a tour for a Euclidean TSP instance $V \subseteq \mathbb{R}^2$ can be viewed as a polygon in $\mathbb{R}^2$ as well as a cycle in a complete graph on the vertex set V with Euclidean distance function.

Let $c : E(G) \to \mathbb{R}_{>0}$ be a weight function for the edges of some graph $G = (V, E)$. To simplify notation, we will denote the weight of an edge $\{x, y\} \in E(G)$ simply by $c(x, y)$ instead of the more cumbersome notation $c(\{x, y\})$. For subsets $F \subseteq E(G)$ we define $c(F) := \sum_{e\in F} c(e)$. We extend this definition to subgraphs H of G by setting $c(H) := c(E(H))$.

The distance function c of a Euclidean TSP instance $G = (V, E)$ satisfies the triangle inequality. Therefore we have for any set of three vertices $x, y, z \in V(G)$:

$$c(x, y) \;+\; c(y, z) \;\geq\; c(x, z). \tag{1}$$

The 2-Opt heuristic repeatedly replaces two edges from the tour by two other edges such that the resulting tour is shorter. Given a tour T and two edges $\{a, b\}$ and $\{x, y\}$ in T, there are two possibilities to replace these two edges by two other edges. Either we can choose the pair $\{a, x\}$ and $\{b, y\}$ or we can choose the pair $\{a, y\}$ and $\{b, x\}$. Exactly one of these two pairs will result in a tour again. Without knowing the other edges of T, we cannot decide which of the two possibilities is the correct one. Therefore, we will assume in the following that the tour T is an *oriented* cycle, i.e., the edges of T have an orientation such that each vertex has exactly one incoming and one outgoing edge. Using this convention, there is only one possibility to exchange a pair of edges such that the new edge set is a tour again: two directed edges (a, b) and (x, y) have to be replaced by the edges (a, x) and (b, y). Note that to obtain an oriented cycle again, one has to reverse the direction of the segment between b and x, see Figure 1.

A TSP tour T is called *2-optimal* if for any two edges (a, b) and (x, y) of T we have

$$c(a, x) + c(b, y) \;\geq\; c(a, b) + c(x, y). \tag{2}$$

We call inequality (2) the *2-optimality condition*.

If (a, b) and (x, y) are two edges in a tour T that violate the 2-optimality condition, i.e., they satisfy the inequality $c(a, x) + c(b, y) < c(a, b) + c(x, y)$, then we can replace the edges (a, b) and (x, y) in T by the edges (a, x) and (b, y) and get a strictly shorter tour. We call this operation of replacing the edges (a, b) and (x, y) in T by the edges (a, x) and (b, y) an *improving 2-move*. Thus, the 2-Opt heuristic can be formulated as follows:

2-Opt Heuristic ($V \subseteq \mathbb{R}^2$).

```
start with an arbitrary tour $T$ for $V$
while there exists an improving 2-move in $T$
    perform an improving 2-move
output $T$
```

We call a Euclidean TSP instance $V \subset \mathbb{R}^2$ *degenerate* if there exists a line in $\mathbb{R}^2$ that contains all points of V. Otherwise we call the instance *non-degenerate.*

It is easily seen that in a degenerate Euclidean TSP instance a 2-optimal tour is also an optimal tour:

▶ **Proposition 3.** *In a degenerate Euclidean TSP instance a 2-optimal tour is an optimal tour.*

Proof. Let $V \subseteq \mathbb{R}^2$ be a degenerate Euclidean TSP instance. Let $c : V \times V \to \mathbb{R}$ be the Euclidean distance between two points in V. Then there exist two points $a, b \in V$ such that the straight line segment S from a to b contains all points of V. The length of an optimal TSP tour for V is $2 \cdot c(a, b)$. Assume there exists a 2-optimal TSP tour T that is not optimal. Orient the tour T. Then there must exist a point in $S \setminus V$ that is contained in at least three edges of the tour T and therefore there must exist a point in $S \setminus V$ that is contained in two edges (v, w) and (x, y) of T that are oriented in the same direction. This contradicts the 2-optimality of T as $c(u, x) + c(w, y) < c(v, w) + c(x, y)$. ◀

Because of Proposition 3 we may assume in the following that we have a non-degenerate Euclidean TSP instance.

Let T be a tour in a Euclidean TSP instance. Each edge of T corresponds to a closed line segment in $\mathbb{R}^2$. A tour in a Euclidean TSP instance is called *simple* if no two edges of the tour intersect in a point that lies in the interior of at least one of the two corresponding line segments. For 2-optimal tours in Euclidean TSP instances we have the following simple but very important result.

▶ **Lemma 4** (Flood [6]). *In a non-degenerate Euclidean TSP instance a 2-optimal tour is simple.*

2.1 Crossing-Free Pairs of Tours

Let T be an optimal tour and T' be a 2-optimal tour in a non-degenerate Euclidean TSP instance. By Lemma 4 we know that both tours are simple. In the following we want to justify a much stronger assumption. Two edges $e \in E(T)$ and $f \in E(T')$ *cross* if e and f intersect in exactly one point in $\mathbb{R}^2$ and this point is in the interior of both line segments. We say that two tours T and T' are *crossing-free* if there does not exist a pair of crossing edges. See Figure 2 for an example of an optimal tour and a 2-optimal tour that have three crossing pairs of edges.

To prove Theorem 1 it will be enough to prove it for the special case of crossing-free tours:

▶ **Theorem 5.** *Let $V \subseteq \mathbb{R}^2$ with $|V| = n$ be a non-degenerate Euclidean TSP instance, T an optimal tour for V and S a 2-optimal tour for V. If T and S are crossing-free then the length of S is bounded by $O(\log n/ \log \log n)$ times the length of T.*

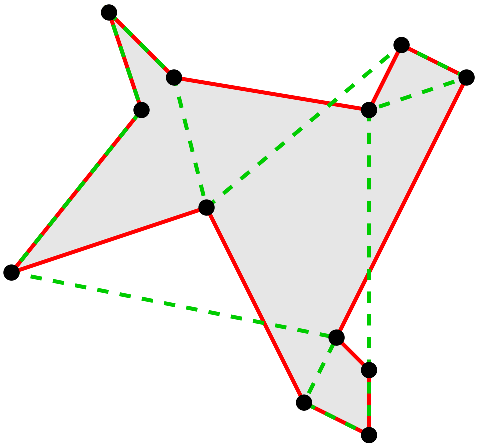


Figure 2 A Euclidean TSP instance with an optimal tour (red edges) and a 2-optimal tour (dashed green edges). Both tours shown in the left picture are simple but there are three pairs of crossing edges. The tours can be made crossing-free by adding three vertices (blue points in the right picture) to the instance.

The proof of Theorem 5 will be presented in Section 5. Here we show how Theorem 5 allows to prove Theorem 1. For this we describe a method to transform a pair of tours into a crossing-free pair of tours.

Let $V \subseteq \mathbb{R}^2$ be a Euclidean TSP instance and T be a tour for V. We say that $V' \subseteq \mathbb{R}^2$ is a *subdivision* for (V, T) if $V \subset V'$ and V' is a subset of the polygon T. The set V' *induces* a new tour T' which results from the tour T by subdividing the edges by points in $V' \setminus V$. Note that T and T' constitute the same polygon. Therefore we have:

▶ **Proposition 6.** *Let $V \subseteq \mathbb{R}^2$ be a Euclidean TSP instance and T be an optimal tour. If V' is a subdivision for (V, T) then the tour T' induced by V' is an optimal tour for V'.*

Subdividing a tour not only preserves the optimality but it also preserves the 2-optimality:

▶ **Lemma 7.** *Let $V \subseteq \mathbb{R}^2$ be a Euclidean TSP instance and T be a 2-optimal tour. If V' is a subdivision for (V, T) then the tour T' induced by V' is a 2-optimal tour for V'.*

Proof. Let us assume that the tour T' is oriented and that (x', y') and (a', b') are two edges of T'. We have to prove that these two edges satisfy the 2-optimality condition (2). As T' is a subdivision of the 2-optimal tour T we know that there exist edges (x, y) and (a, b) in T such that the line segment $a'b'$ is contained in the line segment ab and the line segment $x'y'$ is contained in the line segment xy. The 2-optimality of T implies

$$c(a, b) + c(x, y) \ \leq \ c(a, x) + c(b, y).$$

Using this inequality and the triangle inequality we get:

$$\begin{aligned} c(a', b') + c(x', y') &= c(a, b) - c(a, a') - c(b, b') + c(x, y) - c(x, x') - c(y, y') \\ &\leq c(a, x) - c(a, a') - c(x, x') + c(b, y) - c(b, b') - c(y, y') \\ &\leq c(a', x') + c(b', y'). \end{aligned}$$ ◀

Now we are able to reduce Theorem 1 to Theorem 5:

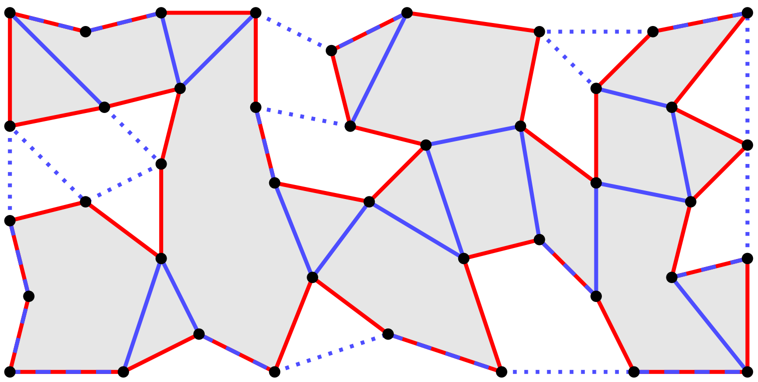


Figure 3 A Euclidean TSP instance with an optimal tour T (red edges) and a 2-optimal tour (blue edges) that are crossing-free. The edges of the 2-optimal tour are partitioned into the edges lying in the interior of T (solid blue lines), the edges that lie in the exterior of T (dotted blue lines), and the edges that are part of T (dashed blue edges).

Proof of Theorem 1. Let $V \subseteq \mathbb{R}^2$ be a Euclidean TSP instance with $|V| = n$, T be an optimal tour for V and S be a 2-optimal tour for V. By Proposition 3 we may assume that V is non-degenerate. Let $V' \subseteq \mathbb{R}^2$ be the set of points obtained by adding to V all crossings between pairs of edges in T and S. Denote the cardinality of V' by n'. Let T' and S' be the tours induced by V' for T and S. Then by Proposition 6 and by Lemma 7 we know that T' has the same length as T and is an optimal tour for V' and S' has the same length as S and is a 2-optimal tour for V'. Now Theorem 5 implies that the length of S is at most $O(\log n' / \log\log n')$ times the length of T. It remains to observe that there can be at most $O(n^2)$ crossings between edges in T and S and therefore $O(\log n' / \log\log n') = O(\log n / \log\log n)$. ◀

2.2 Partitioning the Edge Set of a 2-Optimal Tour

Let $V \subseteq \mathbb{R}^2$ be a non-degenerate Euclidean TSP instance, T be an optimal tour and S be a 2-optimal tour such that S and T are crossing-free. As S and T are simple polygons and S and T are crossing-free we can partition the edge set of S into three sets S_1, S_2, and S_3 such that all edges of S_1 lie in the interior of T, all edges of S_2 lie in the exterior of T and all edges of S_3 are contained in T (see Figure 3). More precisely, an edge $\{a, b\} \in S$ belongs to S_1 resp. S_2 if the corresponding open line segment ab completely lies in the interior resp. exterior of the polygon T. The set S_3 contains all the edges of S that are subsets of the polygon T.

By definition and because of Proposition 3 and Lemma 4 we know that the edges in S_3 are at most as long as the edges in T. To bound the total length of all edges in S_1 in terms of the length of T we proceed as follows: Fix some orientation of the tour S. We may assume that S_1 contains at least two edges as otherwise by the triangle inequality the length of T is an upper bound for the length of the edges in S_1. Choose an edge $e_0 = (x_0, y_0)$ from S_1 such that one of the two x_0-y_0-paths in T does not contain in its interior the endpoints of any other edge in S_1 (see Figure 4).

Let $T_{[x_0,y_0]}$ be the x_0-y_0-path in T that contains the endpoints of all other edges in S_1. The path $T_{[x_0,y_0]}$ is unique if we assume $|S_1| \geq 2$. Then we define the set S_1' to contain all edges from S_1 that are "compatible" with $T_{[x_0,y_0]}$ in the following sense:

$$S_1' := \{(a, b) \in S_1 : \text{ the } x_0\text{-}b\text{-path in } T_{[x_0,y_0]} \text{ contains } a\}.$$

In Figure 4 for the chosen edge $e_0 = (x_0, y_0)$ the edges in S_1' are marked green.

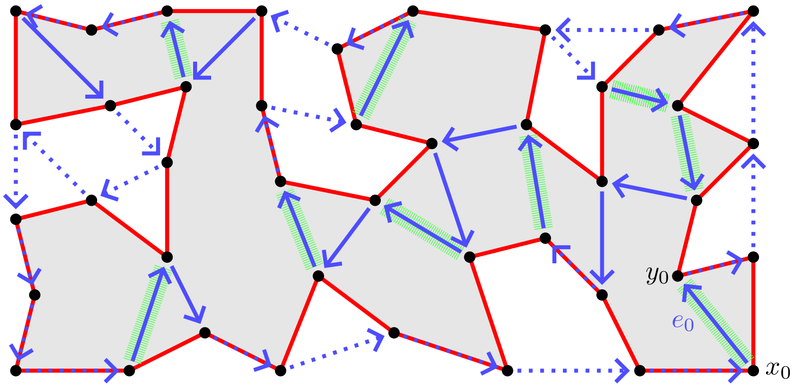


Figure 4 The edge $e_0 = (x_0, y_0)$ of the 2-optimal tour (blue edges) defines the set S'_1 of all green marked edges in the interior of the optimal tour T (red edges).

All edges in S_1 that are oriented in the "wrong" way with respect to $T_{[x_0,y_0]}$ define the set S''_1, i.e., we have $S''_1 := S_1 \setminus S'_1$. Similarly we can define sets S'_2 and S''_2 with respect to some edge $f \in S$ that lies in the exterior of T. We want to prove that for each of the four sets S'_1, S''_1, S'_2, S''_2 we can bound the total length of all edges by $O(\log n / \log\log n)$ times the length of T. To achieve this we will reduce the problem to a problem in weighted arborescences.

3 Arborescences and Pairs of Tours

In this section we will explain why bounding the length of a 2-optimal tour reduces to some problem in weighted arborescences. We start by giving an informal description of the idea.

Let T be an optimal tour for a non-degenerate Euclidean TSP instance $V \subseteq \mathbb{R}^2$ and let S be a 2-optimal tour such that S and T are crossing-free. Then T together with the edge set S'_1 as defined in Section 2.2 is a plane graph. Each region of this plane graph is bounded by edges in $E(T) \cup S'_1$. The boundary of each region is a cycle. Because of the triangle inequality we can bound the length of each edge in a cycle by the sum of the lengths of all other edges in this cycle. This way we get a bound for the length of each edge in S'_1 which we call the *combined triangle inequality* as it arises by applying the triangle inequality to edges from both tours T and S.

From the boundaries of the regions of the plane graph we can derive another type of inequalities. Suppose some boundary B contains at least two edges from S'_1. Then there will be two distinct edges $e, f \in B \cap S'_1$ such that e and f are oriented in opposite direction along the boundary B. (Here we see the reason why we partitioned the edge set S_1 into the two subsets S'_1 and S''_1. In the plane graph arising from T together with the whole edge set S_1 it may happen, that the boundary of a region contains at least two edges from S_1 and these edges are all oriented in the same direction along the boundary.) If we remove e and f from B we get two paths (one of which may be empty) connecting the heads and tails of e and f. Now by applying the triangle inequality to these two paths and using the 2-optimality condition (2) for the edges e and f we get another inequality for the edges in S'_1. We call this inequality the *combined 2-optimality condition* as it arises by applying the 2-optimality condition in combination with the triangle inequality to edges from both tours T and S.

In the following we want to apply the combined triangle inequality and the combined 2-optimality condition to neighboring regions of the plane graph formed by the edge set $T \cup S'_1$. This part of the proof is independent of the embedding induced by the point

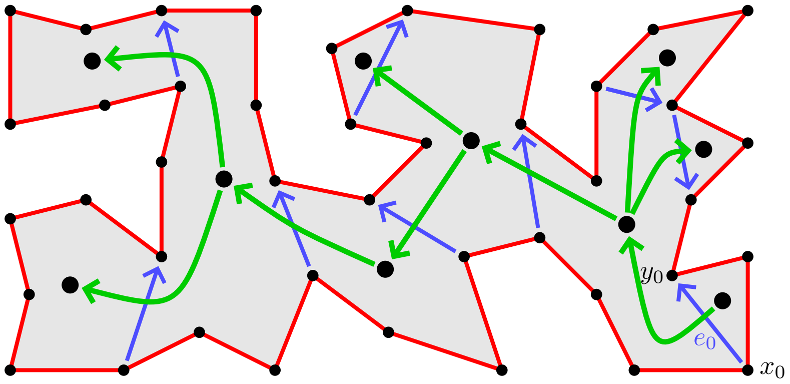


Figure 5 The arborescence (green edges) in the dual of the plane graph formed by the edges of an optimal tour (red edges) and the edges in the set S_1' (blue edges) with respect to the edge $e_0 = (x_0, y_0)$.

coordinates in $\mathbb{R}^2$. In particular this part of the proof can also be applied to point sets in higher dimension by choosing an arbitrary planar embedding of the tour T. We therefore reduce in the following the problem of bounding the total length of all edges in S_1' to a purely combinatorial problem in weighted arborescences.

We now give a formal description of the reduction. Consider the plane graph obtained from T together with the edge set S_1'. Let H be the graph that is obtained from the geometric dual of this plane graph by removing the vertex corresponding to the outer region. Then each edge in H is a dual of an edge in S_1'. As each edge in S_1' is a chord in the polygon T we know that each edge in H is a cut edge and thus H is a tree. See Figure 5 for an example.

We now want to orient the edges of H to get an arborescence. An *arborescence* $A = (V, E)$ is a connected directed acyclic graph that satisfies $|\delta^-(x)| \leq 1$ for all $x \in V(A)$. Each arborescence has exactly one *root* r which is the unique vertex $r \in V(A)$ with $\delta^-(r) = \emptyset$. For an arborescence A with root r we say that A *is rooted at* r.

The set S_1' has been defined with respect to some edge $e_0 = (x_0, y_0)$. The tree H contains a vertex that corresponds to the region of the plane graph that is bounded by the edge e_0 and the edges in $E(T) \setminus T_{[x_0,y_0]}$. By choosing this vertex as the root and orienting all edges in H from the root to the leaves, we get an arborescence A from the tree H (see Figure 5).

We want to define two weight functions on the edge set $E(A)$ of the arborescence A to capture the weights of the edges in T and the edges in S_1'. First we define the function $c : E(A) \to \mathbb{R}_{>0}$ to be the weight of the corresponding dual edge in S_1'. Secondly, we define a weight function $w : E(A) \to \mathbb{R}_{>0}$ as follows. Let $e = (x, y)$ be a directed edge in $E(A)$. Let Y be the region corresponding to the vertex y in the plane graph formed by $E(T) \cup S_1'$. Then we define $w(e)$ to be the weight of all the edges in $E(T)$ that belong to the boundary of Y.

For the arborescence A and the two weight functions c and w we can now state the above mentioned combined triangle inequality and the combined 2-optimality condition as follows:

▶ **Lemma 8.** *Let $V \subseteq \mathbb{R}^2$ be a Euclidean TSP instance with distance function $\bar{c} : V \times V \to \mathbb{R}$ and let T be an optimal tour. Let S be a 2-optimal tour such that S and T are crossing-free. Let S_1' be defined as in Section 2.2 with respect to some edge $e_0 = (x_0, y_0)$. Let A be the arborescence derived from the geometric dual of the plane graph $T \cup S_1'$ with weight functions $w : E(A) \to \mathbb{R}_{>0}$ and $c : E(A) \to \mathbb{R}_{>0}$ as defined above. Then we have*

$$c(e) \;\leq\; w(e) + \sum_{f \in \delta^+(y)} c(f) \quad \textit{for all } e = (x, y) \in E(A) \tag{3}$$

and

$$c(x,y)+c(y,z) \;\leq\; w(x,y)+\sum_{f\in\delta^+(y)\setminus\{(y,z)\}} c(f) \qquad \textit{for all } (x,y),(y,z)\in E(A). \tag{4}$$

Proof. Let $f=(a,b)$ be an edge in S_1' and $f'=(a',b')$ its corresponding dual edge in A. By definition we have $c(f')=\bar{c}(f)$. The vertex b' corresponds to a region R in the plane graph on V with edges $E(T)\cup S_1'$. By the triangle inequality the length $\bar{c}(f)$ of the edge f is bounded by the length of all other edges in the boundary of the region R. Using the definitions of the functions c and w we therefore get:

$$c(f') \;=\; \bar{c}(f) \;\leq\; \sum_{g\in R\cap E(T)} \bar{c}(g)+\sum_{g\in(R\cap S_1')\setminus\{f\}} \bar{c}(g) \;=\; w(f')+\sum_{g\in\delta^+(b')} c(g).$$

This proves condition (3).

We now prove property (4). Let $f\in S_1'$. By definition of the set S_1' we know that S_1' is defined with respect to an edge $e_0=(x_0,y_0)\in S_1$ and the x_0-y_0-path $T_{[x_0,y_0]}$ contains the endpoints of all other edges in S_1. Let $\phi: V(T_{[x_0,y_0]})\to\mathbb{N}$ such that $\phi(z)$ for $z\in V(T_{[x_0,y_0]})$ denotes the distance (in terms of the number of edges) between x_0 and z in $T_{[x_0,y_0]}$. The definition of the set S_1' implies that $\phi(a)<\phi(b)$ for each edge $(a,b)\in S_1'$. Each edge in S_1' can be seen as a shortcut for the path $T_{[x_0,y_0]}$. For an edge $f=(a,b)\in S_1'$ with dual edge $f'=(a',b')\in E(A)$ we denote by $(\delta^+(b'))'$ all edges dual to the edges in $\delta^+(b')$. The edge $f=(a,b)$ and the edges in $(\delta^+(b'))'$ belong to the border of a region of the graph on V with edge set $E(T)\cup S_1'$. Along this border the edge $f=(a,b)$ is directed opposite to all edges in $(\delta^+(b'))'$. Therefore, the triangle inequality together with the 2-optimality condition (2) for the set S_1' imply for each edge $(u,v)\in(\delta^+(b'))'$:

$$\bar{c}(a,b)+\bar{c}(u,v) \;\leq\; w(a',b')+\sum_{g\in(\delta^+(b'))'\setminus\{(u,v)\}} \bar{c}(g).$$

We have $\bar{c}(a,b)=c(a',b')$ and $\bar{c}(u,v)=c(b',x')$ for the vertex $x'\in V(A)$ such that $(u,v)\in S_1$ is the dual edge to $(b',x')\in E(A)$. Therefore we get:

$$c(a',b')+c(b',x') \;\leq\; w(a',b')+\sum_{g\in\delta^+(b')\setminus\{(b',x')\}} c(g). \qquad ◀$$

We call condition (3) the *combined triangle inequality* and condition (4) the *combined 2-optimality condition*. Note that these two conditions can be formulated for any arborescence A with weight functions c and w. In the next section we will show that if these two conditions are satisfied for an arborescence A then we can bound $c(A)/w(A)$ by $O(\log(|E(A)|)/\log\log(|E(A)|))$.

4 The Arborescence Lemmas

Let A be an arborescence with weight functions $w: E(A)\to\mathbb{R}_{>0}$ and $c: E(A)\to\mathbb{R}_{>0}$. We will first show that the combined triangle inequality (3) and the combined 2-optimality condition (4) imply two additional properties. For this we need the following definition. For an arborescence $A=(V,E)$ and an edge $e=(x,y)\in E(A)$ we denote by A_e the sub-arborescence rooted at x that contains the edge e and all descendants of y, see Figure 6 for an example.

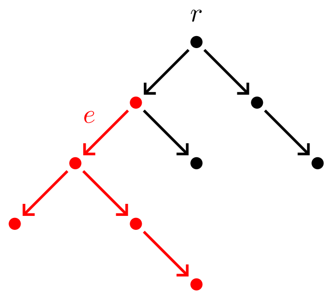


Figure 6 An arborescence A with root r. Shown in red is the sub-arborescence A_e defined by the edge e.

▶ **Lemma 9.** *Let A be an arborescence with weight functions $w : E(A) \to \mathbb{R}_{>0}$ and $c : E(A) \to \mathbb{R}_{>0}$ that satisfies the combined triangle inequality (3). Then we have*

$$c(e) \;\leq\; w(A_e) \quad \textit{for all } e \in E(A). \tag{5}$$

Proof. This follows by induction on the height of the sub-arborescence A_e. If $e = (x, y)$ is an edge in $E(A)$ such that y is a leaf then the combined triangle inequality (3) implies $c(e) \leq w(e) = w(A_e)$. For an arbitrary edge $e = (x, y) \in E(A)$ we get by induction:

$$c(e) \;\leq\; w(e) + \sum_{f \in \delta^+(y)} c(f) \;\leq\; w(e) + \sum_{f \in \delta^+(y)} w(A_f) \;=\; w(A_e).$$ ◀

In the following lemma we have a statement about the maximum weight of a possibly empty edge set. As usual we assume $\max \emptyset = -\infty$.

▶ **Lemma 10.** *Let A be an arborescence with weight functions $w : E(A) \to \mathbb{R}_{>0}$ and $c : E(A) \to \mathbb{R}_{>0}$ that satisfies the combined 2-optimality condition (4). Then we have:*

$$2 \cdot \max_{f \in \delta^+(y)} c(f) \;\leq\; w(x, y) - c(x, y) + \sum_{g \in \delta^+(y)} c(g) \quad \textit{for each edge } (x, y) \in E(A). \tag{6}$$

Proof. From the combined 2-optimality condition (4) we get

$$2 \cdot c(y, z) \;\leq\; w(x, y) - c(x, y) + \sum_{f \in \delta^+(y)} c(f) \qquad \text{for all } (x, y), (y, z) \in E(A).$$

As the right hand side of this inequality is independent of the edge (y, z) we can therefore replace $c(y, z)$ by the term $\max_{f \in \delta^+(y)} c(f)$ on the left hand side. ◀

Our next goal is to bound $c(A)$ in terms of $w(A)$ for arborescences A satisfying (3) and (4). The following two lemmas prove such a statement for certain subsets of $E(A)$.

▶ **Lemma 11.** *Let $A = (V, E)$ be an arborescence with weight functions $w : E(A) \to \mathbb{R}_{>0}$ and $c : E(A) \to \mathbb{R}_{>0}$ that satisfies the combined 2-optimality condition (4). For some fixed number $k \in \mathbb{R}_{>0}$ define $E' := \{(x, y) \in E(A) : \max_{f \in \delta^+(y)} c(f) > \frac{1}{k} \cdot c(x, y)\}$. Then we have:*

$$c(E') \;\leq\; \frac{k}{2} \cdot w(A).$$

Proof. By definition of E' and using Lemma 10 we have for each edge $(x, y) \in E'$:

$$\frac{2}{k} \cdot c(x, y) \;<\; 2 \cdot \max_{f \in \delta^+(y)} c(f) \le w(x, y) - c(x, y) + \sum_{g \in \delta^+(y)} c(g).$$

Adding this inequality for all $(x, y) \in E'$ and using that the left hand side and therefore also the right hand side of inequality (6) is non-negative we get:

$$\begin{aligned}\frac{2}{k} \cdot \sum_{(x,y) \in E'} c(x, y) &< \sum_{(x,y) \in E'} \left(w(x, y) - c(x, y) + \sum_{g \in \delta^+(y)} c(g) \right) \\ &\le \sum_{(x,y) \in E(A)} \left(w(x, y) - c(x, y) + \sum_{g \in \delta^+(y)} c(g) \right) \\ &\le w(A).\end{aligned}$$ ◀

For a fixed number $k \in \mathbb{R}_{>0}$ and a number $r \in \mathbb{R}_{\ge 0}$ we define the edge set $E_r \subseteq E(A)$ as follows:

$$E_r \;:=\; \left\{ e = (x, y) \in E(A) : r < c(e) \le \frac{k}{4} \cdot r \text{ and } c(f) \le \frac{1}{k} \cdot c(e) \;\forall f \in \delta^+(y) \right\}. \tag{7}$$

▶ **Lemma 12.** *Let $A = (V, E)$ be an arborescence with weight functions $w : E(A) \to \mathbb{R}_{>0}$ and $c : E(A) \to \mathbb{R}_{>0}$ that satisfies the combined triangle inequality (3) and the combined 2-optimality condition (4). Let E_r be defined as in (7). Then we have:*

$$c(E_r) \;\le\; 2 \cdot w(A).$$

Proof. Let $e = (x, y) \in E_r$. We first prove by induction on the cardinality of $E(A_e) \cap E_r$:

$$w(A_e) \;\ge\; c(e) + \sum_{f \in (E(A_e) \cap E_r) \setminus \{e\}} \left(c(f) - \frac{r}{4} \right). \tag{8}$$

If $|E(A_e) \cap E_r| = 1$ then $E(A_e) \cap E_r = \{e\}$ and therefore $(E(A_e) \cap E_r) \setminus \{e\} = \emptyset$. Inequality (8) then states $w(A_e) \ge c(e)$ which holds because of Lemma 9.

Now assume that $|E(A_e) \cap E_r| > 1$ and that inequality (8) holds for all edges $f \in E_r$ with $|E(A_f) \cap E_r| < |E(A_e) \cap E_r|$. From the definition of the set E_r we get for each edge $f \in \delta^+(y)$:

$$c(f) \;\le\; \frac{1}{k} \cdot c(e) \;\le\; \frac{1}{k} \cdot \frac{k}{4} \cdot r \;=\; \frac{r}{4}. \tag{9}$$

We define the following two sets of edges:

$$X \;:=\; \{f \in \delta^+(y) : E(A_f) \cap E_r = \emptyset\}$$

and

$$F \;:=\; \{f \in (E_r \cap E(A_e)) \setminus \{e\} : \text{ no edge } h \in E_r \text{ lies on a path from } f \text{ to } e \text{ in } A\}.$$

For each edge $f \in \delta^+(y)$ we either have $E(A_f) \cap E_r = \emptyset$ or $E(A_f) \cap E_r \ne \emptyset$. In the first case the edge f belongs to the set X. In the second case at least one edge from A_f belongs to F. Thus we have

$$|F| + |X| \;\ge\; |\delta^+(y)| \quad \Rightarrow \quad |\delta^+(y) \setminus X| \;\le\; |F|. \tag{10}$$

By the induction hypothesis, inequality (8) holds for each edge $f \in F$ and we can now prove inequality (8) for the edge e:

$$\begin{aligned}
w(A_e) &\geq \sum_{f\in F} w(A_f) + \sum_{f \in X} w(A_f) + w(e) \\
&\overset{(5)}{\geq} \sum_{f\in F} w(A_f) + \sum_{f \in X} c(f) + w(e) \\
&\overset{(3)}{\geq} \sum_{f\in F} w(A_f) + c(e) - \sum_{f \in \delta^+(y)\setminus X} c(f) \\
&\overset{(9)}{\geq} \sum_{f\in F} w(A_f) + c(e) - \sum_{f \in \delta^+(y)\setminus X} \frac{r}{4} \\
&\overset{(10)}{\geq} \sum_{f\in F} \left(w(A_f) - \frac{r}{4}\right) + c(e) \\
&\overset{(8)}{\geq} \sum_{f\in F} \left(c(f) + \sum_{g \in (E(A_f)\cap E_r)\setminus\{f\}} \left(c(g) - \frac{r}{4}\right) - \frac{r}{4}\right) + c(e) \\
&= c(e) + \sum_{f \in (E(A_e)\cap E_r)\setminus\{e\}} \left(c(f) - \frac{r}{4}\right).
\end{aligned}$$

The last equality holds because each edge in $(E(A_e) \cap E_r) \setminus \{e\}$ appears exactly once in the sets $(E(A_f) \cap E_r)$ for $f \in F$.

By definition of the set E_r we have for each edge $f \in E_r$:

$$c(f) \ \geq\ r \quad \Rightarrow \quad \frac{1}{2} \cdot c(f) \ \geq\ \frac{r}{4} \quad \Rightarrow \quad c(f) - \frac{r}{4} \ \geq\ \frac{1}{2} \cdot c(f). \tag{11}$$

Inequality (8) therefore implies

$$\begin{aligned}
w(A_e) &\geq c(e) + \sum_{f \in (E(A_e)\cap E_r)\setminus\{e\}} \left(c(f) - \frac{r}{4}\right) \\
&\geq \sum_{f \in E(A_e)\cap E_r} \left(c(f) - \frac{r}{4}\right) \\
&\overset{(11)}{\geq} \sum_{f \in E(A_e)\cap E_r} \left(\frac{1}{2} \cdot c(f)\right) \\
&= \frac{1}{2} \cdot c(E(A_e) \cap E_r).
\end{aligned}$$

Now choose a minimal set of edges $e_1, e_2, \ldots \in E_r$ such that $E_r \subseteq \bigcup_i E(A_{e_i})$. Then

$$w(A) \ \geq\ w(\bigcup_i E(A_{e_i})) \ =\ \sum_i w(E(A_{e_i})) \ \geq\ \frac{1}{2} \cdot \sum_i c(E(A_{e_i}) \cap E_r) \ =\ \frac{1}{2} \cdot c(E_r). \qquad ◀$$

▶ **Lemma 13.** *Let $A = (V, E)$ be an arborescence with weight functions $w : E(A) \to \mathbb{R}_{>0}$ and $c : E(A) \to \mathbb{R}_{>0}$ that satisfies the combined triangle inequality (3) and the combined 2-optimality condition (4). Moreover we assume that $c(A) \geq 18 \cdot w(A)$. Then we have:*

$$c(A) \ \leq\ 12 \cdot \frac{\log(|E(A)|)}{\log\log(|E(A)|)} \cdot w(A).$$

Proof. We define $k := c(A)/w(A)$. By assumption we have $k \geq 18$. For $i = 1, 2, \ldots, \lfloor k/6 \rfloor$ we define $r_i := \left(\frac{4}{k}\right)^i \cdot w(A)$ and for these numbers we define sets E_{r_i} as in (7). By Lemma 9 we have $c(e) \leq w(A) = \frac{k}{4} \cdot \left(\frac{4}{k}\right)^1 \cdot w(A) = \frac{k}{4} \cdot r_1$ and therefore we have:

$$\bigcup_{i=1}^{\lfloor k/6 \rfloor} E_{r_i} = \left\{ e = (x,y) \in E(A) : \left(\frac{4}{k}\right)^{\lfloor k/6 \rfloor} \cdot w(A) < c(e) \text{ and } c(f) \leq \frac{1}{k} \cdot c(e) \ \forall f \in \delta^+(y) \right\}.$$

Define

$$E' := \{(x,y) \in E(A) : \max_{f \in \delta^+(y)} c(f) > \frac{1}{k} \cdot c(x,y)\}$$

and

$$E^* := \{e \in E(A) : c(e) \leq \left(\frac{4}{k}\right)^{\lfloor k/6 \rfloor} \cdot w(A)\}.$$

Then we have

$$E(A) = E' \cup E^* \cup \bigcup_{i=1}^{\lfloor k/6 \rfloor} E_{r_i}.$$

Using Lemma 11 and Lemma 12 we get:

$$\begin{aligned} k \cdot w(A) \;=\; c(A) &\leq \sum_{i=1}^{\lfloor k/6 \rfloor} c(E_{r_i}) + c(E') + c(E^*) \\ &\leq \lfloor k/6 \rfloor \cdot 2 \cdot w(A) + \frac{k}{2} \cdot w(A) + \left(\frac{4}{k}\right)^{\lfloor k/6 \rfloor} \cdot w(A) \cdot |E^*| \\ &\leq \frac{5}{6} \cdot k \cdot w(A) + \left(\frac{4}{k}\right)^{\lfloor k/6 \rfloor} \cdot w(A) \cdot |E^*|. \end{aligned}$$

This implies

$$|E(A)| \;\geq\; |E^*| \;\geq\; \frac{k}{6} \cdot \left(\frac{k}{4}\right)^{\lfloor k/6 \rfloor} \;\geq\; \left(\frac{k}{6}\right)^{k/6}. \tag{12}$$

The function $\frac{\log x}{\log \log x}$ is monotone increasing for $x > 18$. Therefore we get from inequality (12):

$$\begin{aligned} 2 \cdot \frac{\log(|E(A)|)}{\log \log(|E(A)|)} \cdot w(A) &\geq 2 \cdot \frac{\log\left(\left(\frac{k}{6}\right)^{k/6}\right)}{\log \log\left(\left(\frac{k}{6}\right)^{k/6}\right)} \cdot w(A) \\ &= 2 \cdot \frac{\frac{k}{6} \cdot \log\left(\frac{k}{6}\right)}{\log\left(\frac{k}{6}\right) + \log \log\left(\frac{k}{6}\right)} \cdot w(A) \\ &\geq \frac{k}{6} \cdot w(A) \\ &= \frac{1}{6} \cdot c(A). \end{aligned}$$

◀

5 Proof of Theorem 5

Lemma 8 in combination with Lemma 13 shows that we can bound the length of all edges in S_1' by $O(\log n / \log\log n)$ times the length of an optimal tour T. The statement of Lemma 8 also holds for the set S_1'': We can define an arborescence almost the same way as we did for the set S_1' by taking the dual of the graph on V formed by the edges of T and S_1'' without the vertex for the outer region. The only minor difference is the choice of the root vertex. For S_1' we have chosen as root the vertex that corresponds to the region R bounded by the edge $e_0 = (x_0, y_0)$ and the edges in $E(T) \setminus T_{[x_0,y_0]}$. For the arborescence for S_1'' we choose as a root the region that contains R. The proof of Lemma 8 then without any changes shows that the statement of Lemma 8 also holds for the set S_1''. Similarly, by exchanging the role of the outer and the inner region of T, Lemma 8 also holds for the sets S_2' and S_2''. We are now able to prove our main result:

Proof of Theorem 5. Let $V \subseteq \mathbb{R}^2$ with $|V| = n$ be a non-degenerate Euclidean TSP instance, T an optimal tour for V and S a 2-optimal tour for V such that T and S are crossing-free. We partition the tour S into the five (possibly empty) sets S_1', S_1'', S_2', S_2'', and S_3 as defined in Section 2.2. Then $c(S_3) \leq c(T)$. We claim that $c(S_1') = O(\log n / \log\log n) \cdot c(T)$. If $c(S_1') < 18 \cdot c(T)$ this is certainly the case. Otherwise by Lemma 8 and Lemma 13 we get

$$c(S_1') \;\leq\; 12 \cdot \frac{\log(n)}{\log\log(n)} \cdot c(T)$$

which again proves the claim. As observed above, Lemma 8 also holds for the sets S_1'', S_2', and S_2''. Therefore, we can apply the same argument to the sets S_1'', S_2', and S_2'' and get

$$c(S) \;=\; c(S_1') + c(S_1'') + c(S_2') + c(S_2'') + c(S_3) \;=\; O(\log n / \log\log n) \cdot c(T). \qquad \blacktriangleleft$$

References

1 Sanjeev Arora. Polynomial time approximation schemes for Euclidean Traveling Salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, September 1998.

2 Jon Jouis Bentley. Fast algorithms for geometric Traveling Salesman Problems. *ORSA Journal on Computing*, 4(4):387–411, 1992.

3 Barun Chandra, Howard Karloff, and Craig Tovey. New results on the old k-opt algorithm for the Traveling Salesman Problem. *SIAM J. Comput*, 28(6):1998–2029, 1999.

4 Nicos Christofides. Worst-case analysis of a new heuristic for the Travelling Salesman Problem. Technical Report 388, Carnegie-Mellon University, 1976.

5 Matthias Englert, Heiko Röglin, and Berthold Vöcking. Worst case and probabilistic analysis of the 2-opt algorithm for the TSP. *Algorithmica*, 68:190–264, 2014.

6 Merrill M. Flood. The traveling-salesman problem. *Operations Research*, 4(1):61–75, 1956.

7 Michael R. Garey and David S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

8 Stefan Hougardy, Fabian Zaiser, and Xianghui Zhong. The approximation ratio of the 2-Opt Heuristic for the metric Traveling Salesman Problem. *Operations Research Letters*, 48:401–404, 2020.

9 Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric TSP. arXiv:2007.01409v1 [cs.DS], July 2020. arXiv:2007.01409v1.

10 Sanjeev Khanna, Rajeev Motwani, Madhu Sudan, and Umesh Vazirani. On syntactic versus computational views of approximability. *SIAM J. Comput.*, 28:164–191, 1998.

11 Joseph S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM J. Comput.*, 28(4):1298–1309, 1999.

12 Christos H. Papadimitriou. The Euclidean Traveling Salesman Problem is NP-complete. *Theoretical Computer Science*, 4(3):237–244, June 1977.

13 Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, July 1976.

14 A.I. Serdyukov. O nekotorykh ekstremal'nykh obkhodakh v grafakh. *Upravlyaemye sistemy*, 17:76–79, 1978.

15 Xianghui Zhong. *Approximation Algorithms for the Traveling Salesman Problem.* PhD thesis, Research Institute for Discrete Mathematics, University of Bonn, 2020.

A Framework of Quantum Strong Exponential-Time Hypotheses

Harry Buhrman
QuSoft, CWI, Amsterdam, The Netherlands
University of Amsterdam, The Netherlands

Subhasree Patro
QuSoft, CWI, Amsterdam, The Netherlands
University of Amsterdam, The Netherlands

Florian Speelman
QuSoft, CWI, Amsterdam, The Netherlands
University of Amsterdam, The Netherlands

Abstract

The strong exponential-time hypothesis (SETH) is a commonly used conjecture in the field of complexity theory. It essentially states that determining whether a CNF formula is satisfiable can not be done faster than exhaustive search over all possible assignments. This hypothesis and its variants gave rise to a fruitful field of research, fine-grained complexity, obtaining (mostly tight) lower bounds for many problems in P whose unconditional lower bounds are very likely beyond current techniques. In this work, we introduce an extensive framework of Quantum Strong Exponential-Time Hypotheses, as quantum analogues to what SETH is for classical computation.

Using the QSETH framework, we are able to translate quantum query lower bounds on black-box problems to conditional quantum time lower bounds for many problems in P. As an example, we provide a conditional quantum time lower bound of $\Omega(n^{1.5})$ for the Longest Common Subsequence and Edit Distance problems. We also show that the n^2 SETH-based lower bound for a recent scheme for Proofs of Useful Work carries over to the quantum setting using our framework, maintaining a quadratic gap between verifier and prover.

Lastly, we show that the assumptions in our framework can not be simplified further with relativizing proof techniques, as they are false in relativized worlds.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Quantum complexity theory

Keywords and phrases complexity theory, fine-grained complexity, longest common subsequence, edit distance, quantum query complexity, strong exponential-time hypothesis

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.19

Related Version *Full Version*: `https://arxiv.org/abs/1911.05686`

Funding SP is supported by the Robert Bosch Stiftung. HB, SP, and FS are additionally supported by NWO Gravitation grants NETWORKS and QSC, and EU grant QuantAlgo.

Acknowledgements We would like to thank Andris Ambainis, Gilles Brassard, Frédéric Magniez, Miklos Santha, Mario Szegedy, and Ronald de Wolf for helpful discussions.

1 Introduction

There is a rich diversity of computational problems that are solvable in polynomial time; some that have surprisingly fast algorithms, such as the computation of Fourier transforms or solving linear programs, and some for which the worst-case run time has not improved much for many decades. The problem is that we have no techniques for proving *superlinear* lower bounds. Of the latter category Edit Distance is a good example: this is a problem

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 19; pp. 19:1–19:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

with high practical relevance, and an $O(n^2)$ algorithm using dynamic programming has been known for many decades. Even after considerable effort, no algorithm has been found that can solve this problem essentially faster than n^2. The best known algorithms runs in $O(n^2/\log^2 n)$ time [34], still a nearly quadratic run time.

Traditionally, the field of (structural) complexity theory has studied the time complexity of problems in a relatively coarse manner – the class P, of problems solvable in polynomial time, is one of the central objects of study in complexity theory.

Consider CNF-SAT, the problem of whether a formula, input in conjunctive normal form, has a satisfying assignment. What can complexity theory tell us about how hard it is to solve this problem? For CNF-SAT, the notion of NP-completeness gives a convincing reason why it is hard to find a polynomial-time algorithm for this problem: if such an algorithm is found, all problems in the complexity class NP are also solvable in polynomial time, showing P = NP.

Not only is no polynomial-time algorithm known, but (if the clause-length is arbitrarily large) no significant speed-up over the brute-force method of trying all 2^n assignments is known. Impagliazzo, Paturi, and, Zane [31, 32] studied two ways in which this can be conjectured to be optimal. The first of which is called the *Exponential-Time Hypothesis* (ETH).

▶ **Conjecture 1** (Exponential-Time Hypothesis). *There exists a constant $\alpha > 0$ such that CNF-SAT on n variables and m clauses can not be solved in time $O(m2^{\alpha n})$ by a (classical) Turing machine.*

This conjecture can be directly used to give lower bounds for many natural NP-complete problems, showing that if ETH holds then these problems also require exponential time to solve. The second conjecture, most importantly for the current work, is the *Strong Exponential-Time Hypothesis* (SETH).

▶ **Conjecture 2** (Strong Exponential-Time Hypothesis). *There does not exist $\delta > 0$ such that CNF-SAT on n variables and m clauses can be solved in $O(m2^{n(1-\delta)})$ time by a (classical) Turing machine.*

The strong exponential-time hypothesis also directly implies many interesting exponential lower bounds within NP, giving structure to problems within the complexity class. A wide range of problems (even outside of just NP-complete problems) can be shown to require strong exponential time assuming SETH: for instance, recent work shows that, conditioned on SETH, classical computers require exponential time for *strong simulation* of several models of quantum computation [29, 35].

Surprisingly, SETH is not only a very productive tool for studying the hardness of problems that likely require exponential time, but can also be used to study the difficulty of solving problems within P, forming a foundation for the field of *fine-grained complexity*. The first of such a SETH-based lower bound was given in [40], via a reduction from CNF-SAT to the Orthogonal Vectors problem, showing that a truly subquadratic algorithm that can find a pair of orthogonal vectors among two lists would render SETH false.

The Orthogonal Vectors problem became one of the central starting points for proving SETH-based lower bounds, and conditional lower bounds for problems such as computing the Frechet distance between two curves [20], sequence comparison problems such as the string alignment problem [6] and Dynamic Time Warping [4], can all obtained via a reduction from Orthogonal Vectors. Both the Longest Common Subsequence (LCS) and the Edit Distance problems [11] can also be shown to require quadratic time conditional on SETH, implying that any super-logarithmic improvements over the classic simple dynamic programming algorithm would also imply better algorithms for satisfiability – a barrier which helps explain why it has been hard to find any new algorithms for these problems.

Besides CNF-SAT, the conjectured hardness of other key problems like 3SUM and APSP is also commonly used to prove conditional lower bounds for problems in P. See the recent surveys [38, 39] for an overview of the many time lower bounds that can be obtained when assuming only the hardness of these key problems.

All these results give evidence for the hardness of problems relative to classical computation, but interestingly SETH does not hold relative to *quantum* computation. Using Grover's algorithm [28, 17], quantum computers are able to solve CNF-SAT (and more general circuit satisfiability problems) in time $2^{n/2}$, a quadratic speedup relative to the limit that SETH conjectures for classical computation.

Even though this is in violation of the SETH bound, it is not in contradiction to the concept behind the strong exponential-time hypothesis: the input formula is still being treated as a black box, and the quantum speedup comes "merely" from the general quadratic improvement in unstructured search[1].

It could therefore be natural to formulate the quantum exponential time hypothesis as identical to its classical equivalent, but with an included quadratic speedup, as a "basic QSETH". For some problems, such as Orthogonal Vectors, this conjecture would already give tight results, since these problems are themselves amenable to a speedup using Grover's algorithm. See for instance the Master's thesis [37] for an overview of some of the SETH-based lower bounds that are violated in the quantum setting.

On the other hand, since the conditional lower bound for all problems are a quadratic factor lower than before, such a "basic QSETH" lower bound for LCS or Edit Distance would be merely linear. The best currently-known quantum algorithm that computes edit distance takes quadratic time, so we would lose some of the explanatory usefulness of SETH in this translation to the quantum case.

In this work, we present a way around this limit. Realize that while finding a single marked element is quadratically faster for a quantum algorithm, there is no quantum speedup for many other similar problems. For instance, computing whether the number of marked elements is odd or even can not be done faster when allowing quantum queries to the input, relative to allowing only classical queries [15, 27].

Taking the LCS problem again as an illustrative example, after careful inspection of the reductions from CNF-SAT to LCS [3], we show that the result of such a reduction encodes more than merely the existence of an a satisfying assignment. Instead, the result of these reductions also encodes whether *many* satisfying assignments exist (in a certain pattern), a problem that could be harder for quantum computers than unstructured search. The "basic QSETH" is not able to account for this distinction, and therefore does not directly help with explaining why a linear-time quantum algorithm for LCS has not been found.

We present a framework of conjectures, that together form an analogue of the strong exponential-time hypothesis: QSETH. In this framework, we account for the complexity of computing various properties on the set of satisfying assignments, giving conjectured quantum time lower bounds for variants of the satisfiability problem that range from $2^{n/2}$ up to 2^n.

[1] For unstructured search this bound is tight [16, 19]. Bennett, Bernstein, Brassard, and Vazirani additionally show that with probability 1 relative to a random oracle all of NP cannot be solved by a bounded-error quantum algorithm in time $o(2^{n/2})$.)

Summary of results

- We define the QSETH framework, connecting quantum query complexity to the proving of fine-grained (conditional) lower bounds of quantum algorithms. The framework encompasses both different properties of the set of satisfying assignments, and is also able to handle different input circuit classes – giving a hierarchy of assumptions that encode satisfiability on CNF formulas, general formulas, branching programs, and so on.
  - To be able to handle more-complicated properties of the satisfying assignments, we require such a property to be *compression oblivious* – a notion we define to capture the cases where query complexity is a lower bound for the time complexity, even for inputs that are "compressible" as a truth table of a small formula.[2] We give various results to initiate the study of the set of compression-oblivious languages.
- Some SETH-based $\Omega(T)$ lower bounds carry over to $\Omega(\sqrt{T})$ QSETH lower bounds, from which we immediately gain structural insight to the complexity class BQP.
- We show that, assuming QSETH, the *Proofs of Useful Work* of Ball, Rosen, Sabin and Vasudevan [13] require time $\widetilde{O}(n^2)$ to solve on a quantum computer, matching the classical complexity of these proofs of work.
- We prove that the Longest Common Subsequence (and the Edit Distance) problem requires $\Omega(n^{1.5})$ time to solve on a quantum computer, conditioned on QSETH. We do this by showing that LCS (similarly, edit distance) can be used to compute a harder property of the set of satisfying assignments than merely deciding whether one satisfying assignment exists.

  Following [5], we are able to show this for a version of QSETH where the input formulas are *branching programs* instead, giving a stronger result than assuming the hardness for only CNF inputs.
- As a corollary to the proof of the conditional LCS lower bound, we can show that the query complexity of the restricted Dyck language is linear for any $k = \omega(\log n)$, partially answering an open question posed by Aaronson, Grier, and Schaeffer [2].[3]

Related work

Independently from this work, Aaronson, Chia, Lin, Wang, and Zhang [1] recently also defined a basic quantum version of the strong exponential-time hypothesis, which assumes that a quadratic speed-up over the classical SETH is optimal. They present conditional quantum lower bounds for OV, the closest pair problem, and the bichromatic closest pair problem, by giving fine-grained quantum reductions to CNF-SAT. All such lower bounds have a quadratic gap with the corresponding classical SETH lower bound.

Despite the overlap in topic, these results turn out to be complementary to the current work: In the current work we focus on defining a more extensive framework for QSETH that generalizes in various ways the basic version. Our more general framework can exhibit a quantum-classical gap that is less than quadratic, which allows us to give conditional lower bounds for LCS and edit distance ($\Omega(n^{1.5})$) and useful proofs of work (a quadratic gap between prover and verifier). For our presented applications, the requirements of the fine-grained reductions are lower, e.g., when presenting a lower bound of $n^{1.5}$ for LCS or

[2] This notion is conceptually related to the Black-Box Hypothesis introduced by [14] and studied by [30].

[3] Lower bounds for the restricted Dyck language were recently independently proven by Ambainis, Balodis, Iraids, Khadiev, Klevickis, Prūsis, Shen, Smotrovs and Vihrovs [9].

edit distance it is no problem if the reduction itself takes time $\widetilde{O}(n)$.[4] Conversely, we do not give the reductions that are given by [1]; those results are distinct new consequences of QSETH (both of the QSETH that is presented in that work, and of our more extensive QSETH framework).

Structure of the paper

In Section 2 we motivate and state the QSETH framework. Following that, in Section 3 we present the direct consequences of QSETH, including the maintaining of some current bounds (with a quadratic loss), and the Useful Proof of Work lower bound. In Section 4 we present the conditional lower bounds for LCS and the Edit Distance problem, of which the proofs can be found in the full version of the paper [22]. Additionally, the proof lower bounding the query complexity of the restricted Dyck language can be found in the full version. Finally, we conclude and present several open questions in Section 5.

2 Defining the Quantum Strong Exponential-Time Hypothesis

Almost all known lower bounds for quantum algorithms are defined in terms of *query* complexity, which measures the number of times any quantum algorithm must access the input to solve an instance of a given problem. For example the *polynomial method* [15] and the *adversary method* [8] are two of the main techniques that can be applied in many situations.

Despite the success of quantum query complexity and the fact that we know tight query lower bounds for many problems, the query model does not take into account the computational efforts required after querying the input. In particular, it is not possible to use query complexity to prove any lower bound greater than linear, since any problem is solvable in the query-complexity model after all bits are queried. In general we expect the time needed to solve most problems to be much larger than the number of queries required for the computation, but it still seems rather difficult to formalize methods to provide unconditional quantum time lower bounds for explicit problems. We overcome these difficulties by providing a framework of conjectures that can assist in obtaining *conditional* quantum time lower bounds for many problems in BQP. We refer to this framework as the QSETH framework.

Variants of the classical SETH

The Strong Exponential-Time Hypothesis (SETH) was first studied in [31, 32], who showed that the lack of a $O(2^{n(1-\delta)})$ for a $\delta > 0$ algorithm to solve CNF-SAT is deeply connected to other open problems in complexity theory. Despite it being one the most extensively studied problems in the field of (classical) complexity theory, the best known classical algorithms for solving k-SAT run in $2^{n-n/O(k)}m^{O(1)}$ time [36], while the best algorithm for the more-general CNF-SAT is $2^{n-n/O(\log\Delta)}m^{O(1)}$ [23], where m denotes the number of clauses and $\Delta = m/n$ denotes the clause to variable ratio.

Even though no refutation of SETH has been found yet, it is plausible that the CNF structure of the input formulas does allow for a speed-up. Therefore, if possible, it is preferable to base lower bounds on the hardness of more general kinds of (satisfiability) problems, where

[4] We use $\widetilde{O}$ to denote asymptotic behavior up to polylogarithmic factors.

the input consists of wider classes of circuits. For example, lower bounds based on NC-SETH, satisfiability with NC-circuits as input,[5] have been proven for LCS, Edit Distance and other problems [5], in particular all the problems that fit the framework presented in [21].

Additionally, a different direction in which the exponential-time hypothesis can be weakened, and thereby made more plausible, is requiring the computation of different properties of a formula than whether at least one satisfying assignment exists. For example, hardness of *counting* the number of satisfying assignments is captured by #ETH [26]. Computing existence is equivalent to computing the OR of the set of satisfying assignments, but it could also conceivably be harder to output, e.g., whether the number of satisfying assignments is odd or even, or whether the number of satisfying assignments is larger than some threshold. In the quantum case, generalizing the properties to be computed is not only a way to make the hypothesis more plausible: for many of such tasks it is likely that the quadratic quantum speedup, as given by Grover's algorithm, no longer exist.

2.1 The basic QSETH

To build towards our framework, first consider what would be a natural generalization of the classical SETH.

▶ **Conjecture** (Basic QSETH). *There is no bounded error quantum algorithm that solves CNF-SAT on n variables, m clauses in $O(2^{\frac{n}{2}(1-\delta)}m^{O(1)})$ time, for any $\delta > 0$.*

This conjecture is already a possible useful tool in proving conditional quantum lower bounds, as we present an example of this in Section 3.1.[6]

We first extend this conjecture with the option to consider wider classes of circuits. Let γ denote a class of representations of computational models. Such a representation can for example be polynomial-size CNF formulas, polylog-depth circuits NC, polynomial-size branching programs BP, or the set of all polynomial-size circuits. The complexity of the latter problem is also often studied in the classical case, capturing the hardness of CircuitSAT.

▶ **Conjecture** (Basic γ-QSETH). *A quantum algorithm cannot, given an input C from the set γ, decide in time $O(2^{\frac{n}{2}(1-\delta)})$ whether there exists an input $x \in \{0,1\}^n$ such that $C(x) = 1$ for any $\delta > 0$.*

We also define AC_2^0 to be the set of all depth-2 circuits consisting of unbounded fan-in, consisting only of AND and OR gates. This definition is later convenient when considering wider classes of properties, and it can be easily seen that "basic AC_2^0-QSETH" is precisely the "basic QSETH" as defined above.

Since both these basic QSETH variants already contain a quadratic speedup relative to the classical SETH, conditional quantum lower bounds obtained via these assumptions will usually also be quadratically worse than any corresponding classical lower bounds for the same problems. For some problems, lower bounds obtained using the basic QSETH, or using γ-QSETH for a wider class of computation, will be tight. However, for other problems no quadratic quantum speedup is known.

[5] NC circuits are of polynomial size and polylogarithmic depth consisting of fan-in 2 gates.

[6] Additional examples of implications from such a version of QSETH can be found in the recent independent work of [1].

2.2 Extending QSETH to general properties

We now extend the "basic γ-QSETH" as defined in the previous section, to also include computing different properties of the set of satisfying assignments. By extending QSETH in this way, we can potentially circumvent the quadratic gap between quantum and classical lower bounds for some problems.

Consider a problem in which one is given some circuit representation of a boolean function $f : \{0,1\}^n \to \{0,1\}$ and asked whether a property $\mathrm{P} : \{0,1\}^{2^n} \to \{0,1\}$ on the truth table of this function evaluates to 1, that is, given a circuit C the problem is to decide if $\mathrm{P}(\mathrm{tt}(\mathrm{C})) = 1$, where $\mathrm{tt}(\mathrm{C})$ denotes the truth table of the boolean function computed by the circuit C. If one can only access C as a black box then it is clear that the amount of time taken to compute $\mathrm{P}(\mathrm{tt}(\mathrm{C}))$ is lower bounded by the number of queries made to the string $\mathrm{tt}(\mathrm{C})$. However, if provided with the description of C, which we denote by $\mathrm{desc}(\mathrm{C})$, then one can analyze C to compute $\mathrm{P}(\mathrm{tt}(\mathrm{C}))$ possibly much faster.

For example, take the representation to be polynomial-sized CNF formulas and the property to be OR. Then for polynomial-sized CNF formulas this is precisely the CNF-SAT problem. Conjecturing quantum hardness of this property would make us retrieve the "basic QSETH" of the previous section. Do note that we cannot simply conjecture that any property is hard to compute on CNF formulas: Even though the query complexity of AND on a string of length 2^n is $\Omega(2^n)$ classically and $\Omega(2^{n/2})$ in the quantum case, this property can be easily computed in polynomial time both classically and quantumly when provided with the description of the $n^{O(1)}$ sized CNF formula.

To get around this problem, we can increase the complexity of the input representation: If we consider inputs from AC_2^0, the set of all depth-2 circuits consisting of unbounded fan-in AND and OR gates, we now have a class that is closed under complementation. For this class, it is a reasonable conjecture that both AND, the question whether the input is a tautology and all assignments are satisfying, and OR, the normal SAT problem, are hard to compute.

After this step we can look at further properties than AND and OR. For instance, consider the problem of computing whether there exists an even or an odd number of satisfying assignments. This task is equivalent to computing the PARITY of the truth table of the input formula. How much time do we expect a quantum algorithm to need for such a task?

The quadratic speedup for computing satisfiability, i.e., the OR of the truth table of the input formula, is already captured by the model where the quantum computation only tries possible assignments and then performs Grover's algorithm in a black box manner. If PARITY is also computed in such a way, then we know from query complexity [15] that there is no speedup possible, and the algorithm will have to use $\Omega(2^n)$ steps. Our QSETH framework will be able to consider more-complicated properties, like PARITY.

Finally, observe that such a correspondence, i.e., between the query complexity of a property and the time complexity of computing this property on the set of satisfying assignments, cannot hold for *all* properties, even when we consider more complicated input classes besides CNF formulas. For instance, consider a property which is 0 on exactly the strings that are truth tables of polynomial-sized circuits, and is PARITY of its input on the other strings. Such a property has high quantum query complexity, but is trivial to compute when given a polynomial-sized circuit as input. We introduce the notion of *compression oblivious* below to handle this problem.

White box and black box computation of a property

We formalize the above intuitions in the following way. Let the variable γ denote a class of representation at least as complex as the set AC_2^0, where AC_2^0 denotes the set of poly sized depth-2 circuits consisting of only OR, AND gates of unbounded fan-in and NOT gates. For every n, let $\mathrm{P} : \{0,1\}^{2^n} \to \{0,1\}$ be some function family which defines a property. We define a meta-language L_P such that $L_\mathrm{P} = \{\mathrm{desc(C)} \mid \mathrm{C}$ is an element from the set γ and $\mathrm{P(tt(C))} = 1\}$. We now define the following terms:

▶ **Definition 3** (White-box algorithms). *An algorithm* A *decides the property* P *in* ***white-box*** *if* A *decides the corresponding meta-language* L_P*. That is, given an input string* desc(C)*,* A *accepts if and only if* $\mathrm{P(tt(C))} = 1$*. We use* $\mathrm{qTimeWB}_\epsilon(\mathrm{P})$ *to denote the time taken by a quantum computer to decide the language* L_P *with error probability* ϵ*.*

▶ **Definition 4** (Black-box algorithms). *An algorithm* A *decides the property* P *in* ***black-box*** *if the algorithm* $\mathrm{A}^f(1^n, 1^m)$ *accepts if and only if* $\mathrm{P}(\mathrm{tt}(f)) = 1$*. Here, f is the boolean function computed by the circuit* C *and m is the upper bound on* $|\mathrm{desc(C)}|$ *which is the size of the representation*[7] *that describes f, and* A^f *denotes that the algorithm* A *has oracle access to the boolean function f. We use* $\mathrm{qTimeBB}_\epsilon(\mathrm{P})$ *to denote the time taken by a quantum computer to compute the property* P *in the black-box setting with error probability* ϵ*.*

Compression oblivious properties

We define the set of *compression oblivious* properties corresponding to γ as the set of properties where the time taken to compute this property in the black-box setting is lower bounded by the quantum query complexity of this property on all strings. Formally,

$$\mathcal{CO}(\gamma) = \{\text{properties P such that } \mathrm{qTimeBB}_\epsilon(\mathrm{P}\,|_{\mathrm{S}_\gamma}) \geq \Omega(\mathrm{Q}_\epsilon(\mathrm{P}))\},$$

where $\mathrm{Q}_\epsilon(\mathrm{P})$ denotes the quantum query complexity of the property P in a ϵ-bounded error query model and $\mathrm{S}_\gamma = \{\mathrm{tt(C)} \mid \mathrm{C}$ is an element of the set $\gamma\}$.

Defining QSETH

For each class of representation γ we now define the corresponding γ-QSETH*, which states that computing any compression-oblivious property P in the *white-box* setting is at least as hard as computing P in the *black-box* setting. More formally, for every class of representation γ, such as the class of depth-2 circuits AC_2^0 or poly-sized circuits of a more complex class, we hypothesize the following:

▶ **Conjecture 5** (γ-QSETH*). *For all properties* $\mathrm{P} \in \mathcal{CO}(\gamma)$*, we have* $\mathrm{qTimeWB}_\epsilon(\mathrm{P}\,|_\gamma) \geq \Omega(\mathrm{Q}_\epsilon(\mathrm{P}))$.

2.3 Observations on the set of compression oblivious properties

As the class γ gets more complex, the corresponding γ-QSETH* becomes more credible. The set of compression oblivious properties is an interesting object of study by itself. First consider some representative examples of whether various natural properties are compression oblivious. Note here that the example property that is not compression oblivious has to be

[7] For instance a CNF/DNF formula, an NC circuit, or a general circuit.

carefully constructed for this to be the case – it is natural to conjecture that for most natural properties the knowledge that the input can be written as the truth table of a small circuit does not help in speeding up the computation.[8]

▶ **Example 6.** The properties AND and OR are in $\mathcal{CO}(\mathsf{AC}_2^0)$: The adversarial set that gives the tight query bound for the property AND (OR) are truth tables of functions that can be represented by $n^{O(1)}$ sized DNF (CNF) formulas. Namely, these are given by the formulas that reject (accept) a single possible input, which can be constructed by using n clauses that each contain a single variable or its negation. Because $\mathrm{Q}_\epsilon(\mathrm{AND}|_{\mathsf{S}_{\mathsf{AC}_2^0}}) = \mathrm{Q}_\epsilon(\mathrm{AND})$ and $\mathrm{qTimeBB}_\epsilon(\mathrm{AND}|_{\mathsf{S}_{\mathsf{AC}_2^0}}) \geq \mathrm{Q}_\epsilon(\mathrm{AND}|_{\mathsf{S}_{\mathsf{AC}_2^0}})$, we have $\mathrm{AND} \in \mathcal{CO}(\mathsf{AC}_2^0)$. The same holds for the property OR as well.

▶ **Example 7.** Consider the following property, defined on some string $z \in \{0,1\}^{2^n}$, which we view as the truth table of a formula or circuit:

$$\mathrm{P}_{\text{large-c}}(z) = \mathrm{PARITY}_{2^n}(z) \ \wedge \ [\text{there exists no circuit } C \text{ of size less than } 2^{n/100} \text{ s.t. } z = \mathrm{tt}(C).]$$

Because most strings are not a truth table of a small circuit, the query complexity of this property is close to the query complexity of PARITY, i.e., $\mathrm{Q}_\epsilon(\mathrm{P}_{\text{large-c}}) = \Omega(N)$. Nevertheless, the property is always 0 when restricted to truth tables of small circuits, and therefore trivial to compute. Therefore $\mathrm{P}_{\text{large-c}}$ is not compression oblivious for polynomial-sized circuits (or any smaller class of representations).

▶ **Example 8.** Whether PARITY is compression oblivious is unknown: the quantum query complexity of PARITY is $\Omega(N)$. Restricted to inputs which are truth tables of small formulas/circuits, the query complexity is $O(\sqrt{N})$, this is the maximum query complexity for any property when restricted to truth tables of a small circuit class [10, 33]. Conjecturing that PARITY is compression oblivious is natural, and incomparable to (but not necessarily less likely than) the main QSETH statement.

Given an explicit property P and a class of input representations γ, it would be desirable to unconditionally prove that the property P is *γ-compression oblivious*[9]. This is possible for some simple properties that have query complexity $\Theta(\sqrt{N})$ like OR, corresponding to ordinary satisfiability, and AND. Unfortunately, for more complicated properties, like computing the parity of the number of satisfying assignments, it turns out to be hard to find an unconditional proof that such a property is compression oblivious. The following theorem shows a barrier to finding such an unconditional proof: proving that such a property is compression oblivious implies separating P from PSPACE.

▶ **Theorem 9.** *If there exists a property* P *such that* $\mathrm{Q}_\epsilon(\mathrm{P}) = \widetilde{\omega}(\sqrt{N})$ *and* P *is γ-compression oblivious, and* $\mathrm{P} \in \mathsf{polyL}(N)$*, then* $\mathsf{P} \neq \mathsf{PSPACE}$*. Here* $N = 2^n$ *and* γ *represents the set of poly-sized circuits on* n *input variables.*

Here $\mathsf{polyL}(N)$ is same as $\mathsf{SPACE}(\mathsf{poly}\log N)$, i.e., class of properties computable in $\mathsf{poly}\log N$ amount of space. Note that SETH is already a much stronger assumption than $\mathsf{P} \neq \mathsf{PSPACE}$, therefore this observation leaves open the interesting possibility of proving

[8] In classical complexity theory, a closely related notion is the Black-Box Hypothesis introduced by [14] and studied by [30].

[9] We call a property P a γ-compression oblivious property if $\mathrm{P} \in \mathcal{CO}(\gamma)$.

that properties are compression oblivious assuming that the (Q)SETH holds for simpler properties. (For instance, these simpler properties could include OR and AND, for which it is possible to unconditionally prove that they are compression oblivious.)

Unfortunately, merely making such an assumption alone will likely not be enough to enable an easy proof that simple properties with high query complexity are compression oblivious: We show that there exists an oracle such that, if all computations and input models[10] have access to this oracle, QSETH is true but PARITY (for example) is not compression oblivious. This does give a relativization barrier to this question, showing that a non-relativizing proof will be necessary to prove that properties are compression oblivious.

▶ **Theorem 10.** *There exists an oracle relative to which the basic QSETH holds, but any property $P \in \mathsf{polyL}(N)$ for which $\mathrm{Q}_\epsilon(\mathrm{P}) = \widetilde{\omega}(\sqrt{N})$ is not γ-compression oblivious. Here γ consists of all polynomial-sized circuits (with oracle access).*

See Appendix A for the proofs of Theorems 9 and 10.

3 QSETH lower bounds for Orthogonal Vectors and Proofs of Useful Work

Recall that AC_2^0 denotes the set of polynomial-sized depth-2 circuits consisting of only OR and AND gates of unbounded fan-in. Because of the simple input structure, the AC_2^0-QSETH* conjecture is therefore closest to the classical SETH, and implies the "basic QSETH" as introduced in Section 2.1:

▶ **Corollary 11.** *If AC_2^0-QSETH* is true then there is no bounded error quantum algorithm that solves CNF-SAT on n variables, m clauses in $O(2^{(1-\delta)n/2}m^{O(1)})$ time, for any $\delta > 0$.*

Proof. Consider the property OR: $\{0,1\}^{2^n} \to \{0,1\}$. Using the fact that $\mathrm{OR} \in \mathcal{CO}(\mathsf{AC}_2^0)$, as shown in the previous section, we get $\mathrm{qTimeWB}_\epsilon(\mathrm{OR}|_{\mathsf{AC}_2^0}) \geq \Omega(\mathrm{Q}_\epsilon(\mathrm{OR})) = \Omega(2^{n/2})$. Due to the structure of the DNF formulas one can compute the property OR on DNF formulas on n variables, m clauses in $n^{O(1)}m^{O(1)}$ time. This implies that the hard cases in the set AC_2^0 for the OR property are the CNF formulas. Therefore, $\mathrm{qTimeWB}_\epsilon(\mathrm{OR}|_{\mathrm{CNF}}) \geq \Omega(2^{n/2})$ where the set CNF denotes all the polynomial sized CNF formulas. ◀

In this section we present several immediate consequences of the AC_2^0-QSETH* conjecture:

1. For some problems, classical SETH-based $\Omega(T)$ time lower bounds carry over to the quantum case, with AC_2^0-QSETH*-based $\Omega(\sqrt{T})$ quantum time lower bounds using (almost) the same reduction.
2. The *Proofs of Useful Work* of Ball, Rosen, Sabin and Vasudevan [13] require time $\widetilde{O}(n^2)$ to solve on a quantum computer, equal to their classical complexity, under AC_2^0-QSETH*.

3.1 Quantum time lower bounds based on AC_2^0-QSETH*

The statement of AC_2^0-QSETH* along with Corollary 11 can give quantum time lower bounds for some problems for which we know classical lower bounds under SETH (Conjecture 2).

[10] For example, consider circuit SAT for circuits that have access to an oracle.

▶ **Corollary 12.** *Let* P *be a problem with an* $\Omega(T)$ *time lower bound modulo* SETH*. Then,* P *has an* $\widetilde{\Omega}(\sqrt{T})$ *quantum time lower bound conditioned under* AC_2^0-QSETH* *if there exists a classical reduction from* CNF-SAT *to the problem* P *taking* $O(2^{\frac{n}{2}(1-\alpha)})$ *(for* $\alpha > 0$*) time or if there exists an efficient reduction that can access a single bit of the reduction output.*[11]

In Appendix B we explain how we can preserve the following two classical SETH lower bounds, with a quadratic gap:

▶ **Example 13.** The OV problem is defined as follows. Given two sets U and V of N vectors, each over $\{0,1\}^d$ where $d = \omega(\log N)$, determine whether there exists a $u \in U$ and a $v \in V$ such that $\Sigma_{l\in[d]} u_l v_l = 0$. The reduction of Williams [40], shows a classical lower bound of $\Omega(N^2)$ for this problem, and it can be modified to efficiently return single bits of the reduction. Therefore, assuming AC_2^0-QSETH*, any quantum algorithm requires time $\tilde{\Theta}(N)$ to solve OV for instances of size N.

▶ **Example 14.** The LCS problem is defined as follows. Given two strings a and b over an alphabet set Σ, the LCS(a, b) is the length of the longest subsequence common to both strings a and b. Modifying the reduction of [3], it can be shown that LCS requires time $\widetilde{\Omega}(N)$, assuming AC_2^0-QSETH*. This same bound can also be shown unconditionally, using query complexity and the observation that the majority function can be embedded in LCS.

See the recent results by Aaronson, Chia, Lin, Wang, and Zhang [1] for more examples of reductions from (a variant of) QSETH, that also hold for the basic QSETH of our framework. Additionally, there the authors define the notion of *Quantum Fine-grained Reductions* more generally, and present a study of OV that also includes the case of constant dimension.

We witness that with the AC_2^0-QSETH* conjecture, the SETH-based fine-grained lower bounds at best transfer to a square root lower complexity in the quantum case. This is definitely interesting on its own, but we are aiming for larger quantum lower bounds, in situations where the gap between the classical and quantum complexities is less than quadratic, which is why we focus on our more general framework.

3.2 Quantum Proofs of Useful Work

Other applications of AC_2^0-QSETH* include providing problems for which *Proofs of Useful Work (uPoW)* can be presented in the quantum setting. Ball et al. [13] propose uPoW protocols that are based on delegating the evaluation of low-degree polynomials to the prover. They present a classical uPoW protocol for the Orthogonal Vectors problem (OV) whose security proof is based on the assumption that OV needs $\Omega(n^{2-o(1)})$ classical time in the worst case setting, implying that the evaluation of a polynomial that encodes the instance of OV has average-case hardness. At the end of this protocol, the verifier is able to compute the number of orthogonal vectors in a given instance.

Therefore, the same protocol also works to verify the solutions to $\oplus$OV, where $\oplus$OV denotes the parity version of OV, i.e., given two sets U, V of n vectors from $\{0,1\}^d$ each, output the parity of number of pairs (u, v) such that $u \in U$, $v \in V$ and $\Sigma_{l\in[d]} u_l v_l = 0$,

[11] Note that we use a version of QSETH that relates to CNF-SAT as opposed to bounded clause-size k-SAT problems. One could also define a quantum hardness conjecture for k-CNF or k-DNF, for an arbitrary constant k, in the same way as the original SETH. This variant is required for reductions that use the fact that k is constant, which can occur through usage of the sparsification lemma [31]. For examples where this is necessary within fine-grained complexity, see the *Matching Triangles* problem mentioned in [7] or reductions like in [25].

where d is taken to be $\omega(\log n)$. Assuming AC_2^0-QSETH* and assuming PARITY $\in \mathcal{CO}(\mathsf{AC}_2^0)$ we get that $\oplus$CNF-SAT takes $\Omega(2^n)$ quantum time. Due to the classical reduction[12] given by [40], this protocol then implies a conditional quantum time lower bound of $\Omega(n^2)$ for the $\oplus$OV problem. Therefore, the uPoW protocol by [13] also requires quantum provers to take time $\widetilde{\Omega}(n^2)$.

4 Lower bounds for string problems using NC-QSETH*

In this section we discuss two consequences of the NC-QSETH* conjecture: Quantum time lower bounds for the LCS and Edit Distance problems. For length n input strings, the well-known Wagner–Fischer algorithm (based on dynamic programming) classically computes the edit distance in $O(n^2)$ time. A similar algorithm computes LCS in $O(n^2)$ time. Unfortunately, all the best known classical (and quantum) algorithms to compute these problems are also nearly quadratic. As mentioned above, results by [3, 11] prove that these near-quadratic time bounds might be tight: a sub-quadratic classical algorithm for computing LCS or edit distance would imply that SETH (Conjecture 2) is false.

SETH also implies quadratic lower bounds for many other string comparison problems, like Dynamic Time Warping and Frechet Distance, that also have (close to) optimal algorithms that are based on dynamic programming [21]. Bouroujeni et al. [18] give a sub-quadratic quantum algorithm for approximating edit distance within a constant factor which was followed by a better classical algorithm by Chakraborty et al. [24] However, no quantum improvements over the classical algorithms in the exact case are known to the best of our knowledge. Investigating why this is the case is an interesting open problem: is it possible to prove better (conditional) lower bounds, or can a better algorithm be found? We formulate the following questions for the example of LCS and the Edit Distance problem.

1. Is there a bounded-error quantum algorithm for LCS or Edit Distance that runs in a sub-quadratic amount of time?
2. Is it possible to obtain a superlinear lower bound for LCS or Edit Distance using the "basic QSETH"?
3. Can we use a different reduction to raise the linear lower bound for LCS or Edit Distance that we achieve under "basic-QSETH"?

We don't attempt to find a better algorithm for these string problems in this work, and it remains possible that no sub-quadratic quantum algorithm for these problems exists. Considering the second question: Using the basic QSETH loses a quadratic factor relative to the classical reduction, so it is clear that it will not be possible to directly translate a classical reduction to the quantum setting – since the quadratic classical SETH bound is tight. Therefore, to prove a "basic QSETH" lower bound for a problem where the gap between the best quantum and classical algorithms is less than quadratic, a fundamentally different (inherently quantum) reduction strategy would have to be found.

While the first two questions still remain open, we address the last question in this section. Using (a promise version of) the NC-QSETH* conjecture we prove conditional quantum time lower bounds of $\Omega(n^{1.5})$ for the LCS and Edit Distance problems[13].

[12] Note that here one can use the classical reduction from CNF-SAT to Orthogonal Vectors that runs in time $\widetilde{O}(2^{n/2})$.

[13] Note that, independently from our results, Ambainis et al. [9] recently presented a quantum query lower bound of $\Omega(n^{1.5-o(1)})$ for the Edit Distance problem, for algorithms that use the natural dynamic-

As global strategy, we will analyze earlier reductions [5] from branching program satisfiability to string problems, and show that solving the string problems (such as LCS) on the result of these (slightly modified) reductions can be used to compute a more complicated property of the branching program, which we call $\mathrm{PP}_{\mathrm{lcs}}$. The first step then is to give a reduction from BP-$\mathrm{PP}_{\mathrm{lcs}}$, which can be viewed as showing whether or not $\mathrm{PP}_{\mathrm{lcs}}$ on a branching program is satisfied or not, to LCS. This step is formalized as the following theorem.

▶ **Theorem** (Informal statement of reduction)**.** *There is a reduction from BP-PP_{lcs} on non-deterministic branching programs of size $2^{\mathsf{poly}\log n}$ (length Z, width W) to an instance of the LCS problem on two sequences of length $M = 2^{n/2}(cW)^{O(\log Z)}$ for some constant c, and the reduction runs in $O(M)$ time.*

Our next step is to prove a quantum query complexity lower bound for this property, which, together with the assumption that the property is compression oblivious[14], implies a time lower bound for the LCS problem of $\widetilde{\Omega}(n^{1.5})$. The lower bound strategy for the Edit Distance problem is very similar to that of the LCS problem: the "gadgets" involved have to be constructed in a different way, but these gadgets can then be combined using a very similar method. Therefore, the reduction can be utilized to compute a property of the set of satisfying assignments that is closely related to BP-$\mathrm{PP}_{\mathrm{lcs}}$.

The full proofs of these reductions (together with the definition of $\mathrm{PP}_{\mathrm{lcs}}$) are presented in the full version of the paper [22].

5 Conclusion and Future Directions

We presented a quantum version of the strong exponential-time hypothesis, as QSETH, and demonstrated several consequences from QSETH. These included the transfer of previous Orthogonal-Vector based lower bounds to the quantum case, with a quadratically lower time bound than the equivalent classical lower bounds. We also showed two situations where the new QSETH does not lose this quadratic factor: a lower bound showing that computing edit distance or LCS takes time $n^{1.5}$ for a quantum algorithm, and an n^2 quantum lower bound for Proofs of Useful Work [13], both conditioned on QSETH.

Possible future applications for the QSETH framework are numerous. Most importantly, the QSETH can potentially be a powerful tool to prove conditional lower bounds for additional problems in BQP. The most natural candidates are other string problems, such as Dynamic Time Warping for example, but there are many other problems for which the "basic QSETH" does not immediately give tight bounds.

Additionally, the notion of *compression oblivious* properties are potentially interesting as an independent object of study. We expect most natural properties to be compression oblivious, but leave as an open question what complexity-theoretic assumptions are needed to show that, e.g., the parity function is compression oblivious.

Future directions also include a careful study of quantum time complexity of the other core problems in fine-grained complexity, such as 3SUM and APSP. Just like with satisfiability, the basic versions of these problems are amenable to a Grover-based quadratic speedup. It

programming approach of first reducing Edit Distance to connectivity on a 2D grid. However, that doesn't rule out the possibility of other $\widetilde{O}(n^{1.5-\alpha})$ quantum algorithms for the Edit Distance problem, for $\alpha > 0$.

[14] As discussed in Section 2.3, such an assumption is natural, implicit when considering more-complicated QSETH variants, and hard to prove unconditionally.

is possible that extensions of those key problems can be used to prove stronger conditional lower bounds, in a similar way to the reduction that was used for LCS or Edit Distance in the current work.

References

1 Scott Aaronson, Nai-Hui Chia, Han-Hsuan Lin, Chunhao Wang, and Ruizhe Zhang. On the quantum complexity of closest pair and related problems. *arXiv preprint*, 2019. arXiv:1911.01973.

2 Scott Aaronson, Daniel Grier, and Luke Schaeffer. A Quantum Query Complexity Trichotomy for Regular Languages. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:61, 2019. URL: https://eccc.weizmann.ac.il/report/2019/061.

3 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Quadratic-time hardness of LCS and other sequence similarity measures. *CoRR*, abs/1501.07053, 2015. arXiv:1501.07053.

4 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *Proceedings of the 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, FOCS '15, pages 59–78, Washington, DC, USA, 2015. IEEE Computer Society. doi:10.1109/FOCS.2015.14.

5 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.

6 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *ICALP*, 2014.

7 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 41–50, New York, NY, USA, 2015. ACM. doi:10.1145/2746539.2746594.

8 Andris Ambainis. Quantum lower bounds by quantum arguments. In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 636–643, New York, NY, USA, 2000. ACM. doi:10.1145/335305.335394.

9 Andris Ambainis, Kaspars Balodis, Janis Iraids, Kamil Khadiev, Vladislavs Klevickis, Krisjanis Prusis, Yixin Shen, Juris Smotrovs, and Jevgenijs Vihrovs. Quantum lower and upper bounds for 2d-grid and dyck language. In Javier Esparza and Daniel Král', editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 8:1–8:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.MFCS.2020.8.

10 Andris Ambainis, Kazuo Iwama, Akinori Kawachi, Hiroyuki Masuda, Raymond H. Putra, and Shigeru Yamashita. Quantum identification of boolean oracles. In Volker Diekert and Michel Habib, editors, *STACS 2004*, pages 105–116, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

11 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *STOC*, 2015.

12 Theodore Baker, John Gill, and Robert Solovay. Relativizations of the P=?NP question. *SIAM Journal on computing*, 4(4):431–442, 1975.

13 Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Proofs of useful work. Cryptology ePrint Archive, Report 2017/203, 2017. URL: https://eprint.iacr.org/2017/203.

14 Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im) possibility of obfuscating programs. *Journal of the ACM (JACM)*, 59(2):1–48, 2012.

15 Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *J. ACM*, 48(4):778–797, July 2001. doi:10.1145/502090.502097.

16 Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM J. Comput.*, 26(5):1510–1523, October 1997. doi:10.1137/S0097539796300933.

17 E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997. doi:10.1137/S0097539796300921.

18 Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, Mohammad Taghi Hajiaghayi, and Saeed Seddighin. Approximating edit distance in truly subquadratic time: Quantum and mapreduce. *CoRR*, abs/1804.04178, 2018. arXiv:1804.04178.

19 Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik: Progress of Physics*, 46(4-5):493–505, 1998.

20 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless seth fails. In *Proceedings of the 2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, FOCS '14, pages 661–670, Washington, DC, USA, 2014. IEEE Computer Society. doi:10.1109/FOCS.2014.76.

21 Karl Bringmann and Marvin Kunnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Proceedings of the 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, FOCS '15, pages 79–97, Washington, DC, USA, 2015. IEEE Computer Society. doi:10.1109/FOCS.2015.15.

22 Harry Buhrman, Subhasree Patro, and Florian Speelman. A framework of quantum strong exponential-time hypotheses, 2019. arXiv:1911.05686.

23 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *Proceedings of the 21st Annual IEEE Conference on Computational Complexity*, CCC '06, pages 252–260, Washington, DC, USA, 2006. IEEE Computer Society. doi:10.1109/CCC.2006.6.

24 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. *CoRR*, abs/1810.03664, 2018. arXiv:1810.03664.

25 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, May 2016. doi:10.1145/2925416.

26 Holger Dell, Thore Husfeldt, Dániel Marx, Nina Taslaman, and Martin Wahlén. Exponential time complexity of the permanent and the tutte polynomial. *ACM Transactions on Algorithms (TALG)*, 10(4):21, 2014.

27 Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Limit on the speed of quantum computation in determining parity. *Phys. Rev. Lett.*, 81:5442–5444, December 1998. doi:10.1103/PhysRevLett.81.5442.

28 Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 212–219, New York, NY, USA, 1996. ACM. doi:10.1145/237814.237866.

29 Cupjin Huang, Michael Newman, and Mario Szegedy. Explicit lower bounds on strong quantum simulation. *arXiv preprint*, 2018. arXiv:1804.10368.

30 Russell Impagliazzo, Valentine Kabanets, Antonina Kolokolova, Pierre McKenzie, and Shadab Romani. Does looking inside a circuit help? In *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

31 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

32 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

33 Robin Kothari. An optimal quantum algorithm for the oracle identification problem. In Ernst W. Mayr and Natacha Portier, editors, *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, volume 25 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 482–493, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.STACS.2014.482.

34 William J. Masek and Michael S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980. doi:10.1016/0022-0000(80)90002-1.

35 Tomoyuki Morimae and Suguru Tamaki. Fine-grained quantum computational supremacy. *Quantum Information & Computation*, 19(13&14):1089–1115, 2019. URL: http://www.rintonpress.com/xxqic19/qic-19-1314/1089-1115.pdf.

36 Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for k-SAT. *J. ACM*, 52(3):337–364, May 2005. doi:10.1145/1066100.1066101.

37 Jorg Van Renterghem. The implications of breaking the strong exponential time hypothesis on a quantum computer. Master's thesis, Ghent University, 2019. URL: https://lib.ugent.be/fulltxt/RUG01/002/787/416/RUG01-002787416_2019_0001_AC.pdf.

38 Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis. *IPEC*, 2015.

39 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, 2018. To appear.

40 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2):357–365, December 2005. doi:10.1016/j.tcs.2005.09.023.

A Observations on Compression Oblivious properties

Here we present an extra observation of the set of compression oblivious properties, and missing proofs of the statements in Section 2.3.

First, we show the following fact about how sets of compression-oblivious properties relate, relative to different computational models.

▶ **Fact 15.** *Given two classes of representations ζ and λ, if $\zeta \subseteq \lambda$ then for every property* P, *we have* $\mathrm{P} \in \mathcal{CO}(\lambda)$ *whenever* $\mathrm{P} \in \mathcal{CO}(\zeta)$.

Proof. If $\zeta \subseteq \lambda$ then also for the corresponding sets of truth tables it holds that $\mathrm{S}_\zeta \subseteq \mathrm{S}_\lambda$. If a property $\mathrm{P} \in \mathcal{CO}(\zeta)$, then $\mathrm{qTimeBB}_\epsilon(\mathrm{P}\,|_{S_\zeta}) \geq \Omega(\mathrm{Q}_\epsilon)(\mathrm{P})$ also implies $\mathrm{qTimeBB}_\epsilon(\mathrm{P}\,|_{S_\lambda}) \geq \mathrm{qTimeBB}_\epsilon(\mathrm{P}\,|_{S_\zeta})$ as S_λ is a superset of S_ζ. Therefore, $\mathrm{P} \in \mathcal{CO}(\lambda)$. ◀

▶ **Theorem 9.** *If there exists a property* P *such that* $\mathrm{Q}_\epsilon(\mathrm{P}) = \widetilde{\omega}(\sqrt{N})$ *and* P *is* γ*-compression oblivious, and* $\mathrm{P} \in \mathsf{polyL}(N)$*, then* $\mathsf{P} \neq \mathsf{PSPACE}$*. Here* $N = 2^n$ *and* γ *represents the set of poly-sized circuits on* n *input variables.*

Proof. By way of contradiction, assume $\mathsf{P} = \mathsf{PSPACE}$. We are given a promise that the circuit C to which we have black-box access[15] to is in the set γ, where γ is the set of poly-sized circuits on n input variables. Note that if we would have direct access to the input, instead of black-box access, we can easily solve the problem in polynomial time using the assumption $\mathsf{P} = \mathsf{PSPACE}$.

[15] By black-box access we mean that for any input $x \in \{0,1\}^n$ we can compute $C(x)$.

Using a simplified version of the algorithm for the oracle identification problem [10, 33] we can extract a compressed form of the entire input, effectively going from black-box access back to white-box access, from the set γ using only $\widetilde{O}(\sqrt{N})$ queries. The initial query-efficient algorithm is as follows:

1. Define an $N = 2^n$ bit majority string $m = m_1 m_2 ... m_N$ where $m_i = 1$ if the majority of circuits in γ have 1 in their i^{th} bit of their truth table, else $m_i = 0$.
2. Check whether there exists an index j such that the truth table of circuit C disagrees with m at j. Using Grover's algorithm on the implied string $\mathrm{tt}(C) \oplus m$ this can be achieved using $O(\sqrt{N})$ quantum queries to $\mathrm{tt}(C)$.
   If there is no disagreement, then the string m is the truth table of circuit C and without having to further query C, one can go through all the circuits in γ and compute their respective truth tables to identify C. Using the $\mathsf{P} = \mathsf{PSPACE}$ assumption, this can be done in $\mathsf{poly}(n)$ (classical) time.
3. In the case of a disagreement, remove from γ all the circuits that disagreed with $\mathrm{tt}(C)$ at index j, which, by definition of m, means at least half of the elements from γ are removed.

Repeat these steps until there is no disagreement or until $|\gamma| = 1$. Given that γ initially contained all the poly-sized circuits on n input variables. This whole algorithm requires $O(\sqrt{N} \log |\gamma|) = \widetilde{O}(\sqrt{N})$ quantum queries. Using the $\mathsf{P} = \mathsf{PSPACE}$ assumption, we can implement the same algorithm in $\widetilde{O}(\sqrt{N})$ quantum time as follows.

At any point of the algorithm we have to be able to query the index $i \in [N]$ of $\mathrm{tt}(C)$ and the i^{th} bit of the majority string m at that stage, where the majority string keeps changing every time we update the set γ. Querying any index of $\mathrm{tt}(C)$ is straight forward. On the other hand, the string m is too long to efficiently write down, but will have to be defined implicitly. To enable query access to m, the algorithm will maintain a list of tuples recording previous found positions where the truth table of C differed from the most common values: $\{(i, a_i) \mid i \in [N]$ is the index where there was a disagreement and a_i is the value of the i^{th} bit of $\mathrm{tt}(C)\}$. Now, given such a list, it takes $\mathsf{poly}(n)$ space to compute the current value m_i of the majority string at point i: simply iterate over all elements in the original circuit class up to $\mathsf{poly}(n)$ size, check whether the current circuit D is consistent with the list of previous queries, and then keep tally of $D(i)$. Now we can use the $\mathsf{P} = \mathsf{PSPACE}$ assumption to translate this to a hypothetical algorithm which takes $\mathsf{poly}(n)$ time.

Since $O(\sqrt{N})$ queries suffice to find a single disagreement between $\mathrm{tt}(C)$ and the majority string m at any stage, that means a disagreement (if any) can be found in $\widetilde{O}(\sqrt{N})$ quantum time. Given that there are only $\mathsf{poly}(n)$ such stages, that means we have found the compressed form of circuit C from the set of poly-sized circuits in $\widetilde{O}(\sqrt{N})$ time.

We now have the access to the compressed input of length $n^{O(1)}$. As the property $\mathrm{P} \in \mathsf{polyL}(N)$, we can directly compute P in $O((\log N)^{O(1)}) = O(n^{O(1)})$ amount of space, which again translates to $O((\log N)^{O(1)})$ time under the $\mathsf{P} = \mathsf{PSPACE}$ assumption. Therefore, the total number of (quantum) steps taken is $\widetilde{O}(\sqrt{N}) + O((\log N)^{O(1)})$, which is in contradiction to the assumption that P is γ-compression oblivious. ◀

▶ **Theorem 10.** *There exists an oracle relative to which the basic QSETH holds, but any property $P \in \mathsf{polyL}(N)$ for which $\mathrm{Q}_\epsilon(\mathrm{P}) = \widetilde{\omega}(\sqrt{N})$ is not γ-compression oblivious. Here γ consists of all polynomial-sized circuits (with oracle access).*

Proof. We construct the oracle in two steps. We first start with the Quantified Boolean Formula (QBF) problem as oracle, call this oracle A. Since QBF is complete for PSPACE, and since a call to A can itself be simulated in polynomial space, note that $\mathsf{P}^A = \mathsf{BQP}^A = \mathsf{PSPACE}^A$.

Recall the classic oracle from Baker, Gill, and Solovay [12], relative to which $\mathsf{P} \neq \mathsf{NP}$. This construction occasionally hides a single string of a certain length in the oracle, for a very sparse set of lengths, and shows that it is hard for a Turing machine to find the string in time less than 2^n.

This same construction also works or quantum computation: We will construct the oracle B in steps. Take the i-th oracle quantum Turing machine, with access to oracle A, and consider that it makes at most $o(2^{n_i/2})$ queries when given input 1^{n_i}, where $n_i = 2^{n_{i-1}}$. We aim to construct B such that the language

$$L_B = \{1^n \mid \text{The oracle } B \text{ contains a string of length } n\}$$

can not be decided by such a machine. Via lower bounds for unstructured search [16, 19, 8, 15], there has to exist a single-string setting of the oracle at B that makes the i-th machine fail. I.e., either B has a single string of length n_i, or the oracle is empty at n_i. Via the query lower bound of unstructured search, this language requires $2^{n/2}$ quantum time.

The final oracle C is just the direct sum of the oracle A and B:

$$C = \{(i, x) \mid (i = 0 \wedge x \in A) \vee (i = 1 \wedge x \in B)\}\,.$$

Relative to C, both SETH, as in Conjecture 2, and the basic QSETH are true (where we consider a relativized "basic QSETH" that takes as input circuits which can make oracle queries to C). In particular, satisfiability of the circuit which queries its input to C and outputs the result takes time $2^{n/2}$ to compute for a quantum Turing machine which has oracle access to C (since any hypothetical machine which solves this language faster, would be able to decide the hard language L_B).

Now consider the hardness of computing some property P of a string, for which we only get black box access to this string, and such that it's known that the string is a truth table of a polynomial-sized circuit which has access to oracle C. A quantum computer can first search the part of C that corresponds with B for the hidden string, using Grover's algorithm for unstructured search, taking time $2^{n/2}$. Now, after finding the hidden string, part B of the oracle is no longer relevant since any call to it can be efficiently simulated by a short computation, and therefore the oracle is effectively only a QBF oracle, meaning that after finding the string we effectively have $\mathsf{P} = \mathsf{PSPACE}$ relative to the oracle. The quantum algorithm can next use the A part, using the construction in Theorem 9, to compute the property P in total time $O^*(2^{n/2}) = \widetilde{O}(\sqrt{N})$. Since we assumed that P has query complexity at least $\widetilde{\omega}(\sqrt{N})$, it follows that P is not compression oblivious relative to the oracle. ◀

B Example lower bounds following from the basic QSETH assumption

As examples we will considered the Orthogonal Vectors (OV) and the LCS problem. The OV problem is defined as follows. Given two sets U and V of N vectors, each over $\{0,1\}^d$ where $d = \omega(\log N)$, determine whether there exists a $u \in U$ and a $v \in V$ such that $\Sigma_{l\in[d]} u_l v_l = 0$. In [40], Williams showed that SETH implies the non-existence of a sub-quadratic classical algorithm for the OV problem. In the quantum case the best-known query lower bound is $\Omega(n^{2/3})$, which can be achieved by reducing the 2-to-1 Collision problem to the Orthogonal Vectors problem; however, the known quantum time upper bound is $\widetilde{O}(n)$ [37]. First note that we cannot use Williams' classical reduction directly, since a hypothetical quantum algorithm for OV expects quantum access to the input, and writing down the entire reduction already takes time $2^{n/2}$. Instead, observe that the reduction produces a separate vector for each partial assignment: let $t(n)$ be the time needed to compute a single element of the output of the reduction, then $t(n) = \mathsf{poly}(n)$, which is logarithmic in the size of the total reduction. Let $N = O^*(2^{n/2})$ be the size of the output of the reduction of [40], for some CNF formula with n variables. Any quantum algorithm that solves OV in time N^α, can solve $\mathsf{CNF\text{-}SAT}$ in time $t(n)O^*(2^{\alpha n/2}) = O^*(2^{\alpha n/2})$.[16] Assuming AC_2^0-QSETH*, this implies that a quantum algorithm requires time $\widetilde{\Theta}(N)$ to solve OV for instances of size N.

The next example we consider is the LCS problem. The LCS problem is defined as follows. Given two strings a and b over an alphabet set Σ, the $\mathrm{LCS}(a, b)$ is the length of the longest subsequence common to both strings a and b. A reduction by [3] shows that if LCS of two strings of length $O(n)$ can be computed in time $O(n^{2-\delta})$ for some constant $\delta > 0$, then satisfiability on CNF formulas with n variables and m clauses can be computed in $O(m^{O(1)} \cdot 2^{(1-\frac{\delta}{2})n})$ which would imply that SETH (Conjecture 2) is false. Just like in the Orthogonal Vectors case, we observe that the classical reduction from $\mathsf{CNF\text{-}SAT}$ to LCS is local, in the sense that accessing a single bit of the exponentially-long reduction output can be done in polynomial time: Every segment of the strings that are an output of the reduction, depend only on a single partial satisfying assignment, out of the $2^{n/2}$ possible partial assignments.

This observation directly lets us use the reduction of [3] to give a quantum time lower bound of $\widetilde{\Omega}(N)$ for the LCS problem, where N here is the length of the inputs to LCS, conditioned on AC_2^0-QSETH*. However, an unconditional quantum query lower bound of $\Omega(N)$ can also be easily achieved by embedding of a problem with high query complexity, such as the majority problem, in an LCS instance.

[16] We use O^* to denote asymptotic complexity ignoring polynomial factors.

The Complexity of the Distributed Constraint Satisfaction Problem

Silvia Butti
Department of Information and Communication Technologies, Universitat Pompeu Fabra, Barcelona, Spain

Victor Dalmau
Department of Information and Communication Technologies, Universitat Pompeu Fabra, Barcelona, Spain

Abstract

We study the complexity of the Distributed Constraint Satisfaction Problem (DCSP) on a synchronous, anonymous network from a theoretical standpoint. In this setting, variables and constraints are controlled by agents which communicate with each other by sending messages through fixed communication channels. Our results endorse the well-known fact from classical CSPs that the complexity of fixed-template computational problems depends on the template's invariance under certain operations. Specifically, we show that DCSP(Γ) is polynomial-time tractable if and only if Γ is invariant under symmetric polymorphisms of all arities. Otherwise, there are no algorithms that solve DCSP(Γ) in finite time. We also show that the same condition holds for the search variant of DCSP.

Collaterally, our results unveil a feature of the processes' neighbourhood in a distributed network, its iterated degree, which plays a major role in the analysis. We explore this notion establishing a tight connection with the basic linear programming relaxation of a CSP.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases Constraint Satisfaction Problems, Distributed Algorithms, Polymorphisms

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.20

Related Version *Full Version*: `https://arxiv.org/abs/2007.13594`

Funding *Silvia Butti*: The project that gave rise to these results received the support of a fellowship from "la Caixa" Foundation (ID 100010434). The fellowship code is LCF/BQ/DI18/11660056. This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 713673.
Victor Dalmau: Victor Dalmau was supported by MICCIN grants TIN2016-76573-C2-1P and PID2019-109137GB-C22.

Acknowledgements We would like to thank Gergely Neu for useful discussions on the weighted majority algorithm.

1 Introduction

The Constraint Satisfaction Problem (CSP) consists of a collection of variables and a collection of constraints where each constraint specifies the valid combinations of values that can be taken simultaneously by the variables in its scope. The goal is to decide if there exists an assignment of the elements of a domain to the variables which satisfies all constraints. The CSP is a very rich mathematical framework that is widely used both as a fruitful paradigm for theoretical research, and as a powerful tool for applications in AI, such as scheduling and planning [21, 17].

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 20; pp. 20:1–20:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

While, in its full generality, the finite-domain CSP is known to be NP-complete, applying specific restrictions on the instances can yield tractable subclasses of the problem. One of the most studied approaches consists in requiring that, in each constraint, the set of allowed combinations for its values be drawn from a prescribed set Γ, usually called the constraint language or the template. Thanks to the proof of the CSP dichotomy conjecture obtained separately in [9] and [28], which culminated a decades-long research program, it is possible to determine the complexity (P or NP-complete) of each family of CSPs, CSP(Γ), which is obtained by fixing Γ. This proof confirmed that the complexity of the constraint satisfaction problem is deeply tied to certain algebraic properties of the constraint language. Specifically, it depends on whether or not the constraint language is invariant under certain operations known as its polymorphisms. The polymorphisms of a constraint language enforce a symmetry on the space of solutions of a CSP instance that can possibly be exploited by an algorithm. This connection with algebra is also present in our work.

We study the computational complexity of the distributed counterpart of CSP, which is known as DCSP. This was introduced by Yokoo et al. [25] as a formal framework for the study of cooperative distributed problem solving. In particular, we consider a deterministic, synchronous, anonymous network of agents controlling variables and constraints, and we study the complexity of message passing algorithms on this network. A number of practical applications can be encoded in the DCSP model, for instance resource allocation tasks in wireless networks, routing, networking, and mobile technologies (see for instance [10, 6]).

We notice that this framework is general enough to encompass some simple Graph Neural Network architectures (see for example [20, 14]). In particular, when training a GNN to classify graphs, it is customary that the GNN network ignores the node label when updating its feature vector. This is, in fact, essential as otherwise there would be no way to apply the network trained on a given graph to another one. However, whereas in all variants of GNNs the computation is limited to a reduced number of operations over feature vectors, in the DCSP model the computation at each node is governed by an arbitrary algorithm. GNNs have a wide range of applications including molecule classification or image classification (see [5] for example). Recently, GNNs have been deployed to solve CSPs [22].

While there are a variety of well-performing distributed algorithms for constraint satisfaction and optimisation (see for instance [27, 19, 11]), the theoretical aspects of distributed complexity are to date not well understood. In this paper we initiate the study of the complexity of DCSP parametrized by the constraint language, obtaining a complete characterization of its tractable classes. More specifically, building on the connection between the CSP and algebra, we show that for any finite constraint language Γ, the decision problem for DCSP(Γ) is tractable whenever Γ is invariant under symmetric polymorphisms of all arities, where an operation is symmetric if its result does not depend on the order of its arguments. Otherwise, there are no message passing algorithms that solve DCSP(Γ). Collaterally, we show that the same holds for the search problem for DCSP.

Our work begins with the identification of a feature of the nodes in a distributed network, its iterated degree, which plays a major role in how messages are transmitted in the network. The iterated degree is an extension of the similar concept introduced in the study of the isomorphism problem which turns out to have a variety of alternative characterizations in terms of fractional isomorphisms, the Weisfeiler-Leman test, and definability with counting logics (see [14]). It turns out that, due to the network anonymity, in every distributed algorithm all equivalent agents (with respect to iterated degree) must necessarily behave identically at each round. A similar phenomenon has been observed independently in the context of GNNs in [20, 23] leading to further study in [3].

We use this fact to show that, under the absence of symmetric polymorphisms of any arity in Γ, it is always possible to construct two instances of DCSP(Γ), one satisfiable and the other unsatisfiable, that cannot be distinguished by any message passing algorithm in an anonymous network.

On the other hand, invariance under symmetric polymorphisms is connected with the basic linear programming relaxation of a CSP instance. More precisely, if Γ has symmetric polymorphisms of all arities then one can decide the satisfiability of every instance of CSP(Γ) by checking whether its basic linear programming relaxation is feasible (see for instance [4]). Whereas it is not clear how to directly use this fact to obtain a distributed algorithm for DCSP(Γ), it can be applied to establish a structure theorem that unveils a simple yet surprising structure in the solution space of every satisfiable instance in DCSP(Γ): it must contain a solution that assigns the same value to all variables that have the same iterated degree. The proof of the structure theorem uses the weighted majority algorithm, a weight update method that is widely used in optimisation and machine learning applications (see [2]). The structure theorem is key in the proof of the positive results as it allows to run an adapted variant of the *jpq*-consistency algorithm [16] that overcomes the absence of unique identifiers for the variables, by using instead their iterated degree.

This paper is organised as follows. In Section 2 we introduce some definitions and technical concepts about the DCSP model. In Section 3 we present the basic LP relaxation for CSPs and we show its connection to the symmetry on the solution space, culminating in the statement of the structure theorem. Section 4 is dedicated to the proof of the dichotomy theorem for the complexity of DCSP, with the hardness results in Section 4.1, the details of the distributed algorithm for tractable languages in Section 4.2, and its extension to the search problem in Section 4.3. Finally, in the Conclusion we discuss some directions into which our work could be extended.

2 Preliminaries

Constraint Satisfaction Problems

An instance I of the finite-domain *Constraint Satisfaction Problem* (CSP) is a triple (X, D, C) where X is a set of variables, D is a finite set called the domain, and C is a set of constraints where a constraint $c \in C$ is a pair $(\mathbf{s}, R)$ where $R \subseteq D^k$ for k a positive integer, R is a relation over D of arity k, and $\mathbf{s}$ is a tuple of k variables, known as the *scope* of c. We use $\mathit{arity}(\cdot)$ to denote the arity of a relation, tuple, or constraint and we write $x \in c$ for any variable x in the scope of c. An *assignment* $\nu : X \to D$ is said to be *satisfying* if for all constraints $c = (\mathbf{s}, R) \in C$ we have $\nu(\mathbf{s}) \in R$, where ν is applied to $\mathbf{s}$ coordinate-wise. Usually we denote the number of variables by n and the number of constraints by m.

Let Γ be a set of relations over some finite domain D, and let CSP(Γ) denote the set of CSP instances with all constraint relations lying in Γ. In this context, Γ is known as the *constraint language*. Throughout this paper, we will assume that Γ is always finite. Then, the *decision problem* for CSP(Γ) is the problem of deciding whether a satisfying assignment exists for an instance $I \in$ CSP(Γ). The *search problem* for CSP(Γ) is the problem of deciding whether a satisfying assignment exists and, if it does, to find one such assignment.

The Distributed Model

We consider the DCSP model of [25] with some small modifications. The basic idea is to assign the task of solving a constraint satisfaction problem to a multi-agent system. In the original model, which assumes that all constraints are binary [26, 27], the assumption is that each variable is controlled by an agent, and two agents can communicate with one another if and only if they share a constraint. Here we deviate slightly from the original model to allow for non-binary constraints and we assume that both variables and constraints are controlled by distributed agents in the network. An instance of the *Distributed Constraint Satisfaction Problem* (DCSP) is a tuple (A, X, D, C, α), where X, D, and C are as in the classical CSP, A is a finite set of agents, and $\alpha : X \cup C \to A$ is a surjective function which assigns the control of each variable $x \in X$ and each constraint $c \in C$ to an agent $\alpha(x)$, $\alpha(c)$ respectively. For the purpose of this paper, we assume that there are exactly $n + m$ agents, and therefore each agent controls exactly one variable or one constraint. This can be done without loss of generality since any agent controlling multiple nodes can simulate multiple agents, each controlling a node. Under this assumption, there is a one-to-one correspondence between instances of CSP and DCSP, and thus we shall switch freely between them.

Distributed Networks and Message Passing

We now present some fundamental concepts relating to the message-passing paradigm for distributed networks. For a general introduction to distributed algorithms, we refer the reader to [12]. A *distributed system* consists of a finite set of nodes or processes, which are connected through communication channels to form a network. Any process in the network can perform events of three kinds: *send*, *receive* and *internal*. Send and receive events are self-explanatory, as they denote the sending or receiving of a message over a communication channel. Any kind of local computation performed at the process level, as well as state changes and decisions, are classified as internal events.

We assume a fully *synchronous* communication model, meaning that the send event at a process a and the corresponding receive event at a process a' can be considered *de facto* as a unique event, with no time delay. As a whole, a synchronous system proceeds in rounds, where at each round a process can perform some internal computation and then send messages to and receive messages from its neighbours. A round needs to terminate at every process before the next round begins. Note that while for simplicity we assume a synchronous network, all our algorithms can be adapted to asynchronous systems by applying a simple synchronizer. Nonetheless, we point out that our negative results rely on the network operating in synchronous rounds.

We make the fundamental assumption that the network is *anonymous*, meaning that variables, constraints and agents do not have IDs. For practical purposes, we still refer to variables and constraints with names (such as x_i, c_i), however these cannot be communicated through the channels. The assumption of anonymity can have various practical justifications: the processes may actually lack the hardware to have an ID, or they may be unable to reveal their ID due to security or privacy concerns. For instance, the basic architecture of GNNs is anonymous. This is a very desirable property as it allows to deploy GNNs in different networks than those in which they were trained.

We assume that all the processes run locally the same deterministic algorithm, therefore IDs cannot be created and deadlocks cannot be broken by for instance flipping a random coin. Hence, the lack of IDs makes the processes essentially indistinguishable from one another - except, as we will see later, for the structure of their neighbourhood in the network.

Leader election is a procedure by which the processes in a network select a single process to be the leader in a distributed way. If a leader is elected, then she can assign unique identifiers to every process. Moreover, all the information about the instance can be gathered to the leader, who can then solve the CSP locally. It is a well-known result that there does not exist a terminating deterministic algorithm for electing a leader in an anonymous ring [1].

Therefore, the assumptions of anonymity and determinism ensure that the DCSP model is intrinsically different from the (centralised) CSP framework, and open up the way for establishing novel, non-trivial complexity results. We remark that while considerable effort has been put into characterizing under what conditions an anonymous network is able to elect a leader [7, 24] or compute relations [8], our work focuses on characterizing the complexity of the DCSP as parametrised by the constraint language. Therefore, all of our algorithms work regardless of the topology of the network, and hence regardless of whether or not a leader can be elected.

The encoding of a DCSP instance into the message passing framework is straightforward. The processes correspond to the agents of the network, and there is a labelled communication channel between a variable agent $\alpha(x)$ and a constraint agent $\alpha(c)$ if and only if $x \in c$. More formally, the *Factor Graph* [11] G_I of an instance $I = (X, D, C)$ of CSP is the undirected bipartite graph with vertex set $X \cup C$ and edge set $\{\{x, c\} \mid x \in c\}$. Each edge in G_I that is incident to a variable x and a constraint c where $c = (\mathbf{s}, R)$ has a label $\ell_{x,c} = (S, R)$ for $S = \{i \mid \mathbf{s}[i] = x\}$, where for a tuple $\mathbf{t}$, $\mathbf{t}[i]$ denotes the i^{th} entry of $\mathbf{t}$.[1] Then, the message passing network corresponds to the factor graph where every node (variable or constraint) is replaced by their associated agent and every edge by a communication channel of the same label. Note that between any two nodes there is at most one channel. If privacy is a concern, we point out that labeling channels does not reveal any more information about the processes than what is strictly necessary for the problem instance to be well defined. Unless explicitly stated we only consider instances whose factor graph consists of a unique connected component. It is easy to prove (see the full version for details) that in the case that all relations are binary, the original model where only variables are controlled by agents is equivalent to our model.

At the start of an algorithm, a process only has access to very limited information. All processes know the total number n of variables in the CSP instance, the total number m of constraints, the labels of the communication channels that they are incident to in the network, and naturally whether they are controlling a variable or a constraint. During a run of the algorithm a process can acquire further knowledge from the messages that it receives from its neighbours. We assume that at any time each process is in one of a set of states, a subset of which are terminating states. When it enters a terminating state, a process performs no more send or internal events, and all receive events are disregarded. The local algorithm is then a deterministic function which determines the process' next state, and the messages it will send to its neighbours. The output of such function only depends on the process' current knowledge, on its state, and on the global time. We allow processes to send different messages through different channels. However, since processes can only distinguish the channels based on their labels, identical messages must be sent through channels with identical labels. Note that the power of the model would not decrease if only one message was allowed to be passed through all the channels, since a process can simulate sending a separate message through each channel by tagging each message with the label of the desired channel and concatenating them in a unique string. This, however, comes at the cost of increased message size. Moreover, if a process needs to broadcast multiple messages, these can be concatenated into one. We say that an algorithm terminates when all processes are in a terminating state.

[1] For mathematical clarity, we label edges with the relation itself. However, in algorithmic applications, every relation can be substituted with a corresponding symbol.

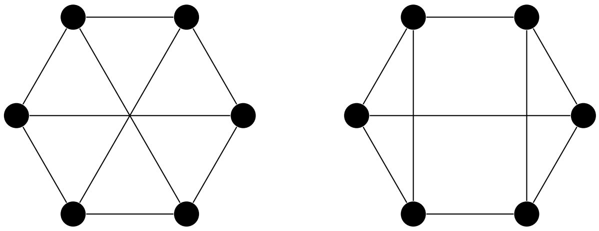


Figure 1 Both graphs depicted above are 3-regular and hence they have the same iterated degree sequence. However, they are clearly not isomorphic, since the left graph is bipartite while the right one is not.

We say that a distributed algorithm solves an instance I of DCSP if the algorithm terminates and the terminating state of every process correctly states that I is satisfiable if it is, and that it is not satisfiable otherwise. Moreover, we consider the search version of DCSP, denoted DCSP-Search. In the search version, if the input instance I is satisfiable, the terminating state of every variable process $\alpha(x)$ must additionally specify a value $\nu(x) \in D$ such that $\nu : X \to D$ is a satisfying assignment. For every constraint language Γ, we denote by DCSP(Γ) and DCSP-Search(Γ) the restrictions of DCSP and DCSP-Search, respectively, to instances containing only constraint relations from Γ.

In terms of algorithmic complexity, there are a number of measures that can be of interest. Time complexity, which is our primary concern, corresponds to the total amount of time required for the algorithm to terminate, including the time needed for internal events. This is closely related to the number of rounds of the algorithm, which is another measure that we are concerned with. Message complexity and bit complexity measure the total number of messages and bits exchanged respectively. These can be bounded easily from the maximum size of a message.

Iterated Degree and Degree Sequence

We present a number of concepts from graph theory that carry over to CSPs. Their adaptation to DCSPs is straightforward in all cases. In an undirected graph G, the degree of a vertex v is the number of edges incident at v. The zeroth iterated degree of v is equal to its degree. For $k \geq 1$, the k^{th} iterated degree of v is the multiset of $(k-1)^{th}$ degrees of v's neighbours in G. The k^{th} iterated degree sequence of a graph is the multiset of k^{th} iterated degrees of its vertices.

▶ **Example 2.1.** In the context of graph theory the *colour refinement algorithm*, which calculates the iterated degree sequence of a graph, is often used as a simple heuristic for the graph isomorphism problem. If two graphs are isomorphic then they must have the same iterated degree sequence, but the opposite is not true (see for example Figure 1). ⌟

We extend the notion of iterated degree to CSPs as follows. Consider the labelled factor graph G_I of an instance I described in the previous paragraph. In what follows it will be convenient to allow instances I with a disconnected factor graph G_I. Let v be a node of G_I and denote its neighbourhood in the factor graph by $N(v)$. The (zeroth) *degree*, denoted $\delta_0(v)$, of a node in the factor graph is simply a symbol that distinguishes variables from constraints: we set $\delta_0(x) =$ "●" for all $x \in X$ and $\delta_0(c) =$ "▲" for all $c \in C$. The k^{th} *iterated* *degree*[2] ($k \geq 1$) of a node v is defined as $\delta_k(v) = \{(\ell_{v,w}, \delta_{k-1}(w)) \mid w \in N(v)\}$. We write $v \sim_\delta^k v'$ if $\delta_k(v) = \delta_k(v')$, and simply $v \sim_\delta v'$ if $v \sim_\delta^k v'$ for all $k \geq 0$. In this case, we say that v and v' are *iterated degree equivalent*. It can be shown (see the full version) that as k increases, the partition induced by $\sim_\delta^k$ gets more refined, and indeed it reaches a fixed point for some $k \leq 2n$ where $n = |X|$. The notion of iterated degree is strikingly relevant in our work as it captures what it means for two processes in a network to be indistinguishable. This implies that no distributed algorithm can differentiate between two iterated degree equivalent nodes, as we illustrate in the following result.

▶ **Proposition 2.2.** *Let $I = (A, X, D, C, \alpha)$ be an instance of* DCSP(Γ) *whose factor graph is not necessarily connected and consider two variables $v, v' \in G_I$. Then, $v \sim_\delta v'$ if and only if any terminating decision algorithm over I outputs the same decision at $\alpha(v)$ and $\alpha(v')$. Furthermore, if $v, v' \in X$ and I is satisfiable, then any terminating search algorithm outputs the same values $\nu(v) = \nu(v')$ at $\alpha(v)$ and $\alpha(v')$.*

The following is a direct consequence of Proposition 2.2. We say that two instances I and I' have the same *iterated degree sequence* if there exists a bijection γ between the nodes of G_I and the nodes of $G_{I'}$ such that for every $k \geq 0$ and every node v of G_I, the k^{th} degree of v in I is equal to the k^{th} degree of $\gamma(v)$ in I'. We note that in this case, if we construct the (disconnected) instance $I \cup I'$ containing all the variables and constraints in I and I', then $v \sim_\delta \gamma(v)$ for every node $v \in G_I$. Hence the result below follows.

▶ **Corollary 2.3.** *Let $I, I' \in$* DCSP(Γ) *have the same iterated degree sequence. Then with both inputs any terminating decision algorithm will report the same decision.*

Polymorphisms

Let R be a k-ary relation over a finite domain D. An ℓ-ary *polymorphism* of R is an operation $f : D^\ell \to D$ such that the coordinate-wise application of f to any set of ℓ tuples from R gives a tuple in R. More precisely, for any $\mathbf{t}_1, \ldots, \mathbf{t}_\ell \in R$, we have that $(f(\mathbf{t}_1[1], \ldots, \mathbf{t}_\ell[1]), \ldots, f(\mathbf{t}_1[k], \ldots, \mathbf{t}_\ell[k])) \in R$. We say that a function f is a polymorphism of a constraint language Γ if f is a polymorphism of all relations $R \in \Gamma$. Equivalently, we say that Γ is *invariant* under f. The set of polymorphisms of a constraint language Γ will be denoted by $\mathsf{Pol}(\Gamma)$. There is a particular construction of a CSP instance that is closely related to the clone of polymorphisms of the corresponding constraint language. Let Γ be a constraint language over a finite domain D. For any positive integer r, the *indicator problem of order* r for Γ is the instance $I = (X, D, C) \in$ CSP(Γ) where $X = D^r$ and C contains for every relation $R \in \Gamma$ and for every $\mathbf{t}_1, \ldots, \mathbf{t}_r \in R$, the constraint $(\mathbf{s}, R)$ where $\mathbf{s}[i] = (\mathbf{t}_1[i], \ldots, \mathbf{t}_r[i])$ for every $i \in \{1, \ldots, arity(R)\}$. It follows easily that for every $\nu : D^r \to D$, ν satisfies I if and only if ν is a polymorphism of Γ.

An ℓ-ary operation f is said to be *symmetric* if for all $x_1, \ldots, x_\ell$ and for all permutations σ of $\{1, \ldots, \ell\}$ we have that $f(x_1, \ldots, x_\ell) = f(x_{\sigma(1)}, \ldots, x_{\sigma(\ell)})$.

▶ **Example 2.4.** Consider the Boolean relation $R = \{(0, 1), (1, 0)\}$. It is easy to see that the ternary minority operation f given by $f(x, y, z) = x \oplus y \oplus z$ is a polymorphism of R. On the other hand, one can show that R does not have symmetric polymorphisms of arity 2. In particular, let $\mathbf{t}_1 = (0, 1)$ and $\mathbf{t}_2 = (1, 0)$. Since a symmetric binary operation f needs to satisfy $f(0, 1) = f(1, 0)$, the coordinate-wise application of f to $\mathbf{t}_1, \mathbf{t}_2$ would yield a reflexive tuple, which cannot possibly belong to R. ⌟

[2] We remark that the notions of degree and iterated degree are well-defined concepts in graph theory. We borrow this terminology to refer to the analogous concepts in CSPs.

Our work unveils a novel structure in the space of solutions of a CSP instance that is deeply connected to the symmetry of its polymorphisms. In particular, $\mathsf{Pol}(\Gamma)$ containing symmetric polymorphisms of all arities is equivalent to the existence of a satisfying assignment to every satisfiable instance of $\mathrm{CSP}(\Gamma)$ that preserves the partition induced by $\sim_\delta$. This is the main result of the next section.

3 Basic Linear Programming relaxation

For any CSP instance $I = (X, D, C)$ there is a LP relaxation (usually called *basic LP relaxation*, see for example [18]) denoted $\mathrm{BLP}(I)$, which is defined as follows. It has a variable $v(x, d)$ for each $x \in X$ and $d \in D$, and a variable $v(c, \mathbf{t})$ for each $c \in C$ and $\mathbf{t} \in R$ where R is the constraint relation of c. All variables must take values in the range $[0, 1]$. The value of $v(x, d)$ is interpreted as the probability that v is assigned to d. Similarly, the value of $v(c, \mathbf{t})$ is interpreted as the probability that the scope of c is assigned component-wise to the tuple $\mathbf{t}$. In this paper we only deal with a feasibility problem (that is, there is no objective function). The variables are restricted by the following equations:

$$\sum_{d\in D} v(x,d) = 1 \qquad \text{for all } x \in X \tag{1}$$

$$\sum_{\substack{\mathbf{t}\in R_c\\ \mathbf{t}[i]=d}} v(c,\mathbf{t}) - v(\mathbf{s}_c[i], d) = 0 \qquad \text{for all } c \in C, \text{ all } i \in \{1,\dots,\mathit{arity}(c)\}, \text{ and all } d \in D \tag{2}$$

where we denote the relation and scope of a constraint c by R_c and $\mathbf{s}_c$ respectively. We say that BLP decides $\mathrm{CSP}(\Gamma)$ if for every instance $I \in \mathrm{CSP}(\Gamma)$, I is satisfiable whenever $\mathrm{BLP}(I)$ is feasible. We will use the following well-known result.

▶ **Theorem 3.1** (see [18]). *If Γ has symmetric polymorphisms of all arities, then BLP decides* $\mathrm{CSP}(\Gamma)$*. Moreover, if $I \in \mathrm{CSP}(\Gamma)$ is satisfiable then it has a solution ν such that for all x, x' with $v(x, d) = v(x', d)$ for all $d \in D$, we have $\nu(x) = \nu(x')$.*

The following theorem reveals a useful structure inside the solutions of the BLP.

▶ **Theorem 3.2.** *Let $I = (X, D, C)$ be an instance of* $\mathrm{CSP}(\Gamma)$ *such that* $\mathrm{BLP}(I)$ *is feasible. Then,* $\mathrm{BLP}(I)$ *has a feasible solution such that for every $x, x' \in X$ with $x \sim_\delta x'$ and every $d \in D$, $v(x, d) = v(x', d)$.*

Proof (Sketch). We start by rewriting the program in the form

$$\exists ? \mathbf{v} \in [0,1]^V \quad B\mathbf{v} \geq \mathbf{b} \tag{3}$$

by replacing every equality $a = b$ by the inequalities $a \geq b$ and $-a \geq -b$.

Let us use W and V to denote the rows and columns of B respectively. The main idea of the proof is to apply the Multiplicative Weight Update (MWU) algorithm, a well-known technique that is widely used in optimisation and machine learning. MWU was discovered independently by researchers of different communities; for a survey of its different variants we refer the reader to [2]. The version that is relevant to our paper is described in Algorithm 1. Assuming that a feasible solution to (3) does exist, the algorithm only requires the existence of an oracle which, given a probability W-vector $\mathbf{p}$ (i.e, a non-negative vector $\mathbf{p}$ such that the sum of all its entries is 1), outputs a vector $\mathbf{v}$ which is a solution to the weaker problem

$$\exists ? \mathbf{v} \in [0,1]^V \quad \mathbf{p}^T B\mathbf{v} \geq \mathbf{p}^T\mathbf{b} \tag{4}$$

if one exists, or correctly states that no such vectors exist otherwise.

Algorithm 1 Multiplicative Weight Update.

```
Initialisation: Fix $\eta \leq \frac{1}{2}$, let $\rho = \max_{[0,1]^V} \max_{w \in W} |B_w \mathbf{v} - \mathbf{b}[w]|$, and let $\mathbf{w}^{(1)}$ be
  a $W$-vector, whose entries, called weights, are initially set to 1.
for $t = 1, \ldots, T$ do
    Compute the probability vector $\mathbf{p}^{(t)} = \frac{1}{\Phi(t)} \mathbf{w}^{(t)}$, where $\Phi(t) = \sum_{j=1}^{|W|} \mathbf{w}^{(t)}[j]$
    Let $\mathbf{v}^{(t)}$ be a solution satisfying $(\mathbf{p}^{(t)})^T B \mathbf{v}^{(t)} \geq (\mathbf{p}^{(t)})^T \mathbf{b}$ given by oracle $\mathsf{O}$
    Compute the losses $\boldsymbol{\ell}^{(t)} = \frac{1}{\rho}(B\mathbf{v}^{(t)} - \mathbf{b})$
    Compute the new weights $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)}(1 - \eta \boldsymbol{\ell}^{(t)})$
end
return $\mathbf{v} := \frac{1}{T} \sum_{t=1}^{T} \mathbf{v}^{(t)}$
```

Under some technical conditions that provide an upper bound on the number of rounds T necessary to achieve a given approximation (see full version for details) it follows that when $T \to \infty$ MWU converges to a solution of BLP(I). Now consider an oracle O that, given a W-vector $\mathbf{p}$, returns the V-vector $\mathbf{v}$ where for every $v \in V$, $\mathbf{v}[v] = 1$ if $\mathbf{p}^T B[v]$ is positive and $\mathbf{v}[v] = 0$ otherwise. Since $\mathbf{v}$ maximizes $\mathbf{p}^T B \mathbf{v}$ under the restriction $\mathbf{v} \in [0,1]^V$ it follows that $\mathbf{v}$ satisfies (4).

We note that $\sim_\delta$ induces an equivalence relation on the variables of BLP(I) (namely, $v(x,d)$ is equivalent to $v(x',d')$ whenever $x \sim_\delta x'$ and $d = d'$) which can be extended to an equivalence relation $\sim_V$ on the set V of columns in B. Similarly, $\sim_\delta$ induces an equivalence relation $\sim_W$ on the rows W of B in a natural way. Then our goal is to show that the positions of $\sim_V$-equivalent entries in the output $\mathbf{v} := \frac{1}{T} \sum_{t=1}^{T} \mathbf{v}^{(t)}$ are identical. This is done by showing by induction the more general fact that at each iteration t of the algorithm, the positions of all $\sim_V$-equivalent entries in $\mathbf{v}^{(t)}$ are identical, and that for each of the W-vectors ($\mathbf{w}^{(t)}$, $\mathbf{p}^{(t)}$, and $\boldsymbol{\ell}^{(t)}$) the positions of all $\sim_W$-equivalent entries are identical as well. ◀

We finalize the section by presenting the theorem on the structure of the solution space of CSP instances.

▶ **Theorem 3.3.** *Let Γ be a finite constraint language. The following are equivalent:*

1. *Γ has symmetric polymorphisms of all arities.*
2. *For all satisfiable instances $I = (X, D, C) \in \mathrm{CSP}(\Gamma)$ there exists a satisfying assignment $\nu : X \to D$ such that for all pairs of variables $x, x' \in X$, if $x \sim_\delta x'$ then $\nu(x) = \nu(x')$.*

Proof. (1) ⇒ (2). Let I be a satisfiable instance of CSP(Γ), where Γ has symmetric polymorphisms of all arities. Consider the solution of BLP(I) given by Theorem 3.2 and note that it satisfies $v(x,d) = v(x',d)$ for all $x \sim_\delta x'$ and all $d \in D$. Then, by Theorem 3.1, I has a solution ν which satisfies $\nu(x) = \nu(x')$ for all $x \sim_\delta x'$.

(2) ⇒ (1). Let Γ satisfy (2) and let $r \geq 1$. We shall prove that Γ has a symmetric polymorphism of arity r. Let $I = (X, D, C)$ be the indicator problem of order r. Recall that every solution to I corresponds to an r-ary polymorphism of Γ, and hence the indicator problem is always satisfiable since for instance the projection to the first coordinate is a polymorphism of Γ. Let ν be a solution of the indicator problem which satisfies condition (2). It is easy to show by induction that for every tuple $(t_1 \ldots, t_r) \in D^r$, every permutation σ of $\{1, \ldots, r\}$ and every $k \geq 0$, $(t_1, \ldots, t_r) \sim_\sigma^k (t_{\sigma(1)}, \ldots, t_{\sigma(r)})$ which implies that $\nu(t_1, \ldots, t_r) = \nu(t_{\sigma(1)}, \ldots, t_{\sigma(r)})$. We conclude that ν is symmetric as required. ◀

4 The Complexity of DCSP

The primary goal of this section is to prove the main theorem of this paper, namely, the dichotomy theorem for tractability of DCSP(Γ), which we now state.

▶ **Theorem 4.1.** DCSP(Γ) *is solvable in polynomial time if and only if* Pol(Γ) *contains symmetric polymorphisms of all arities. Otherwise,* DCSP(Γ) *cannot be solved in finite time.*

We show hardness of constraint languages that do not have symmetric polymorphisms of all arities in Section 4.1 and tractability of the remaining languages in Section 4.2. In addition, in Section 4.3 we extend the decision algorithm so that, additionally, it also provides a solution to the search problem. Hence we have:

▶ **Theorem 4.2.** DCSP-Search(Γ) *is solvable in polynomial time if and only if* Pol(Γ) *contains symmetric polymorphisms of all arities. Otherwise,* DCSP-Search(Γ) *cannot be solved in finite time.*

4.1 Intractable Languages

In this section we focus on intractable languages, that is, the hardness part of Theorem 4.1.

▶ **Theorem 4.3.** *Let Γ be a constraint language on a finite domain D. If* Pol(Γ) *does not contain symmetric operations of all arities, then there is no algorithm that solves* DCSP(Γ) *in finite time.*

Schematically, the proof goes as follows. Assume that Γ does not have symmetric polymorphisms of some arity r. Consider the relation U defined by the set of solutions of the indicator problem of order r. It can be shown that if DCSP(Γ) is solvable in polynomial (or finite) time then so is DCSP($\{U\}$). Then, we show that there always exist two instances of DCSP($\{U\}$), one which is satisfiable and the other one which is not, that have the same iterated degree sequence. Therefore, any algorithm will return the same output on both instances, meaning that one of these outputs is wrong. Before embarking on the proof we state the following useful combinatorial lemma.

▶ **Lemma 4.4.** *Let $0 < k < d$ be positive integers. If n is a large enough multiple of k, then there exists a collection $\mathbb{S}$ of n^k k-element subsets of $\{0, 1, \ldots, kn-1\}$ satisfying the following properties:*

(a) *$\mathbb{S}$ contains every k-element subset of $\{0, \ldots, d-1\}$*

(b) *Every element of $\{0, 1, \ldots, kn-1\}$ appears in the same number of sets of $\mathbb{S}$.*

Proof of Theorem 4.3. Assume that Pol(Γ) does not contain symmetric polymorphisms of arity r. Fix any arbitrary order $\mathbf{t}_1, \ldots, \mathbf{t}_{|D|^r}$ on the tuples of D^r and consider the relation U defined as

$$\{(f(\mathbf{t}_1), \ldots, f(\mathbf{t}_{|D|^r})) \mid f \text{ is a polymorphism of } \Gamma \text{ of arity } r\}$$

This is, U is the set of solutions of the indicator problem of order r. It follows easily (see full version) that if DCSP($\{U\}$) is not solvable in finite time then neither is DCSP(Γ). In particular, this follows from an adaptation of standard complexity reductions, given that U is pp-definable from Γ without using equality.

Partition D^r into equivalence classes where two tuples $\mathbf{t}, \mathbf{t}' \in D^r$ are related, denoted $\mathbf{t} \equiv \mathbf{t}'$, if there exists some permutation σ on $\{1, \ldots, r\}$ such that $\mathbf{t}'[i] = \mathbf{t}[\sigma(i)]$ for every $i \in \{1, \ldots, r\}$. We shall use $D^r_{\equiv}$ to refer to the collection of classes and $[\mathbf{t}]_{\equiv}$ to refer to the class of tuple $\mathbf{t}$. For every $\mathbf{t} \in D^r$, define $k_{[\mathbf{t}]_\equiv}$ to be the number of tuples in $[\mathbf{t}]_\equiv$. Then we can choose an integer n large enough such that for every $\mathbf{t} \in D^r$, n is a multiple of $k_{[\mathbf{t}]_\equiv}$, and n satisfies Lemma 4.4 for $k = k_{[\mathbf{t}]_\equiv}$ and $d = k_{[\mathbf{t}]_\equiv} \cdot |D|$.

We are now ready to construct two instances I_1 and I_2 of DCSP($\{U\}$), which are indistinguishable with respect to their iterated degree sequence, but differ with regards to satisfiability. The two instances have the same set of variables, defined to be $\bigcup_{[\mathbf{t}]_\equiv \in D^r_\equiv} V_{[\mathbf{t}]_\equiv}$ where $V_{[\mathbf{t}]_\equiv} = \{v^i_{[\mathbf{t}]_\equiv} \mid 0 \leq i < k_{[\mathbf{t}]_\equiv} n\}$ is a set of $k_{[\mathbf{t}]_\equiv} n$ distinct variables.

We start by constructing the constraints of the unsatisfiable instance I_1, which we will do in two stages. First, for every class $[\mathbf{t}]_\equiv$, let $\mathbb{S}_{[\mathbf{t}]_\equiv}$ be the collection of $n^{k_{[\mathbf{t}]_\equiv}}$ sets of cardinality $k_{[\mathbf{t}]_\equiv}$ given by Lemma 4.4, as before with $d = k_{[\mathbf{t}]_\equiv} \cdot |D|$ and $k = k_{[\mathbf{t}]_\equiv}$. Note that each set in $\mathbb{S}_{[\mathbf{t}]_\equiv}$ defines naturally a subset of $V_{[\mathbf{t}]_\equiv}$ so we shall abuse notation and assume that $\mathbb{S}_{[\mathbf{t}]_\equiv}$ is a collection of subsets of $V_{[\mathbf{t}]_\equiv}$.

To simplify notation it will be convenient to use $\mathbb{S}$ as a shorthand for the indexed family $\{\mathbb{S}_{[\mathbf{t}]_\equiv} \mid [\mathbf{t}]_\equiv \in D^r_\equiv\}$. Now let S be $\{S_{[\mathbf{t}]_\equiv} \mid [\mathbf{t}]_\equiv \in D^r_\equiv\}$ satisfying $S_{[\mathbf{t}]_\equiv} \in \mathbb{S}_{[\mathbf{t}]_\equiv}$ for every $[\mathbf{t}]_\equiv \in D^r_\equiv$. We associate to S the constraint $(\mathbf{s}, U)$ where the scope $\mathbf{s}$ is constructed as follows. Before defining $\mathbf{s}$ we need some preparation. Recall that every coordinate of U, and hence of $\mathbf{s}$, is associated to a tuple $\mathbf{t} \in D^r$, so we can talk of the class $[\mathbf{t}]_\equiv$ to which each coordinate belongs. In particular, there are $k_{[\mathbf{t}]_\equiv}$ coordinates in $\mathbf{s}$ of class $[\mathbf{t}]_\equiv$. Hence, by fixing some arbitrary ordering we can use $\mathbf{s}^i_{[\mathbf{t}]_\equiv}$, $i = 1, \ldots, k_{[\mathbf{t}]_\equiv}$ to refer to the coordinates in $\mathbf{s}$ of class $[\mathbf{t}]_\equiv$. Then, informally, $S_{[\mathbf{t}]_\equiv}$ describes which variables from $v^0_{[\mathbf{t}]_\equiv}, \ldots, v^{k_{[\mathbf{t}]_\equiv} n-1}_{[\mathbf{t}]_\equiv}$ to use in order to fill coordinates $\mathbf{s}^i_{[\mathbf{t}]_\equiv}$, $i = 1, \ldots, k_{[\mathbf{t}]_\equiv}$. Formally, for every $[\mathbf{t}]_\equiv \in D^r_\equiv$ and each $i = 1, \ldots, k_{[\mathbf{t}]_\equiv}$, $\mathbf{s}^i_{[\mathbf{t}]_\equiv}$ is assigned to the i^{th} element in $S_{[\mathbf{t}]_\equiv}$ in increasing order.

We add such a constraint for each of the $\Pi_{[\mathbf{t}]_\equiv \in D^r_\equiv} n^{k_{[\mathbf{t}]_\equiv}} = n^{(|D|^r)}$ possible choices for S. Therefore, after the first stage we have exactly $n^{(|D|^r)}$ constraints.

In the second stage we add more constraints which will yield the particular symmetry of I_1. Note that every permutation σ on $\{1, \ldots, r\}$ induces a permutation σ' on the coordinates of U in a natural way. Specifically, if coordinate i of U is associated to tuple $\mathbf{t}_i$, then $\sigma'(i) = j$ where $\mathbf{t}_j = (\mathbf{t}_i[\sigma(1)], \ldots, \mathbf{t}_i[\sigma(r)])$. Then, in the second stage, for each permutation σ on $\{1, \ldots, r\}$ and for every constraint $(\mathbf{s}, U)$ added in the first stage we add the constraint $(\mathbf{s}', U)$ where for every $1 \leq i \leq |D|^r$, $\mathbf{s}'[i] = \mathbf{s}[\sigma'(i)]$. Therefore, after the second stage we have a total of $m = r! \cdot n^{(|D|^r)}$ constraints as needed.

We now turn to I_2. The constraints are constructed in a similar way, but instead of using the family $\mathbb{S}$ in the first stage, we use a different family $\mathbb{S}'$. In particular, for each class $[\mathbf{t}]_\equiv$, ${\mathbb{S}'}_{[\mathbf{t}]_\equiv}$ is obtained by partitioning $V_{[\mathbf{t}]_\equiv}$ in $k_{[\mathbf{t}]_\equiv}$ blocks of consecutive elements, so that each block has exactly n elements. Then, ${\mathbb{S}'}_{[\mathbf{t}]_\equiv}$ contains the $n^{k_{[\mathbf{t}]_\equiv}}$ sets that can be obtained by selecting one element from each block. The second stage is done exactly as in I_1.

▷ Claim 4.5. I_1 and I_2 have the same iterated degree sequence.

Proof. Let $[\mathbf{t}]_\equiv \in D^r_\equiv$. First, we observe that in both instances after the first stage, every variable of $V_{[\mathbf{t}]_\equiv}$ appears in the same number of constraints. More specifically, every variable in $V_{[\mathbf{t}]_\equiv}$ appears in an n-fraction of the constraints added in stage 1. In the case of instance I_1 this is due to the fact that $\mathbb{S}_{[\mathbf{t}]_\equiv}$ satisfies condition (b) in Lemma 4.4 and in instance I_2 this follows from the fact that $\mathbb{S}'_{[\mathbf{t}]_\equiv}$ contains all possible sets obtained by choosing an element within each one of the blocks of size n. After the second stage (in both I_1 and I_2 since the second stage is common) every variable in $V_{[\mathbf{t}]_\equiv}$ still participates in an n-fraction of the total number of constraints. In addition, it follows easily that the positions of the scope in which a variable in $V_{[\mathbf{t}]_\equiv}$ participates distribute evenly among the $k_{[\mathbf{t}]_\equiv}$ positions associated to $\mathbf{t}$. That is, in both instances, we have that for every $[\mathbf{t}]_\equiv \in D^r_\equiv$, every variable $x \in V_{[\mathbf{t}]_\equiv}$, and every position i associated to $[\mathbf{t}]_\equiv$ there are exactly $\frac{m}{nk_{[\mathbf{t}]_\equiv}}$ constraints in which x appears at position i of the scope, where $m = r! \cdot n^{|D|^r}$. Using this fact it is very easy to prove that I_1 and I_2 have the same iterated degree sequence. Formally, one could show by induction on k that for every $[\mathbf{t}]_\equiv \in D^r_\equiv$ and $x_1, x_2 \in V_{[\mathbf{t}]_\equiv}$, $\delta_k^{I_1}(x_1) = \delta_k^{I_2}(x_2)$ and that for any two constraints c_1, c_2 in I_1 and I_2 respectively $\delta_k^{I_1}(c_1) = \delta_k^{I_2}(c_2)$. Here we are using $\delta_k^{I_1}(\cdot)$ and $\delta_k^{I_2}(\cdot)$ to denote the k^{th} degree of a node in the factor graphs of I_1 and I_2 respectively. ◁

▷ Claim 4.6. Instance I_1 is unsatisfiable while instance I_2 is satisfiable.

Proof. We start by showing that I_1 is not satisfiable. Assume by contradiction that I_1 has a satisfying assignment ν. For each class $[\mathbf{t}]_\equiv$, consider the values given by ν to the first d variables $v_0, \ldots, v_{d-1}$ in $V_{[\mathbf{t}]_\equiv}$. Since $d = k_{[\mathbf{t}]_\equiv} \cdot |D|$, it follows by the pigeon-hole principle that at least $k_{[\mathbf{t}]_\equiv}$ of these variables are assigned by ν to the same value of D. Let $S_{[\mathbf{t}]_\equiv}$ be a subset of $V_{[\mathbf{t}]_\equiv}$ containing $k_{[\mathbf{t}]_\equiv}$ of these variables (we know that this subset belongs to $\mathbb{S}_{[\mathbf{t}]_\equiv}$ by condition (a) of Lemma 4.4). Now consider the constraint $(\mathbf{s}, U)$ in I_1 associated to $S := \{S_{[\mathbf{t}]_\equiv} \mid [\mathbf{t}]_\equiv \in D^r_\equiv\}$, which belongs to I_1. If ν is a solution to I_1, then the restriction of ν to $\mathbf{s}$ corresponds to an r-ary polymorphism of Γ. But ν assigns the same value to any two related tuples $\mathbf{t} \equiv \mathbf{t}'$, which implies that ν is symmetric, a contradiction.

We now turn our focus to I_2. Let f be any r-ary polymorphism of Γ (for example the i^{th} $(1 \leq i \leq r)$ projection operation defined as $f(x_1, \ldots, x_r) = x_i$). We shall construct a solution ν of I_2 in the following way. Recall that in the definition of I_2 we have partitioned the tuples of $V_{[\mathbf{t}]_\equiv}$ in $k_{[\mathbf{t}]_\equiv}$ consecutive blocks. In the first stage, all the elements in each block are placed in the same coordinate of U. So, if $\mathbf{t}_1, \ldots, \mathbf{t}_{|D|^r}$ are the tuples associated to coordinates $1, \ldots, |D|^r$ and hence block $1, \ldots, |D|^r$ respectively, then we only need that all variables in the i^{th} block are assigned to $f(\mathbf{t}_i)$ to satisfy all constraints added in the first stage. This assignment also satisfies the constraints added in the second stage, because if f is an r-ary polymorphism of Γ, then for every permutation σ on $\{1, \ldots, r\}$, the operation $g(x_1, \ldots, x_r)$ defined as $f(x_{\sigma(1)}, \ldots, x_{\sigma(r)})$ is also a polymorphism of Γ. ◁

To sum up, we constructed two instances I_1 and I_2, the latter of which is satisfiable while the former is not, which have the same iterated degree sequence. It follows from Corollary 2.3 that any distributed algorithm will give the same output on both instances, meaning that no algorithm can solve DCSP$(\{U\})$. As anticipated at the beginning of the proof then it follows that there are also no algorithms that solve DCSP(Γ). ◀

4.2 Tractable Languages

In this section we turn our attention to the tractable case. In particular we shall show the following:

▶ **Theorem 4.7.** *Let Γ be a constraint language that is invariant under symmetric polymorphisms of all arities. Then there is an algorithm* Alg *that solves* DCSP(Γ)*. The total running time, number of rounds, and maximum message size of* Alg *are, respectively,* $\mathcal{O}(n^3 m \log n)$*,* $\mathcal{O}(n^2)$*, and* $\mathcal{O}(m \log n)$ *where n and m are the number of variables and constraints, respectively, of the input instance.*

Note that this implies the "if" part of Theorem 4.1. Alg is composed of two phases. In the first phase, a distributed version of the colour refinement algorithm allows every process to calculate its iterated degree. Then, thanks to Theorem 3.3 we can use the degree of a variable as its ID for the second phase, implying that a distributed adapted version of the *jpq*-consistency algorithm [16] where messages are tagged with a process' iterated degree solves the decision problem for Γ.

Distributed Colour Refinement

Let $I = (A, X, D, C, \alpha)$ be an instance of DCSP(Γ) and let $n = |X|$ and $m = |C|$. There is a very natural way to calculate an agent's iterated degree in a distributed way, both for variables and for constraints. This is a mere adaptation of the 1-dimensional Weisfeiler-Leman algorithm, also known as colour refinement, an algorithm that partitions the vertices of a graph into classes by iteratively distinguishing them on the basis of their degree (see for example [15, 14]). The algorithm proceeds in rounds. At round $k = 0$, each agent $\alpha(v)$ for $v \in X \cup C$ computes $\delta_0(v)$ and broadcasts it to all its neighbours. At round $k > 0$, each agent $\alpha(v)$ knows the $(k-1)^{th}$ degrees of its neighbours which it had received in the previous round, uses them to compute $\delta_k(v)$, and broadcasts it to its neighbours. If $k = 2n$ then for every $x, x' \in X$ satisfying $x \sim_\delta^k x'$ we have that $x \sim_\delta x'$, which implies that we can essentially regard the k^{th} iterated degree as the unique common ID for all variables that are iterated degree equivalent. Then in $2n$ rounds each agent $\alpha(v)$ can compute $\delta_\infty(v)$, where we use δ_∞ as a shorthand for δ_{2n}. As we described it, the distributed colour refinement algorithm is not particularly efficient in terms of message complexity. Although this is not necessary to achieve polynomial time, we can reduce the space required to encode $\delta_\infty(v)$.

▶ **Lemma 4.8.** *Let $s_{\max}$ denote the size of the encoding of $\delta_\infty(v)$. A modified version of the distributed colour refinement algorithm that runs over $\mathcal{O}(n^2)$ rounds achieves $s_{\max} = \mathcal{O}(\log n)$. The time at each round and the maximum size of a message are both bounded above by $\mathcal{O}(ms_{max})$.*

As we will see, the price of an increase in the number of rounds (from n to n^2) is compensated by the effect of $s_{\max}$ on both time complexity and the size of the messages.

The Distributed Consistency Algorithm

It is well known that if a constraint language Γ has symmetric operations of all arities then it satisfies the so-called *bounded width* property (see [4]). We avoid introducing the definition of bounded width as it is not needed in our results and refer the reader to [4] for reference. Then, it has been shown in [16] that if Γ has bounded width and $I \in$ CSP(Γ) satisfies a combinatorial condition called *jpq*-consistency, then I has a solution. Instead of stating literally the result in [16] we shall state a weaker version that uses a different notion of consistency, more suitable to the model of distributed computation introduced in the paper.

A *set system* S is a subset of $X \times D$. We shall use S_x to denote the set $\{d \in D \mid (x, d) \in S\}$. A walk of length ℓ (in instance I) is any sequence $x_0 c_0 \ldots c_{\ell-1} x_\ell$ where $x_0, \ldots, x_\ell$ are variables, $c_0, \ldots, c_{\ell-1}$ are constraints, and $x_i, x_{i+1} \in c_i$ for every $0 \leq i < \ell$. Note that walks are precisely the walks in the factor graph G_I (in the standard graph-theoretic sense) starting and finishing in X.

Let S be a set system, p be a walk, and $B \subseteq S_x$ where x is the starting node of p. The *propagation* of B via p under S, denoted $B +_S p$, is the subset of D defined inductively on the length ℓ of p as follows. If $\ell = 0$ then $B +_S p = B$. Otherwise, $p = p' c_{\ell-1} x_\ell$ where p' is a path of length $\ell - 1$ ending at $x_{\ell-1}$. Let $c_{\ell-1} = (\mathbf{s}, R)$. Then we define $B +_S p$ to contain all $e \in D$ such that there exists $d \in B +_S p'$ and $\mathbf{t} \in R$ such that for every $1 \leq i \leq arity(R)$, $\mathbf{t}[i]$ satisfies the following conditions:

1. $\mathbf{t}[i] \in S_{\mathbf{s}[i]}$,
2. if $\mathbf{s}[i] = x_{\ell-1}$ then $\mathbf{t}[i] = d$, and
3. if $\mathbf{s}[i] = x_\ell$ then $\mathbf{t}[i] = e$.

We are now ready to state the result from [16] that we shall use.

▶ **Theorem 4.9** (follows from [16]). *Let I be an instance of* $\mathrm{CSP}(\Gamma)$ *where* Γ *has bounded width and let S be a set system such that $S_x \neq \emptyset$ for every $x \in X$ and such that for every walk p starting and finishing at the same node x and for every $d \in S_x$, d belongs to $\{d\} +_S p$. Then I is satisfiable.*

Our goal is to design a distributed algorithm that either correctly determines that an instance I is unsatisfiable, or produces a set system S verifying the conditions of Theorem 4.9. This is not possible in general due to the fact that agents are anonymous and hence a hypothetical algorithm that would generate a walk in a distributed way would be unable to determine if the initial and end nodes are the same. However, thanks to the structure established by Theorem 3.3, this difficulty can be overcome when Γ has symmetric polymorphisms of all arities because, essentially, the iterated degree of a node can act as its unique identifier. To make this intuition precise we will need to introduce a few more definitions.

We say that a pair $(x, d) \in S$ is *S-supported* if for every walk p starting at x and finishing at a node y with $x \sim_\delta y$, we have that $\{d\} +_S p$ contains d.

▶ Remark 4.10. We note that if $(x, d) \in S$ is not S-supported and $p = x_0 c_0 \dots x_\ell$ is a walk of minimal length among all walks witnessing that (x, d) is not S-supported then $\ell \leq n2^{|D|}$. Indeed if we let $B_i = \{d\} + x_0 c_0 \dots x_i$, $i = 0, \dots, \ell$ then we have that $(x_i, B_i) \neq (x_j, B_j)$ for every $0 \leq i < j \leq \ell$, since otherwise the shorter walk $x_0 c_0, \dots, x_i, c_j, \dots, x_\ell$ would contradict the minimality of p. Since there are n choices for each x_i and $2^{|D|}$ choices for B_i, the bound follows. ⌟

We say that a set system S is *safe* if for every solution $\nu \in I$ we have

$$\nu(x) = \nu(y) \text{ for all } x, y \in X \text{ with } x \sim_\delta y \implies \nu(x) \in S_x \text{ for all } x \in X.$$

Then, we have

▶ **Lemma 4.11.** *Let S be a safe set system and let $(x, d) \in S$ be a pair that is not S-supported. Then $S \setminus \{(x, d)\}$ is safe.*

Our distributed consistency algorithm (that is, the second phase of Alg) works as follows. Every variable agent $\alpha(x)$ maintains a set $S_x \subseteq D$ in such a way that the set system S is guaranteed to be safe at all times. As a result of an iterative process S is modified. We shall use S^i to denote the content of S at the i^{th} iteration, where an iteration is, in turn, a loop of $T = 2n(2^{|D|} + 1) = \mathcal{O}(n)$ consecutive rounds. The rationale behind this exact value will be made clear later. Initially, S^0_x is set to D for every $x \in X$. At iteration i for $i \geq 1$, S^i is obtained by removing all the elements in S^{i-1} that are not S^{i-1}-supported. Then, in at most $n|D| = \mathcal{O}(n)$ iterations we shall obtain a fixed point S^∞.

The key observation is that when Γ has symmetric polymorphisms of all arities, the satisfiability of I can be determined from S^∞. Indeed, if $S^\infty_x = \emptyset$ for some $x \in X$ then we can conclude from the fact that S^∞ is safe and Theorem 3.3 that I has no solution. Otherwise, S^∞ satisfies the conditions of Theorem 4.9 and, hence, I is satisfiable.

It remains to see how to compute S^{i+1} from S^i. In an initial preparation step for every iteration, every variable agent $\alpha(x)$ sends S^i_x to all its neighbours. To compute S^{i+1} the algorithm proceeds in rounds. All the messages sent are sets containing triplets of the form (δ_∞, d, B) where $d \in D$, $B \subseteq D$, and δ_∞ is the iterated degree of some variable $x \in X$. It follows from the fact that there are at most n possibilities for the degree of a variable that the size of each message is $\mathcal{O}(ns_{\max})$.

The agents controlling variables and constraints alternate. That is, variables perform internal and send events at even rounds and receive messages at odd rounds, while constraints perform internal and send events at odd rounds and receive messages at even rounds. More specifically, in round $j = 0$ of iteration i, every variable agent $\alpha(x)$ sends to its neighbours the message M containing all triplets of the form $(\delta_\infty(x), d, \{d\})$ with $d \in S^i_x$. At round $2j$ for $j > 0$, $\alpha(x)$ computes $M = M_1 \cup \cdots \cup M_r$ where $M_1, \ldots, M_r$ are the messages it received at the end of round $2j - 1$. Subsequently, for every triplet $(\delta_\infty, d, B) \in M$ with $\delta_\infty = \delta_\infty(x)$ and $d \notin B$, $\alpha(x)$ marks d as "not S^i-supported". Finally, it sends message M to all its neighbours. This computation can be done in time $\mathcal{O}(rns_{\max}) = \mathcal{O}(mns_{\max})$ provided that each message is stored as an ordered array.

In round $2j + 1$, every constraint agent $\alpha(c)$ computes from the messages M_x (received from each neighbour $\alpha(x)$ in the previous round) the set M'_x, which contains for every variable $y \in c$ and every (δ_∞, d, B) in M_y, the triplet $(\delta_\infty, d, B +_{S^i} p)$ where $p = y, c, x$. Finally, it sends to each neighbour $\alpha(x)$ the corresponding message M'_x. Note that while $\alpha(c)$ doesn't know the address of $\alpha(x)$ specifically, knowing the label of the channel that connects them is sufficient to calculate M'_x correctly and send the message accordingly. Moreover, for given y and x, $\alpha(c)$ can compute $B +_{S^i} p$ in $\mathcal{O}(1)$ time as $\alpha(c)$ knows both S^i_y and S^i_x. Hence, since the arity of the relations is fixed (as Γ is fixed) the total running time at iteration $2j + 1$ of a constraint agent $\alpha(c)$ is $\mathcal{O}(ns_{\max})$.

Now it is immediate to show by induction that for every $j \geq 0$, every $x \in X$ and $c \in C$ with $x \in c$ the message sent by $\alpha(x)$ to $\alpha(c)$ at the end of round $2j$ is precisely

$$\{(\delta_\infty(y), d, \{d\} + p) \mid y \in X, d \in S^i_y, p \text{ is a walk of length } j \text{ of the form } p = y, \ldots, x\}$$

and the message sent by $\alpha(c)$ to $\alpha(x)$ at the end of round $2j + 1$ is precisely

$$\{(\delta_\infty(y), d, \{d\}+p) \mid y \in X, d \in S^i_y, p \text{ is a walk of length } j + 1 \text{ of the form } p = y, \ldots, c, x\}.$$

By Remark 4.10 only $2n2^{|D|} = T - 2n$ iterations are needed to identify all elements in S^i that are not S^i-supported. Hence, after exactly $T - 2n$ rounds every variable agent $\alpha(x)$ computes S^{i+1}_x by removing all the elements in S^i that are marked as "not S^i-supported". If $S^{i+1}_x = \emptyset$, then $\alpha(x)$ initiates a wave, which is propagated by all its neighbours, broadcasting that an inconsistency was detected. In this case, in at most $2n$ additional rounds all agents can correctly declare that I is unsatisfiable. Otherwise, a new iteration begins.

To sum up, the distributed consistency algorithm consists of $\mathcal{O}(n)$ iterations consisting, each, of $\mathcal{O}(n)$ rounds where the total running time for internal events at a given round is $\mathcal{O}(mns_{\max})$ and the maximum size of each message transmitted is $\mathcal{O}(ns_{\max})$. Together with the bounds given by Lemma 4.8 for the distributed colour refinement phase, this completes the proof of Theorem 4.7.

4.3 The Search Algorithm

We conclude by presenting the proof of Theorem 4.2. The hardness part follows immediately from Theorem 4.1 as the search problem is as difficult as the decision problem. For the positive result we shall present an adaptation of the algorithm solving the decision version. Let I be an instance of DCSP-Search(Γ) where Γ contains symmetric polymorphisms of all arities. In what follows we shall use intensively the fact that $\mathsf{Pol}(\Gamma)$ is closed under composition. Let $J \subseteq D$ be minimal with the property that $f(D) = J$ for some unary polymorphism f in $\mathsf{Pol}(\Gamma)$. It is fairly standard to show that for every $r \geq 0$ there is a r-ary symmetric operation g such that $g(x, \ldots, x) = x$ for every $x \in J$. Indeed, let f satisfy $f(D) = J$ and let g be any r-ary symmetric polymorphism in $\mathsf{Pol}(\Gamma)$. Then the unary operation h defined by $h(x) = f \circ g(x, \ldots, x)$ is a unary polymorphism of Γ. By the choice of f we have $h(D) \subseteq J$. We note that $h(J) = J$ since otherwise h^2 would contradict the minimality of f. Consequently, h^{-1} belongs to $\mathsf{Pol}(\Gamma)$ and, hence, the r-ary operation defined as $h^{-1} \circ f \circ g$ satisfies the claim. This implies that if we enlarge the constraint language by adding all singletons $\{d\}$, $d \in J$, the resulting constraint language, which we shall denote by Γ', still has symmetric polymorphisms of all arities. For convenience we also include D in Γ'.

The algorithm has two phases. In the first phase it runs the decision algorithm to determine whether the instance is satisfiable. As a byproduct, every variable agent $\alpha(x)$ has computed its iterated degree $\delta_\infty(x)$ and knows as well its rank in a prescribed ordering of all variable degrees $\delta^1_\infty, \dots, \delta^r_\infty$, $r \le n$. This (partial) order will be used to coordinate between the agents. An i-agent, $1 \le i \le r$ is any agent $\alpha(x)$ with $\delta_\infty(x) = \delta^i_\infty$. We also assume a fixed ordering on the elements in D. If the instance is unsatisfiable nothing else remains to be done so from now on we shall assume that the instance is satisfiable.

In the second phase the algorithm searches for a solution. Every variable agent $\alpha(x)$ maintains a set $F_x \subseteq D$ with the property that there is a solution ν that *falls within* F, i.e, such that $\nu(x) \in F_x$ for every $x \in X$. Initially every agent $\alpha(x)$ sets $F_x = D$ so it is only necessary to make sure that this condition is preserved during the execution of the algorithm. The second phase contains two nested loops. The outer loop has r iterations and the inner loop consists of at most $|D|$ iterations so that we shall use iteration (i, d) to indicate the run of the algorithm at the $i = 1, \dots, r$ iteration of the outer loop and at the iteration d of the inner loop.

At the beginning of iteration (i, d) every variable agent $\alpha(x)$ defines $S_x \subseteq D$ to be $S_x = \{d\}$ whenever $\alpha(x)$ is an i-agent and $S_x = F_x$ elsewhere. Then it runs the distributed consistency algorithm starting at S obtaining a fixed point S^∞. We note that since all initial sets S_x belong to Γ' and Γ' contains symmetric polymorphisms of all arities then the obtained fixed point S^∞ correctly determines whether there exists a solution ν that falls within S. Then every i-agent $\alpha(x)$ checks whether $S^\infty_x = \emptyset$. In case of positive answer nothing else is done and round (i, d) finishes. Otherwise, $\alpha(x)$ sets F_x to $\{d\}$ and starts a wave to indicate to all processes that the i^{th} iteration of the outer loop is finished and that the next iteration of the outer loop can start. When the r iterations of the outer loop have been completed the set system F contains only singletons. The assignment that sets every variable $x \in X$ to the only element in F_x is necessarily a solution. This concludes the proof of Theorem 4.2.

5 Conclusion

We analysed the complexity of the distributed constraint satisfaction problem on a synchronous, anonymous network parametrised by the constraint language. We showed that, depending on the polymorphisms of Γ, DCSP(Γ) is either solvable in polynomial time, or not solvable altogether. A number of natural open questions arise in this context. For instance, it is not clear whether asynchronous networks are strictly more powerful than their synchronous counterpart. Moreover, it would be interesting to explore the role of allowing agents to make random choices - provided this is not used to create and share unique IDs.

In the spirit of [13], one could consider characterizing the structural restrictions on tractable distributed CSPs, or in other words, determining which *classes of networks* are tractable in the DCSP framework, regardless of the constraint language. The starting point for this analysis could be the work on fibrations by Boldi et al. (see for example [7, 8]). In particular, we propose the question of establishing a connection between the universal fibration of a graph and its iterated degree sequence.

References

1 Dana Angluin. Local and global properties in networks of processors (extended abstract). In Raymond E. Miller, Seymour Ginsburg, Walter A. Burkhard, and Richard J. Lipton, editors, *Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA*, pages 82–93. ACM, 1980. doi:10.1145/800141.804655.

2 Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory Comput.*, 8(1):121–164, 2012. doi:10.4086/toc.2012.v008a006.

3 Pablo Barceló, Egor V. Kostylev, Mikaël Monet, Jorge Pérez, Juan L. Reutter, and Juan Pablo Silva. The logical expressiveness of graph neural networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=r1lZ7AEKvB.

4 Libor Barto, Andrei A. Krokhin, and Ross Willard. Polymorphisms, and how to use them. In Andrei A. Krokhin and Stanislav Zivný, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 1–44. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/DFU.Vol7.15301.1.

5 Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018. arXiv:1806.01261.

6 Ramon Bejar, Bhaskar Krishnamachari, Carla Gomes, and Bart Selman. Distributed constraint satisfaction in a wireless sensor tracking system. In *Workshop on Distributed Constraint Reasoning, International Joint Conference on Artificial Intelligence*, volume 4, 2001.

7 Paolo Boldi, Shella Shammah, Sebastiano Vigna, Bruno Codenotti, Peter Gemmell, and Janos Simon. Symmetry breaking in anonymous networks: Characterizations. In *Fourth Israel Symposium on Theory of Computing and Systems, ISTCS 1996, Jerusalem, Israel, June 10-12, 1996, Proceedings*, pages 16–26. IEEE Computer Society, 1996.

8 Paolo Boldi and Sebastiano Vigna. An effective characterization of computability in anonymous networks. In Jennifer L. Welch, editor, *Distributed Computing, 15th International Conference, DISC 2001, Lisbon, Portugal, October 3-5, 2001, Proceedings*, volume 2180 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2001. doi:10.1007/3-540-45414-4_3.

9 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.

10 Ken R. Duffy, Charles Bordenave, and Douglas J. Leith. Decentralized constraint satisfaction. *IEEE/ACM Trans. Netw.*, 21(4):1298–1308, 2013. doi:10.1109/TNET.2012.2222923.

11 Ferdinando Fioretto, Enrico Pontelli, and William Yeoh. Distributed constraint optimization problems and applications: A survey. *J. Artif. Intell. Res.*, 61:623–698, 2018. doi:10.1613/jair.5565.

12 Wan Fokkink. *Distributed algorithms: an intuitive approach.* MIT Press, 2013.

13 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1):1:1–1:24, 2007. doi:10.1145/1206035.1206036.

14 Martin Grohe. word2vec, node2vec, graph2vec, x2vec: Towards a theory of vector embeddings of structured data. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, pages 1–16. ACM, 2020. doi:10.1145/3375395.3387641.

15 Martin Grohe, Kristian Kersting, Martin Mladenov, and Pascal Schweitzer. Color refinement and its applications. *Van den Broeck, G.; Kersting, K.; Natarajan, S*, 2017.

16 Marcin Kozik. Solving CSPs using weak local consistency. *SIAM Journal on Computing, to appear*, 2020. URL: `https://marcinkozik.staff.tcs.uj.edu.pl/Solving.CSPs.using.weak.local.consistency.pdf`.

17 Andrei A. Krokhin and Stanislav Živný, editors. *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. URL: `http://www.dagstuhl.de/dagpub/978-3-95977-003-3`.

18 Gábor Kun, Ryan O'Donnell, Suguru Tamaki, Yuichi Yoshida, and Yuan Zhou. Linear programming, width-1 csps, and robust satisfaction. In Shafi Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 484–495. ACM, 2012. `doi:10.1145/2090236.2090274`.

19 Amnon Meisels. *Distributed Search by Constrained Agents - Algorithms, Performance, Communication*. Advanced Information and Knowledge Processing. Springer, 2008. `doi:10.1007/978-1-84800-040-7`.

20 Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 4602–4609. AAAI Press, 2019. `doi:10.1609/aaai.v33i01.33014602`.

21 Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006. URL: `http://www.sciencedirect.com/science/bookseries/15746526/2`.

22 Jan Toenshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. RUN-CSP: unsupervised learning of message passing networks for binary constraint satisfaction problems. *CoRR*, abs/1909.08387, 2019. `arXiv:1909.08387`.

23 Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: `https://openreview.net/forum?id=ryGs6iA5Km`.

24 Masafumi Yamashita and Tiko Kameda. Computing on an anonymous network. In Danny Dolev, editor, *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing, Toronto, Ontario, Canada, August 15-17, 1988*, pages 117–130. ACM, 1988. `doi:10.1145/62546.62568`.

25 Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *Proceedings of the 12th International Conference on Distributed Computing Systems, Yokohama, Japan, June 9-12, 1992*, pages 614–621. IEEE Computer Society, 1992. `doi:10.1109/ICDCS.1992.235101`.

26 Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Trans. Knowl. Data Eng.*, 10(5):673–685, 1998. `doi:10.1109/69.729707`.

27 Makoto Yokoo and Katsutoshi Hirayama. Algorithms for distributed constraint satisfaction: A review. *Auton. Agents Multi Agent Syst.*, 3(2):185–207, 2000. `doi:10.1023/A:1010078712316`.

28 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`.

Distance Computations in the Hybrid Network Model via Oracle Simulations

Keren Censor-Hillel
Technion – Israel Institute of Technology, Haifa, Israel

Dean Leitersdorf
Technion – Israel Institute of Technology, Haifa, Israel

Volodymyr Polosukhin
Technion – Israel Institute of Technology, Haifa, Israel

Abstract

The Hybrid network model was introduced in [Augustine et al., SODA '20] for laying down a theoretical foundation for networks which combine two possible modes of communication: One mode allows high-bandwidth communication with neighboring nodes, and the other allows low-bandwidth communication over few long-range connections at a time. This fundamentally abstracts networks such as hybrid data centers, and class-based software-defined networks.

Our technical contribution is a *density-aware* approach that allows us to simulate a set of *oracles* for an overlay skeleton graph over a Hybrid network.

As applications of our oracle simulations, with additional machinery that we provide, we derive fast algorithms for fundamental distance-related tasks. One of our core contributions is an algorithm in the Hybrid model for computing *exact* weighted shortest paths from $\tilde{O}(n^{1/3})$ sources which completes in $\tilde{O}(n^{1/3})$ rounds w.h.p. This improves, in both the runtime and the number of sources, upon the algorithm of [Kuhn and Schneider, PODC '20], which computes shortest paths from a single source in $\tilde{O}(n^{2/5})$ rounds w.h.p.

We additionally show a 2-approximation for weighted diameter and a $(1+\epsilon)$-approximation for unweighted diameter, both in $\tilde{O}(n^{1/3})$ rounds w.h.p., which is comparable to the $\tilde{\Omega}(n^{1/3})$ lower bound of [Kuhn and Schneider, PODC '20] for a $(2-\epsilon)$-approximation for weighted diameter and an exact unweighted diameter. We also provide fast distance *approximations* from multiple sources and fast approximations for eccentricities.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Distributed graph algorithms, Hybrid network model, Distance computations

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.21

Related Version *Full Version*: `https://arxiv.org/abs/2010.13831`

Funding This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 755839-ERC-BANDWIDTH.

Acknowledgements The authors would like to thank Michal Dory and Yuval Efron for various helpful conversations. We also thank Fabian Kuhn for sharing a preprint of [30] with us.

1 Introduction

The Hybrid model of computation was recently introduced by Augustine et al. [9], for abstracting networks which can utilize both high-bandwidth local communication links, as well as very few low-bandwidth global communication channels. This model abstracts fundamental systems, such as a combination of device-to-device communication with cellular networks (e.g. 5G) [7], wired data centers with wireless links (hybrid DCNs) [17, 26, 29, 41], and Class-Based Hybrid software-defined networks (SDNs) [40].

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 21; pp. 21:1–21:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The pioneering works of [9, 20, 30] provide fast algorithms for various distance-related tasks in the Hybrid model. At the heart of many of these algorithms lies a framework for using skeleton overlay graphs for computation and approximation of distances, as well as for fast communication.

In this paper, we define and show how to efficiently simulate *oracles* over skeleton graphs in the Hybrid model. Using additional machinery that we provide, the implications of our simulations are faster algorithms for distance computations in the Hybrid model. Our oracle models could also be of independent interest, presenting a generic approach which can potentially be applied elsewhere.

1.1 Our Contributions

The Hybrid model, which we consider in this paper, abstracts a synchronous network of nodes, where in each round, every node can send and receive arbitrarily many messages of $O(\log n)$ bits to/from each of its neighbors (over *local edges*) and an additional $O(\log n)$ messages, in total, to/from any other nodes in the network (over *global edges*). The high bandwidth permissible over the local edges is aligned with previous research in the Hybrid model as well as with the extensively studied LOCAL distributed model.

The main idea which our results hinge upon is exploiting an inherent *asymmetry* in the Hybrid model which we observe. This asymmetry allows nodes with *dense* neighborhoods to effectively receive significantly more information. To see this, note that every node can use the global edges of the Hybrid model to send and receive a limited number of messages every round. However, since in the next round a node can communicate with its neighborhood in the graph using local edges and share with them the information which it received, this implies that a node is able to learn much more information from the entire graph if it is in a more dense neighborhood. Thus, *density-aware* algorithms are inherently useful in the Hybrid model.

We use this asymmetry to simulate Oracle and Tiered Oracles models. To capture what an oracle can do, we introduce the Oracle and Tiered Oracles models. Roughly speaking, in the Oracle model there is a node ℓ, the *oracle*, which can receive $\deg(v)$ messages from each node v, within *a single round.* In particular, this implies that the oracle can learn the entire communication graph.

We cannot afford to directly simulate the Oracle model as it requires too much communication in the Hybrid model. Instead, we simulate the Oracle model over a *skeleton graph.* Roughly speaking, given an input graph G, a skeleton graph is a subset of the nodes of G, connected by virtual edges which represent paths in G. Skeleton graphs are a common tool for distance computations in various models [9, 30, 33, 37, 39], and it has been shown in [9, 30] that some distances on the skeleton graph can be efficiently extended to distances on the entire graph in the Hybrid model.

As a warm-up, we show that a single round of the Oracle model over certain skeleton graphs can be simulated in $\tilde{O}(n^{1/3})$ rounds, w.h.p.[1], in the Hybrid model. Combining this with a simple, constant-round algorithm for exact weighted *single source shortest paths (SSSP)* which we show in the Oracle model, gives the following theorem.

▶ **Theorem 1** (Exact SSSP). *Given a weighted graph $G = (V, E)$, there is an algorithm in the* Hybrid *model that computes an exact weighted SSSP in $\tilde{O}(n^{1/3})$ rounds w.h.p.*

[1] As common, w.h.p. indicates a probability that is at least $1 - n^{-c}$, for some constant $c > 1$.

This result should be compared with the previous state-of-the-art algorithms for exact weighted SSSP in $\tilde{O}(n^{2/5})$ rounds [30], and in $\tilde{O}(\sqrt{SPD})$ rounds [9], where SPD is the length of the shortest path diameter. Further, it improves upon the $\tilde{O}(n^{1/3}/\epsilon^6)$ round algorithm for a $(1+\epsilon)$-approximation of weighted SSSP [9], in both the runtime and in being exact. We stress that this is a warm up, and later on we extend this result to shortest path distances from $O(n^{1/3})$ sources, instead of a single source, in the same round complexity of $\tilde{O}(n^{1/3})$.

It is well known that one can approximate the diameter using a solution to SSSP, and so as a byproduct we get the following result.

▶ **Corollary 2** (2-Approx. Weighted Diameter). *There is an algorithm in the* Hybrid *model that computes a 2-approximation of weighted diameter in $\tilde{O}(n^{1/3})$ rounds w.h.p.*

Notably, $\tilde{\Omega}(n^{1/3})$ rounds are necessary for a $(2-\epsilon)$-approximation for weighted diameter [30]. Our algorithm in Corollary 2 thus raises the interesting open question of whether one can go below this complexity for a 2-approximation.

While efficiently simulating an oracle is powerful, it still does not exploit the full capacity of the Hybrid model. This observation brings us to enhance the Oracle model and introduce the Tiered Oracles model, which consists of multiple oracles with varying abilities. In a nutshell, in the Tiered Oracles model, in each round every node v can send (the same) $\deg(v)$ messages to all nodes u with $\deg(u) \geq \deg(v)/2$. This basically means that nodes are bucketed according to degrees and each node is an oracle for all nodes in buckets below it. One can notice that the node with the highest degree in the graph is equivalent to the oracle in the Oracle model, but here, the other nodes in the graph also have some *partial* oracle capabilities.

We show how to simulate the Tiered Oracles model over skeleton graphs in the Hybrid model within $\tilde{O}(n^{1/3})$ rounds. Subsequently, we present an algorithm which solves all pairs shortest paths (APSP) using one round of the Tiered Oracles model and $O(\log n)$ rounds of the Congested Clique model[2]. We then utilize our Tiered Oracles model simulation, along with a previously known simulation of the Congested Clique model from [30], to simulate the APSP algorithm over skeleton graphs in the Hybrid model. Our efficient computation of APSP over a skeleton graph in the Hybrid model then leads to computing multi-source shortest paths from *random* sources in the Hybrid model.

Shortest paths from random sources is a crucial stepping stone for our later results. We show that computing shortest path distances from random sources to the entire graph, allows us to subsequently obtain fast algorithms for other distance problems. We call the problem of computing distances from sources sampled with probability n^{x-1} i.i.d n^x-RSSP.

▶ **Theorem 3** (n^x-RSSP). *Given a graph $G=(V,E)$, $0<x<1$, and a set of nodes M sampled independently with probability n^{x-1}, there is an algorithm in the* Hybrid *model that ensures that every $v \in V$ knows the exact, weighted distance from itself to every node in M within $\tilde{O}(n^{1/3}+n^{2x-1})$ rounds w.h.p.*

We complement Theorem 3 with a lower bound, following the lines of [9, 30], for approximating distances from many random sources, to any reasonable approximation factor, which tightly matches the upper bound when $x = 2/3$.

▶ **Theorem 4** (Lower Bound Exact Shortest Paths, Sources Sampled i.i.d.). *Let $p = \Omega(\log n/n)$ and $\alpha < \sqrt{n/p} \cdot \log(n)/2$. Any α-approximate unweighted algorithm from random sources sampled independently with probability p in the* Hybrid *network model takes $\Omega\left(\sqrt{p \cdot n}/\log n\right)$ rounds w.h.p.*

[2] The Congested Clique is a synchronous distributed model where every two nodes in the graph can exchange messages of $O(\log n)$ bits in every round.

We leverage our *near-optimal* (tight up to polylogarithmic factors) algorithm for shortest paths from a set M of $\tilde{O}(n^{2/3})$ random sources in order to obtain exact weighted shortest paths from any *given* set U of $O(n^{1/3})$ sources. We achieve this by adapting the behavior of the given fixed source nodes to the density of their neighborhoods, as follows. A source node $s \in U$ in a sparse neighborhood broadcasts the distances to all the random source nodes from M it sees in its neighborhood. A source node $s \in U$ in a dense neighborhood takes control of one of the random sources in M in its neighborhood and uses it as a proxy in order to communicate enough information to all the other nodes in the graph so that they could determine their distances from s. We remark that this proxy approach is a *key insight* which we later encapsulate as a general tool in the Hybrid model and may potentially be of independent interest. Our approach gives the following theorem.

▶ **Theorem 5** (Exact $n^{1/3}$ Sources Shortest Paths). *Given a weighted graph $G = (V, E)$, and a set of sources U, such that $|U| = O(n^{1/3})$, there exists an algorithm, at the end of which each $v \in V$ knows its distance from every $s \in U$, which runs in $\tilde{O}(n^{1/3})$ rounds w.h.p.*

Theorem 5 raises an interesting open question of whether the complexity of SSSP in the Hybrid model is below that of computing shortest paths from $\tilde{O}(n^{1/3})$ sources.

We also exploit our aforementioned solution for computing APSP on the skeleton graph to obtain approximate distances from a larger set of given sources (n^x-SSP), as follows.

▶ **Theorem 6** (Approximate Multiple Source Shortest Paths). *Given a graph $G = (V, E)$, a set of sources U, where $|U| = \tilde{\Theta}(n^y)$ for some constant $0 < y < 1$, and a value $0 < \epsilon$, there is an algorithm in the* Hybrid *model which ensures that every node $v \in V$ knows an approximation to its distance from every $s \in U$, where the approximation factor is $(1 + \epsilon)$ if G is unweighted and 3 if G is weighted. The complexity of the algorithm is $\tilde{O}(n^{1/3}/\epsilon + n^{y/2})$ rounds, w.h.p.*

This result improves both in round complexity and approximation factors upon the previous results in [30]. The reason for this is that we compute APSP over skeleton graphs using the efficient, exact algorithm from the Tiered Oracles oracle model, while [30] simulate the slower, approximate algorithms of [12, 14] in the Congested Clique model. Particularly, this result is tight up to polylogarithmic factors for $y \geq 2/3$ due to a lower bound of [30].

We can also approximate unweighted eccentricities by a combination of computing shortest path distances from $n^{2/3}$ random sources and performing local explorations using the local edges of the model. For approximating weighted eccentricities, this is insufficient, and here our approach is to additionally broadcast required information from each random source node regarding its $\tilde{O}(n^{1/3})$-hop neighborhood in the graph. We obtain the following result.

▶ **Theorem 7** (Approx. Eccentricities). *Given a graph $G = (V, E)$, there is an algorithm in the* Hybrid *model that computes a $(1 + \epsilon)$-approximation of unweighted and 3-approximation of weighted eccentricities in $\tilde{O}(n^{1/3}/\epsilon)$ rounds, w.h.p.*

Finally, the unweighted eccentricities approximation directly implies a $(1 + \epsilon)$ approximation for unweighted diameter. This should be compared with the lower bound of $\tilde{\Omega}(n^{1/3})$ rounds for exact unweighted diameter due to [30].

▶ **Corollary 8** ($(1 + \epsilon)$-Approx. Unweighted Diameter). *Let $G = (V, E)$ be an unweighted graph, and let $\epsilon > 0$. There exists an algorithm in the* Hybrid *model which computes a $(1 + \epsilon)$-approximation of the diameter in $\tilde{O}(n^{1/3}/\epsilon)$ rounds, w.h.p.*

We refer the reader to Table 1, for a visual summary of our end results with comparison to related work.

Roadmap. The remainder of the current section is dedicated towards surveying related work. In Section 2, we provide all formal definitions. Next, in Section 3 we formally define the Oracle and Tiered Oracles models, and show how to simulate them over skeleton graphs in the Hybrid model. Finally, Section 4, gives our algorithms for distance problems in the oracle models and, using our simulations, also in the Hybrid model. Some additional results are deferred to the the full version of the paper [15]. There we show the approximation for shortest path from n^x given sources, eccentricities, diameter and lower bound for shortest paths from sources sampled i.i.d.

Table 1 Comparison of our results. SPD is the length of the shortest path diameter. The results for n^x-RSSP, and weighed diameter approximation upper bounds from previous works are implicit in [9, 30]. Our upper bound for $n^{2/3}$-RSSP is tight up to poly-logarithmic factors due to our lower bound. Our approximations for n^x-SSP are also tight up to poly-logarithmic factors for $x \geq 2/3$, due to [30].

Problem	Variant	Approximation	This work	Previous works
SSSP	weighted	exact	$\tilde{O}(n^{1/3})$	$\tilde{O}(n^{2/5})$ [30], $\tilde{O}(\sqrt{SPD})$ [9]
	weighted	$1+\epsilon$		$\tilde{O}(n^{1/3} \cdot \epsilon^{-6})$ [9]
	weighted	$(1/\epsilon)^{O(1/\epsilon)}$		n^{ϵ} [9]
n^x-RSSP	unweighted	$\tilde{O}(n^{1-x/2})$	$\tilde{\Omega}(n^{x/2})$	
	weighted	exact	$\tilde{O}(n^{1/3}+n^{2x-1})$	
	weighted	$2+\epsilon$		$\tilde{O}(n^{1/3}+n^{2x-1})$ [30]
$n^{1/3}$-SSP	unweighted	$1+\epsilon$		$\tilde{O}(n^{1/3}/\epsilon)$ [30]
	weighted	exact	$\tilde{O}(n^{1/3})$	
	weighted	$3+\epsilon$		$\tilde{O}(n^{1/3}/\epsilon)$ [30]
n^x-SSP	unweighted	$\tilde{O}(n^{1-x/2})$		$\tilde{\Omega}(n^{x/2})$ [30]
	unweighted	$1+\epsilon$	$\tilde{O}(n^{1/3}/\epsilon+n^{x/2})$	
	unweighted	$2+\epsilon$		$\tilde{O}(n^{1/3}/\epsilon+n^{x/2})$ [30]
	weighted	3	$\tilde{O}(n^{1/3}+n^{x/2})$	
	weighted	$3+\epsilon$		$\tilde{O}(n^{0.397}+n^{x/2})$ [30]
	weighted	$7+\epsilon$		$\tilde{O}(n^{1/3}/\epsilon+n^{x/2})$ [30]
eccentricities	unweighted	$1+\epsilon$	$\tilde{O}(n^{1/3}/\epsilon)$	
	weighted	3	$\tilde{O}(n^{1/3})$	
diameter	unweighted	exact		$\tilde{\Omega}(n^{1/3})$ [30]
	unweighted	$1+\epsilon$	$\tilde{O}(n^{1/3}/\epsilon)$	$\tilde{O}(n^{0.397}/\epsilon)$ [30]
	unweighted	$3/2+\epsilon$		$\tilde{O}(n^{1/3}/\epsilon)$ [30]
	weighted	$2-\epsilon$		$\tilde{\Omega}(n^{1/3})$ [30]
	weighted	2	$\tilde{O}(n^{1/3})$	$\tilde{O}(n^{2/5})$ [30]
	weighted	$2+\epsilon$		$\tilde{O}(n^{1/3} \cdot \epsilon^{-6})$ [9]
	weighted	$2 \cdot (1/\epsilon)^{O(1/\epsilon)}$		n^{ϵ} [9]

1.2 Related Work

Hybrid Models. The Hybrid network model was studied in [9, 20, 30]. In [20], distance results are obtained in one of the harsher variants of the model, where the local edges are restricted to have $\log n$ bandwidth. However, these apply only to extremely sparse graphs of at most $n + O(n^{1/3})$ edges and cactus graphs. In [2, 25], slightly different models of hybrid nature are studied.

Augustine et al. [8] proposed the Node-Capacitated Clique model, which is similar to the Congested Clique model, but each node has $\log n$ bandwidth. This model is also a special case of the generalised Hybrid model [9] without local edges. This allows one to use the results from the Node-Capacitated Clique model in the Hybrid model without modifications.

Distributed Distance Computations. Distance related problems have been extensively studied in many distributed models. For example, in the CONGEST model, there is a long line of research on APSP [3–5, 11, 19, 32, 34, 36] which culminated in tight, up to polylogarithmic factors, $\tilde{O}(n)$ round exact weighted APSP randomized algorithm of Bernstein and Nanongkai [11] and a $\tilde{O}(n^{4/3})$ round deterministic algorithm of Agarwal and Ramachandran [4]. [34, 36] develop an $\tilde{O}(n)$ round algorithm, optimal up to polylogarithmic factors, for unweighted APSP. The study of approximate SSSP algorithms was the focus of many recent paper [10, 27, 38] and lately Becker et al. [10] showed the solution which is close to the lower bound of Das Sarma et al. [38]. In case of exact SSSP, after recent works [19, 21, 24], there still is a gap between upper and lower bounds. The diameter and eccentricities problems are studied in the CONGEST model in [1, 6, 22, 36].

In the Congested Clique model, k-SSP, APSP and diameter are extensively studied in [14, 16, 23, 35] and approximate versions of the k-SSP and APSP problem are solved in polylogarithmic [13] and even polyloglogaritmic [18] time. In the more restricted Broadcast Congested Clique model, in which each message a node sends in a round is the same for all recipients, distance computations are researched by [10, 28].

2 Preliminaries

We provide here some definitions and claims that are critical for reading the main part of the paper. Full version of the paper [15] contains additional definitions and basic claims. We use the following variant of the Hybrid model, introduced in [9].

▶ **Definition 9** (Hybrid Model). *In the* Hybrid *model, a synchronous network of n nodes with identifiers in $[n]$, is given by a graph $G = (V, E)$. In each round, every node can send and receive λ messages of $O(\log n)$ bits to/from each of its neighbors (over* local edges*) and an additional γ messages in total to/from any other nodes in the network (over* global edges*). If in some round more than γ messages are sent via global edges to/from a node, only γ messages selected adversarially are delivered.*

We follow the previous work of [9, 30] and consider $\lambda = \infty, \gamma = O(\log n)$. Notice that the Hybrid model can also capture the classic LOCAL[3] model, with $\lambda = \infty, \gamma = 0$, the classic CONGEST model, with $\lambda = O(1), \gamma = 0$, the Congested Clique model, with $\lambda = O(1), \gamma = 0$ and G being a clique, the Congested Clique + Lenzen's Routing with $\lambda = 0, \gamma = n$ and the Node-Capacitated Clique model [8], with $\lambda = 0, \gamma = O(\log(n))$.

Many of our results hold for weighted graphs $G = (V, E, w)$. We assume an edge weight is given by a function $w \colon E \mapsto \{1, 2, \ldots, W\}$ for a W which is polynomial in n. When we *send* an edge as part of a message in any algorithm, we assume it is sent along with its weight.

[3] The LOCAL and CONGEST models are synchronous distributed models where every two neighbors in the graph can exchange messages of unlimited size or of $O(\log n)$ bits, respectively, in each round.

2.1 Notations and Problem Definitions

We use the following definitions related to graphs. Given a graph $G = (V, E)$ and a pair of nodes $u, v \in V$, we denote by $hop(u, v)$ the hop distance between u and v, by $N^h_G(v)$ the h-hop neighborhood of v, by $d^h_G(u, v)$ the weight of the lightest path between u and v of at most h-hops, and if there is no path of at most h-hops then $d^h_G(u, v) = \infty$. In the special case of $h = 1$, we denote by $N_G(v)$ the neighbors of v and in the special case of $h = \infty$, we denote by $d_G(u, v)$ the weight of the lightest path between u and v. We also denote by $\deg_G(v)$ the degree of v in G. Whenever it is clear from the context we drop the subscript of G and just write N, N^h, d, d^h or $\deg(v)$.

We define the following problems in the Hybrid model.

▶ **Definition 10** (k-Source Shortest Paths (k-SSP)). *Given a graph $G = (V, E)$, and a set $S \subseteq V$ of k sources. Every $u \in V$ is required to learn the distance $d_G(u, s)$ for each $s \in S$. The case where $k = 1$, is called the* single source shortest paths *problem (SSSP).*

▶ **Definition 11** (n^x-Random Sources Shortest Path (n^x-RSSP)). *Given a graph $G = (V, E)$, and a set $M \subseteq V$ of sources, such that each $v \in V$ is sampled independently with probability n^{x-1} to be in M. Every $u \in V$ is required to learn the distance $d_G(u, s)$ for each $s \in M$.*

In the approximate versions of these problems, each $u \in V$ is required to learn an (α, β)-approximate distance $\widetilde{d}(u, v)$ which satisfies $d(u, v) \leq \widetilde{d}(u, v) \leq \alpha \cdot d(u, v) + \beta$, and in case $\beta = 0$, $\widetilde{d}(u, v)$ is called an α-approximate distance.

▶ **Definition 12** (Eccentricity and diameter). *Given a graph $G = (V, E)$ and node $v \in V$, the* eccentricity *of v is the farthest shortest path distance from v, i.e., $ecc(v) = \max_{u \in V} d(v, u)$ and the diameter $D = \max_{v \in V} \{ ecc(v) \}$ is the maximum eccentricity. An α-approximation of all eccentricities is a function $\widetilde{ecc}(v)$ which satisfies $ecc(v)/\alpha \leq \widetilde{ecc}(v) \leq ecc(v)$ for all nodes v. An α-approximation of the diameter is a value $\widetilde{D}$ which satisfies $D/\alpha \leq \widetilde{D} \leq D$.*

2.2 Skeleton Graphs

In a nutshell, given a graph $G = (V, E)$, a skeleton graph $S_x = (M, E_S)$, for some constant $0 < x < 1$, is generated by letting every node in V independently join M with probability n^{x-1}. Two nodes in M have an edge in E_S if there exists a path between them in G of at most $h = \tilde{O}(n^{1-x})$ hops. This graph w.h.p. satisfies many useful properties in terms of distance computation, which for simplicity of presentation we add to its definition, provided below. A crucial property is that for any two nodes, if the shortest path between them in G has more than h hops, then there exists a shortest path between them in G on which every roughly h nodes there is a node from M (all such skeleton properties hold w.h.p.).

▶ **Definition 13** (Skeleton Graph, Combined Definition of [9, 30]). *Given a graph $G = (V, E)$ and a value $0 < x < 1$, a graph $S_x = (M, E_S)$ is called a skeleton graph in G, if all of the following hold.*

1. *Each $v \in V$ is included to M independently with probability n^{x-1}.*
2. *$\{v, u\} \in E_S$ if and only if there is a path of at most $h = \tilde{\Theta}(n^{1-x})$ edges between v, u in G.*
3. *Every node $v \in M$ knows all its incident edges in E_S.*
4. *S_x is connected.*
5. *For any two nodes $v, v' \in M$, $d_S(v, v') = d_G(v, v')$.*
6. *For any two nodes $u, v \in V$ with $hop(u, v) \geq h$, there is at least one shortest path P from u to v in G, such that any sub-path Q of P with at least h nodes contains a node $w \in M$.*
7. *$|M| = \tilde{O}(n^x)$.*

The following claim summarizes what is proven in [9] regarding the construction of a skeleton graph from a set of random marked nodes, w.h.p.

▷ Claim 14 (Skeleton from Random Nodes). Given a graph $G = (V, E)$, a value $0 < x < 1$, and a set of nodes M marked independently with probability n^{x-1}, there is an algorithm in the Hybrid model which constructs a skeleton graph $S_x = (M, E_S)$ in $\tilde{O}(n^{1-x})$ rounds w.h.p. If also given a single node $s \in V$, it is possible to construct $S_x = (M \cup \{s\}, E_S)$, without damaging the properties of S_x.

We extract the following basic claim, used in the proof of [9, Theorem 2.7] for a $(1+\epsilon)$-approximation for SSSP, and slightly extend it to use for multiple source problem and arbitrary approximation factors. It states that given a skeleton graph and a set of sources, if every skeleton node knows any approximation to its distance from every source, then it is possible to efficiently reach a state where every node in the graph knows the approximation for its own distance from any of the sources. The idea is that each node locally explores its $\tilde{O}(n^{1-x})$ neighborhood and identifies for each source the best skeleton node in its neighborhood to go through.

▷ Claim 15 (Extend Distances). [9, Theorem 2.7] Let $G = (V, E)$, let $S_x = (M, E_S)$ be a skeleton graph, and let $V' \subseteq V$ be the set of source nodes. If for each source node $s \in V'$, each skeleton node $v \in M$ knows the (α, β)-approximate distance $\tilde{d}(v, s)$ such that $d(v, s) \leq \tilde{d}(v, s) \leq \alpha d(v, s) + \beta$, then each node $u \in V$ can compute for all source nodes $s \in V'$, a value $\tilde{d}(u, s)$ such that $d(u, s) \leq \tilde{d}(u, s) \leq \alpha d(u, s) + \beta$ in $\tilde{O}(n^{1-x})$ rounds.

3 Oracles in the Hybrid model

This section is split into three parts. Initially, as preliminaries, we show simulations of the LOCAL and Congested Clique models in the Hybrid model, citing [30] for the Congested Clique simulation. Then, we devote a section to each of the two new oracle models in order to introduce them and present their simulations in the Hybrid model.

3.1 Model Simulation Preliminaries

We will use simulations of the LOCAL and Congested Clique models as follows.

▶ **Lemma 16** (LOCAL Simulation). *Given a graph $G = (V, E)$, and a skeleton graph $S_x = (M, E_S)$, it is possible to simulate one round of the* LOCAL *model over S_x within $\tilde{O}(n^{1-x})$ rounds in G in the* Hybrid *model. That is, within $\tilde{O}(n^{1-x})$ rounds in G in the* Hybrid *model, any two adjacent nodes in S_x can communicate any amount of data between each other.*

The proof follows trivially due to the definition of the Hybrid model and Property 2 in the definition of a skeleton graph S_x, since in S_x two skeleton nodes are connected if they are within $\tilde{\Theta}(n^{1-x})$ hops in the original graph G. Thus, one round of the LOCAL model over S_x is obtained in the Hybrid network in $\tilde{O}(n^{1-x})$ rounds, by having neighboring skeleton nodes communicate through the local edges.

▶ **Lemma 17** (Congested Clique Simulation). *[30, Corollary 4.1.] Given a graph $G = (V, E)$, and a skeleton graph $S_x = (M, E_S)$, for some constant $0 < x < 1$, it is possible to simulate one round of the* Congested Clique *model over S_x in $\tilde{O}(n^{2x-1} + n^{\frac{x}{2}})$ rounds of the* Hybrid *model on G, w.h.p. That is, within $\tilde{O}(n^{2x-1} + n^{\frac{x}{2}})$ rounds of the* Hybrid *model on G, w.h.p., every node $v \in M$ can, for each node $u \in M$, each send a unique $O(\log n)$ bit message to u.*

3.2 Simulating the Oracle Model

Here, we define the Oracle model and then show how to efficiently simulate it over a skeleton graph in the Hybrid model.

▶ **Definition 18** (Oracle Model)**.** *In the* Oracle *model over a network G, there exists one* oracle *node ℓ, which in every round can send to and receive from every node v a number of $O(\log n)$-bit messages that is equal to the degree of v in G.*

▶ **Theorem 19** (Oracle Simulation)**.** *Given a graph $G = (V, E)$, for every constant $0 < x < 1$, there is an algorithm which simulates one round of the* Oracle *model, on a skeleton graph $S_x = (M, E_S)$, in $\tilde{O}(n^{1-x} + n^{2x-1})$ rounds of the* Hybrid *model on G, w.h.p.*

Proof. We prove the claim by showing how to simulate a round of the Oracle model in $O(1)$ rounds of the Congested Clique model and 1 round of the LOCAL model. Then, invoking the simulations of Lemmas 16 and 17, gives the desired round complexity in the Hybrid model.

We show how to send messages to the oracle, and the receiving part is symmetric. The pseudocode is given by Algorithm 1. First, each $v \in M$ broadcasts its degree in S_x using one round of the Congested Clique model (Line 1) and selects as an oracle ℓ the node with largest degree in S_x, breaking ties by identifier (Line 2). Then, the identifiers of the neighbors of ℓ are broadcast using one round of the Congested Clique model (Line 3). The actual messages are sent to these neighbors instead of to ℓ itself (Line 4) and ℓ learns all these messages in 1 round of the LOCAL model in Line 5.

Clearly, all the nodes select the same oracle ℓ (Line 2). Due to the definition of the Oracle model, each node $v \in M$ has $\deg_{S_x}(v)$ messages to send, and since $\deg_{S_x}(\ell) \geq \deg_{S_x}(v)$, there are enough neighbors of ℓ to receive one message from v per neighbor, which is why Line 4 can work. ◀

Algorithm 1 Simulating the Oracle model in the Congested Clique with LOCAL.

```
1 Congested Clique model: $v \in M$ broadcasts $\deg_{S_x}(v)$
2 Select an oracle $\ell \leftarrow \arg\max_{v \in M} \{ (\deg_{S_x}(v), v) \}$
3 Congested Clique model: $v \in M$ broadcasts if it is a neighbor of $\ell$
4 Congested Clique model: $v \in M$ sends $i$-th message to $i$-th neighbor of $\ell$ for each $i$
5 LOCAL model: $\ell$ collects the messages from its neighbors
```

3.3 Simulating the Tiered Oracles Model

We further enhance our Oracle model and define the Tiered Oracles model, where, roughly speaking, all nodes in parallel can learn all the edges adjacent to nodes with degrees in lower degree buckets. To simulate the stronger Tiered Oracles model over a skeleton graph in the Hybrid model, we need additional insights. Here, we use the fact that when we scatter messages independently at random, denser neighborhoods are more likely to receive a given message than sparse neighborhoods. In other words, while for simulating the Oracle model we used the LOCAL round only to concentrate information in a single node ℓ, here we exploit the information that *each* node can gather from its neighborhood.

▶ **Definition 20** (Tiered Oracles Model)**.** *In the* Tiered Oracles *model over a network G, in every round, suppose each node v has a set of $O(\log n)$-bit messages M_v of size $|M_v| = \deg(v)$, then each node u can receive all messages in M_v for every v such that $\deg(u) \geq \deg(v)/2$.*

To simulate the Tiered Oracles model, we first prove the following model-independent tool.

▶ **Lemma 21** (Sampled neighbors)**.** *Given is a graph $G = (V, E)$. For a value $c \leq n$, there is a value $x = \tilde{O}(n/c)$ such that the following holds w.h.p.: Let $V' \subseteq V$ be a subset of $|V'| = x$ nodes sampled uniformly at random from M. Then each node $u \in V$ with $\deg(u) \geq c$ has a neighbor in V'.*

Proof. For some node $u \in V$, the probability of not having a neighbor sampled to the set V' is $(1 - \deg(u)/n)^x \leq e^{-x \cdot \det(u)/n} \leq e^{x \cdot c/n}$. Thus, there exists $x = \tilde{O}(n/c)$ such that node u has a neighbor in the set V', w.h.p. ◀

Finally, we show how to simulate the Tiered Oracles model over the skeleton graph in the Hybrid model.

▶ **Theorem 22** (Tiered Oracles Simulation)**.** *Given a graph $G = (V, E)$, for every constant $0 < x < 1$, there is an algorithm which simulates one round of the* Tiered Oracles *model, on a skeleton graph $S_x = (M, E_S)$, in $\tilde{O}(n^{1-x} + n^{2x-1})$ rounds of the* Hybrid *model on G, w.h.p.*

Proof. We prove the claim by reducing one round of the Tiered Oracles model to $\tilde{O}(1)$ rounds of the Congested Clique model followed by a round of the LOCAL model on the skeleton graph S_x. By Lemmas 16 and 17, we obtain that the resulting round complexity is $\tilde{O}(n^{1-x} + n^{2x-1})$.

For each $v \in M$, let M_v be the set of messages, of size $|M_v| = \deg_{S_x}(v)$, which v desires to broadcast. For each message in M_v node $v \in M$ samples uniformly at random $x = \tilde{O}(2 \cdot n/\deg_{S_x}(v))$ nodes of M and sends the message to those nodes. As each node sends and receives $\tilde{O}(|M|)$ messages this can be done using with the well known routing theorem of Lenzen [31, Theorem 3.7] by simulating $\tilde{O}(1)$ rounds of the Congested Clique model. Alternatively, this can be done in the same round complexity by applying the algorithm for *token routing* [30, Theorem 2.2]. Afterwards, we simulate one round of the LOCAL model over S_x for each node to learn tokens received by its neighbors in S. Due to Lemma 21 (*Sampled neighbors*), each node $u \in V$ learns messages from each v such that $\deg_{S_x}(u) \geq \deg_{S_x}(v)/2$ w.h.p. ◀

4 Shortest Paths Algorithms

4.1 Warm-Up: Exact SSSP

As a warm-up, we show how to compute exact SSSP in the Oracle model, and then we simulate this on a skeleton graph in the Hybrid model in order to get exact SSSP in the Hybrid model within $\tilde{O}(n^{1/3})$ rounds. We note that later, in Section 4.3, we obtain this complexity for exact distances from a much larger set, of $O(n^{1/3})$ sources.

▶ **Lemma 23** (Exact SSSP in the Oracle Model)**.** *There is a* deterministic *algorithm in the* Oracle *model that given a weighted graph $G = (V, E)$ and source $s \in V$ solves exact SSSP in $O(1)$ rounds.*

Proof. Let $s \in V$ be the source node. We solve the problem in two communication rounds. In the first round, oracle ℓ learns all of E by receiving from each node v its adjacent edges. Afterwards, oracle ℓ, given all the edges in the graph G, locally computes the distance from s to every other node. In the second round, oracle ℓ sends for each $v \in V$ the value $d(s, v)$. It is clear that the algorithm computes SSSP from $s \in V$, and that it takes two rounds in the Oracle model. ◀

▶ **Theorem 1** (Exact SSSP). *Given a weighted graph $G = (V, E)$, there is an algorithm in the* Hybrid *model that computes an exact weighted SSSP in $\tilde{O}(n^{1/3})$ rounds w.h.p.*

Proof. Let s be the source node, and let $x = 2/3$. We start by constructing a skeleton graph $S_x = (M, E_S)$, by sampling nodes with probability $n^{-1/3}$ and using Claim 14 (*Skeleton from Random Nodes*). Then, we simulate the algorithm given in Lemma 23 in the Oracle model, which computes the distance $d_S(s, v)$ from s to each node $v \in M$. By Property 5 of the skeleton graph, for every $v \in M$, it holds that $d_S(s, v) = d_G(s, v)$. To extend this and compute the distance from s for each node $v \in V$, we apply Claim 15 (*Extend Distances*).

Constructing the skeleton graph takes $O(h) = \tilde{O}(n^{1/3})$ rounds w.h.p., by Claim 14 (*Skeleton from Random Nodes*). Simulating the algorithm from Lemma 23 completes in $\tilde{O}(n^{1/3})$ rounds w.h.p. by Theorem 19 (Oracle *Simulation*). Applying Claim 15 (*Extend Distances*) takes $\tilde{O}(n^{1/3})$ rounds. Therefore, overall, the execution of the algorithm completes in $\tilde{O}(n^{1/3})$ rounds w.h.p. ◀

4.2 Exact n^x-RSSP

Recall that in Definition 11 (*n^x-Random Sources Shortest Path (n^x-RSSP)*), we are given set of roughly n^x sources sampled independently with probability n^{x-1}, and we need for each node to compute its distance to each source. We do so by constructing a skeleton graph S_x from the random sources. We show that using one round of the Tiered Oracles model, and $O(\log n)$ rounds of the Congested Clique model, one can solve APSP over S_x. To do so, we split the nodes of the graph into $\lceil \log n \rceil$ tiers by degree and compute APSP by proceeding tier after tier and computing distances from current tier to all the tiers below.

▶ **Lemma 24** (APSP in Congested Clique with Tiered Oracles). *There is a* deterministic *algorithm which, given a weighted graph $G = (V, E)$, solves exact APSP on G using $O(\log |V|)$ rounds of the* Congested Clique *model and one round of the* Tiered Oracles *model.*

Proof. The pseudocode for the algorithm appears in Algorithm 2. We partition the nodes V by their degrees into $\lceil \log |V| \rceil$ tiers, $T_j = \{ v \in V : 2^j \leq \deg(v) < 2^{j+1} \}$ for $0 \leq j < \lceil \log |V| \rceil$. Denote by $T_{>i} = \bigcup_{k>i} T_k$ the nodes in all tiers $k > i$ and by $T_{\leq i} = \bigcup_{k \leq i} T_k$ the nodes in all tiers $k \leq i$. Similarly, define $T_{\geq i}$ and $T_{<i}$. Denote by $d_{\leq i}(u, v)$ the weight of the shortest path between u and v that uses only edges adjacent to at least one node in $T_{\leq i}$.

Algorithm 2 **Exact-APSP**: Computes exact APSP using the Congested Clique and Tiered Oracles models.

```
1 Tiered Oracles model: each $v \in T_i$ broadcasts to $u \in T_{\geq i}$ its incident edges
2 for $i = \lceil \log |V| \rceil - 1$ downto 0 do
3     For each node $u \in T_{\leq i}$, each node $v \in T_i$ computes
        $\tilde{d}(v, u) \leftarrow \min \{ d_{\leq i}(v, u), \min_{w \in T_{>i}} \{ \tilde{d}(v, w) + d_{\leq i}(w, u) \} \}$
4     Congested Clique model: $v \in T_i$ sends to $u \in T_{\leq i}$, the value $\tilde{d}(v, u)$
```

The outline of our algorithm is as follows. We start by having each node $v \in T_i$ broadcast its incident edges to all the nodes in tiers greater than or equal to its own, that is, to all $u \in T_{\geq i}$, using one round of the Tiered Oracles model (Line 1). Afterwards, in the loop in Line 2, we compute the solution tier by tier, starting from the topmost tier, which contains nodes knowing all the edges in the graph. While processing the i-th tier, every node $v \in T_i$ already knows its distance to every node in $T_{>i}$, and so computes its distances to every node

$u \in T_{\leq i}$. A shortest path between such v and u can either pass through edges which are all known to v, or be broken into a subpath from v to some node $w \in T_{>i}$ and then a path from w to u which is known to v. Thus, we compute the distance from $v \in T_i$ to the nodes $T_{\leq i}$ (Line 3). On Line 4, node $v \in T_i$, which knows for each node $u \in T_{\leq i}$ the distance to u, sends it to u.

For each $u, v \in V$, Algorithm 2 outputs a value $\tilde{d}(u,v)$. We show that it is the correct distance in G, that is $\tilde{d}(u,v) = d(u,v)$.

One round of the Tiered Oracles model suffices for ensuring that for each tier, T_i, every node $v \in T_i$ knows all the edges incident to all the nodes $u \in T_{\leq i}$. Let $v \in T_i$, and $u \in T_j$ such that $i \geq j$, observe that it holds that $\deg(v) \geq 2^i \geq \frac{1}{2}2^j = \frac{1}{2}\deg(u)$, and therefore after Line 1 node v knows the edges incident to u. Thus, each node $v \in T_i$ knows enough information to compute the function $d_{\leq i}$, which is the distance function in G limited to edges incident to nodes in $T_{\leq i}$.

By induction on tier index i, we show that after iteration i of the loop in Line 2 all the nodes in V know the exact distances to all nodes in tiers $T_{\geq i}$.

Base case: In iteration $i = \lceil \log |V| \rceil - 1$, node $v \in T_{\lceil \log |V| \rceil - 1}$ (if exists) in the topmost tier knows about all the edges in E since it knows about all edges incident to nodes $T_{\leq \lceil \log |V| \rceil - 1} = V$. Thus, v can compute the solution to the entire APSP on G, since $d_{\leq \lceil \log |V| \rceil - 1} = d$. Since the set

$$\{\, d(v,w) + d_{\leq \lceil \log |V| \rceil - 1}(w,u) \,\}_{w \in T_{> \lceil \log |V| \rceil - 1}}$$

is empty, we get that $\tilde{d}(v,u) = d_{\leq \lceil \log |V| \rceil - 1}(v,u) = d(v,u)$. That is, node $v \in T_{\lceil \log |V| \rceil - 1}$ computes for each other node $u \in V$ its weighted distance $d(v,u)$ and sends it to u on Line 4.

Induction Step: In iteration $i < \lceil \log |V| \rceil - 1$, consider $v \in T_i$ and $u \in T_{\leq i}$, and let P be a shortest path between them. Recall that node v can locally compute $d_{\leq i}$, and thus knows the value $d_{\leq i}(v,u)$ and for each $w \in T_{>i}$, it knows the value $d_{\leq i}(w,u)$. Further, for each $w \in T_{>i}$, the value $\tilde{d}(v,w) = d(v,w)$ is known to v from one of the previous iterations of the loop in Line 2, by the induction assumption. All values in the set $\{\, \tilde{d}(v,w) + d_{\leq i}(w,u) \,\}_{w \in T_{>i}} \cup \{\, d_{\leq i}(v,u) \,\}$ are either infinite or correspond to some (not necessary simple) path from v to u, thus $\tilde{d}(v,u) \geq d(v,u)$. To show that $\tilde{d}(v,u) \leq d(v,u)$, we consider two cases. If P does not contain nodes from $T_{>i}$, then $d_{\leq i}(v,u) = d(v,u)$ is the length of P. Otherwise, let $w' \in T_{>i}$ be the last node on P (closest to u) which belongs to $T_{>i}$. By the induction hypothesis, v knows $\tilde{d}(v,w') = d(v,w')$. Moreover, the subpath from w' to u only contains edges with at least one endpoint incident to node in $T_{\leq i}$, thus $d_{\leq i}(w',u) = d(w',u)$. For this node w' the value $\{\, \tilde{d}(v,w') + d_{\leq i}(w',u) \,\}$ belongs to $\{\, \tilde{d}(v,w) + d_{\leq i}(w,u) \,\}_{w \in T_{>i}}$. Thus, in both cases the computed $\tilde{d}(v,u)$ is at most the weighted length of P. Hence, $\tilde{d}(v,u) = d(v,u)$. On Line 4, node v informs u about the correct $d(v,u)$, which completes the induction proof.

Lines 1 and 4 each take a single round of the Tiered Oracles model and the Congested Clique model, respectively, and thus the execution of the entire algorithm takes $O(\log |V|)$ rounds of the Congested Clique model and one round of the Tiered Oracles model. ◀

By simulating the algorithm given in Lemma 24 (*APSP in* Congested Clique *with* Tiered Oracles) using Theorem 22 (Tiered Oracles *Simulation*) and Lemma 17 (Congested Clique *Simulation*), we get exact APSP over the skeleton graph, as follows.

▶ **Corollary 25** (Exact APSP on Skeleton Graph). *For any constant $0 < x < 1$, there is an algorithm in the* Hybrid *model that computes an exact weighted APSP on a skeleton graph $S_x = (M, E_S)$, in $\tilde{O}(n^{1-x} + n^{2x-1})$ rounds w.h.p.*

Finally, we extend the result to n^x-RSSP on G, by having each node in the graph learn the information stored in the skeletons in its $\tilde{O}(n^{1-x})$ neighborhood.

▶ **Theorem 3** (n^x-RSSP). *Given a graph $G = (V, E)$, $0 < x < 1$, and a set of nodes M sampled independently with probability n^{x-1}, there is an algorithm in the* Hybrid *model that ensures that every $v \in V$ knows the exact, weighted distance from itself to every node in M within $\tilde{O}(n^{1/3} + n^{2x-1})$ rounds w.h.p.*

Proof. Primarily, assume that $x \geq \frac{2}{3}$. Otherwise, we add each node outside of M with probability $(n^{-1/3} - n^{x-1})/(1 - n^{x-1})$ into the set M. Thus, each node has probability exactly $(n^{x-1} \cdot 1) + (1 - n^{x-1}) \cdot (n^{-1/3} - n^{x-1})/(1 - n^{x-1}) = n^{-1/3}$ to be sampled into M, ensuring $x = 2/3$. We use Claim 14 (*Skeleton from Random Nodes*) to build a skeleton graph $S_x = (M, E_S)$ in $\tilde{O}(n^{1/3})$ rounds w.h.p. Then, we compute exact APSP on the skeleton graph using Corollary 25 (*Exact APSP on Skeleton Graph*) in $\tilde{O}(n^{1/3} + n^{2x-1})$ rounds w.h.p. By Property 5 of the skeleton graph, for each $v, u \in M$ it holds that $d_S(v, u) = d(v, u)$, where $d_S(v, u)$ is the distance in the skeleton graph. So, we apply Claim 15 (*Extend Distances*) with $\alpha = 1, \beta = 0$ and set of sources $V' = M$, to compute an exact weighted shortest paths distances, from M to all of V, in additional $\tilde{O}(n^{1/3})$ rounds w.h.p. ◀

Instantiating Theorem 3 with $x = 2/3$ gives $n^{2/3}$-RSSP in $\tilde{\Theta}(n^{1/3})$ rounds w.h.p., which is tight due to our lower bound given in Theorem 4 (*Lower Bound Exact Shortest Paths, Sources Sampled i.i.d*). We extensively use our $n^{2/3}$-RSSP algorithm for our following results.

4.3 Exact $n^{1/3}$-SSP

We now present an improvement over the warm-up exact SSSP algorithm which we showed previously, by providing an algorithm for exact shortest paths from a *given* set of $n^{1/3}$ nodes ($n^{1/3}$-SSP) in $\tilde{O}(n^{1/3})$ rounds. To do so, we create a skeleton graph and use our algorithm for $n^{2/3}$-RSSP algorithm to compute exact distances from the skeleton nodes to the entire graph. Then, we adapt the behavior of the source nodes depending on the number of skeleton nodes in their neighborhood (which is proportional to the density of the neighborhoods). That is, nodes in sparse neighborhoods can broadcast the distances from themselves to all the skeleton nodes which they see surrounding them, while a node in dense neighborhoods can take over a skeleton node surrounding it and use it as a proxy to communicate efficiently with the other skeleton nodes in the graph. We formalize this in this section, as well as refer to Lemma 26 (*Reassign Skeletons*) which is a generic tool which performs this action of *taking over* skeleton nodes as proxies.

We show the following fundamental algorithm, which allows *assigning* skeletons to help other skeletons. That is, given a set of nodes A where each node in A sees many skeleton nodes in its neighborhood, it is possible to assign skeleton nodes to service the nodes of A. We use this to increase sending and receiving *capacity* of the nodes of A. This is a key tool which we use in the proof of Theorem 5 (*Exact $n^{1/3}$ Sources Shortest Paths*) and we believe it may be useful for additional tasks.

▶ **Lemma 26** (Reassign Skeletons). *Given graph $G = (V, E)$, a skeleton graph $S_x = (M, E_S)$, a value k which is known to all the nodes, and nodes $A \subseteq V$ such that each $u \in A$ has at least $\tilde{\Theta}(k \cdot |A|)$ nodes $M_u \subseteq M$ in its $\tilde{\Theta}(n^{1-x})$ neighborhood, there is an algorithm that assigns $K_u \subseteq M_u$ nodes to u, where $|K_u| = \tilde{\Omega}(k)$, such that each node in M is assigned to at most $\tilde{O}(1)$ nodes in A. With respect to the set A, it is only required that every node in G must know whether or not it itself is in A – that is, the entire contents of A do not have to be globally known. The algorithm runs in $\tilde{O}(n^{1-x})$ rounds in the* Hybrid *model, w.h.p.*

Proof. The pseudocode is provided by Algorithm 3.

Algorithm 3 **Reassign-Skeletons(A, k).**

```
1 Compute $|A|$ by running Aggregate-And-Broadcast
2 Skeleton node $v \in M$ learns its $\tilde{O}(n^{1-x})$-hop neighborhood
3 Skeleton node $v \in M$ samples each $u \in A \cap N_G^{\tilde{\Theta}(n^{1-x})}(v)$ with probability $\frac{1}{|A|}$
4 Skeleton node $v \in M$ informs each sampled node $u$ about $v \in K_u$
```

First, each node w learns the size of the set A by invoking using aggregate and broadcast routine [8, Theorem 2.2] with value 1 if $w \in A$ and 0 otherwise, and the summation function (Line 1). Then, each skeleton node $v \in M$, learns its $\tilde{\Theta}(n^{1-x})$-hop neighborhood (Line 2), and in particular it learns the nodes $A_v = A \cap N_G^{\tilde{\Theta}(n^{1-x})}(v)$. Then, v samples each $u \in A_v$ independently with probability $\frac{1}{|A|}$ (Line 3). Afterwards, v informs each node u it sampled on the previous stage that $v \in K_u$ (Line 4).

For every $v \in M$, since $|A_v| \leq |A|$, and since v samples nodes from there A_v independently with probability $\frac{1}{|A|}$, by Chernoff Bounds each v assigns itself to at most $\tilde{O}(1)$ nodes $a \in A_v$ w.h.p. Hence, by a union bound over all skeleton nodes, each skeleton node is assigned to $\tilde{O}(1)$ nodes w.h.p.

For every $u \in A$, since it is sampled by at least $\tilde{\Omega}(k \cdot |A|)$ skeleton nodes independently with probability $\frac{1}{|A|}$, by Chernoff Bounds it is sampled by $|K_u| = \tilde{\Omega}(1)$ skeleton nodes w.h.p. Thus, by union bound over all skeleton nodes, each $u \in A$ has $|K_u| = \tilde{\Omega}(1)$ assigned nodes w.h.p.

By [8, Theorem 2.2], Line 1 takes $\tilde{O}(1)$ rounds w.h.p., and Lines 2 and 4 take $\tilde{O}(n^{1-x})$ rounds, and thus the entire execution completes in $\tilde{O}(n^{1-x})$ rounds w.h.p. ◀

Now we apply Token Dissemination [9, Theorem 2.1] or Lemma 26 (*Reassign Skeletons*) depending on density of each source's neighborhood and show how to compute exact $n^{1/3}$-SSP in $\tilde{O}(n^{1/3})$ rounds. For sources in "sparse" neighborhoods, in which there is a small number of skeleton nodes, we use Token Dissemination to inform all nodes about their distances to those skeletons. For source v with "dense" neighborhood, in which there are many skeleton nodes, we use Lemma 26 (*Reassign Skeletons*) to get at least one skeleton node u which participates in the round of the Congested Clique model on behalf of that source and sends each other skeleton node v' the distance $d(v, v')$.

▶ **Theorem 5** (Exact $n^{1/3}$ Sources Shortest Paths)**.** *Given a weighted graph $G = (V, E)$, and a set of sources U, such that $|U| = O(n^{1/3})$, there exists an algorithm, at the end of which each $v \in V$ knows its distance from every $s \in U$, which runs in $\tilde{O}(n^{1/3})$ rounds w.h.p.*

Proof. The pseudocode for the algorithm appears in Algorithm 4.

Without loss of generality the set of nodes U is globally known (it can be disseminated in $\tilde{O}(n^{1/6})$ rounds w.h.p. using Token Dissemination from [9, Theorem 2.1]). We build $M \subseteq V$ by marking nodes independently with probability $n^{-1/3}$ (Line 1). Then we run the algorithm from Theorem 3 (*n^x-RSSP*) with $x = 2/3$ to obtain w.h.p. $n^{2/3}$-RSSP from the set of nodes M (Line 2), such that w.h.p. every $u \in V$ knows its distance to every node in M. Afterwards, we apply Claim 14 (*Skeleton from Random Nodes*) to construct a skeleton graph $S_{2/3} = (M, E_S)$ w.h.p. Then, each source learns the information in its h-hop neighborhood (Line 4), for $h \in \tilde{\Theta}(n^{1/3})$. In particular, it counts the skeleton nodes in its h-hop neighborhood.

Algorithm 4 **Exact-$n^{1/3}$-SSP**: Computes an exact weighted $n^{1/3}$-SSP. Routine for node $u \in V$.

```
1  Join $M$ independently with probability $n^{-1/3}$
2  Compute $n^{2/3}$-RSSP from $M$
3  Construct skeleton graph $S_{2/3} = (M, E_S)$
4  Learn $h = \tilde{\Theta}(n^{1/3})$-hop neighborhood
5  if $u \in U$ then
6      if $\left|N_G^h(u) \cap M\right| = \tilde{O}(n^{1/3})$ then
7          Participate in Token-Dissemination with a token $\langle u, v', d(v', u)\rangle$ for each
            $v' \in M \cap N_G^h(s)$
8      else
9          $K_u \leftarrow$Reassign-Skeletons $\left(\{\, u\colon \left|N_G^h(u) \cap M\right| = \tilde{\Omega}(n^{1/3}) \,\}, \tilde{O}(1)\right)$
10         Send each $v \in K_u$ the values $d(u, v')$ for each $v' \in M$
11 if $u \in M$ then
12     In the Congested Clique model: for each $v' \in M$ and each $v \in U$ such that $u \in K_v$
        send $d(v, v')$ to $v'$
13     For each $s \in U$, compute $\tilde{d}(u, s)$ by Equation (1) and output it
14 Apply Claim 15, which given distances from skeleton to sources $\tilde{d}\colon M \times U \mapsto \mathbb{N}$
    extends it to distances from each nodes to sources $\tilde{d}\colon V \times U \mapsto \mathbb{N}$
```

If a source finds that the number of skeleton nodes in its h-hop neighborhood is $\tilde{O}(n^{1/3})$, then it participates in a token dissemination protocol ([9, Theorem 2.1]) and w.h.p. informs all the graph about its distance to these skeleton nodes.

Otherwise, each source $u \in U$ which finds that there are at least $\tilde{\Omega}(n^{1/3})$ skeleton nodes in its h-hop neighborhood, applies Lemma 26 (*Reassign Skeletons*) with $k = \tilde{O}(1)$ and $A = \{\, u\colon \left|N_G^h(u) \cap M\right| = \tilde{\Omega}(n^{1/3}) \,\}$ and receives $K_u \subseteq N_G^h(u) \cap M$, a set of $\tilde{\Omega}(1)$ skeletons. Such a source $u \in U$ sends by local edges to $v \in K_u$ the distance $d(u, v')$ to each $v' \in M$ (Line 10). Each skeleton node $u \in M$ sends the distances $d(s, v')$ to each $v' \in M$, for each source node $s \in U$ that it is assigned to, by simulating the Congested Clique model. If skeleton node $u \in M$ for source $s \in U$ did not receive the distance from s, it computes it using Equation (1) (Line 13) based on the information it received in Line 7.

$$\tilde{d}(u, s) = \min\left\{\, d^h(u, s), \min_{v' \in M}\left\{\, d(u, v') + d^h(v', s) \,\right\} \right\} \tag{1}$$ [4]

After each skeleton knows the distance to each source, we apply Claim 15 to compute distances from sources to all the nodes.

▶ **Lemma 27.** *After Line 13, each $u \in M$ knows $d(v, s)$ for each $s \in U$ w.h.p.*

By Lemma 27, whose proof appears in the full version of the paper [15], each node in M knows the distance to each node in U, thus by Claim 15 (*Extend Distances*) with $\alpha = 1, \beta = 0$ there is an algorithm to compute shortest paths distance from U.

By Theorem 3 (*n^x-RSSP*) with $x = \frac{2}{3}$, Line 2 completes in $\tilde{O}(n^{1/3})$ rounds w.h.p. For Line 3, by Claim 14 (*Skeleton from Random Nodes*), the round complexity, w.h.p., is $\tilde{O}(n^{1/3})$ as well. Line 4 completes in $\tilde{O}(n^{1/3})$ rounds. Since there are at most $\ell = \tilde{O}(n^{1/3})$ tokens per source and $k = \Omega\left(n^{2/3}\right)$ tokens overall, Line 7 takes $\tilde{O}(n^{1/3})$ rounds w.h.p. by [9, Theorem 2.1]. By Lemma 26 (*Reassign Skeletons*), Line 9 takes $\tilde{O}(n^{1/3})$ rounds w.h.p. All skeleton nodes are assigned to some helpers in their $\tilde{O}(n^{1/3})$-hop neighborhood by Line 9, so Line 4 takes $\tilde{O}(n^{1/3})$ rounds. Since each skeleton selects $\tilde{O}(1)$ sources w.h.p. in Line 9 by Lemma 26 (*Reassign Skeletons*), Line 12 simulates $\tilde{O}(1)$ rounds of the Congested Clique model and takes $\tilde{O}(n^{1/3})$ rounds by Lemma 17 (Congested Clique *Simulation*) w.h.p. Finally by Claim 15 (*Extend Distances*), Line 14 for $x = \frac{2}{3}$ takes $\tilde{O}(n^{1/3})$ rounds as well. Thus, the overall execution of the algorithm takes $\tilde{O}(n^{1/3})$ rounds. ◀

References

1 Amir Abboud, Keren Censor-Hillel, and Seri Khoury. Near-linear lower bounds for distributed distance computations, even in sparse networks. In Cyril Gavoille and David Ilcinkas, editors, *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, volume 9888 of *Lecture Notes in Computer Science*, pages 29–42. Springer, 2016. doi:10.1007/978-3-662-53426-7_3.

2 Yehuda Afek, Gad M. Landau, Baruch Schieber, and Moti Yung. The power of multimedia: Combining point-to-point and multiaccess networks. *Inf. Comput.*, 84(1):97–118, 1990. doi:10.1016/0890-5401(90)90035-G.

3 Udit Agarwal and Vijaya Ramachandran. Distributed weighted all pairs shortest paths through pipelining. In *2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2019, Rio de Janeiro, Brazil, May 20-24, 2019*, pages 23–32. IEEE, 2019. doi:10.1109/IPDPS.2019.00014.

4 Udit Agarwal and Vijaya Ramachandran. Faster deterministic all pairs shortest paths in congest model. In Christian Scheideler and Michael Spear, editors, *SPAA '20: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15-17, 2020*, pages 11–21. ACM, 2020. doi:10.1145/3350755.3400256.

5 Udit Agarwal, Vijaya Ramachandran, Valerie King, and Matteo Pontecorvi. A deterministic distributed algorithm for exact weighted all-pairs shortest paths in õ(n $^{3/2}$) rounds. In Calvin Newport and Idit Keidar, editors, *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 199–205. ACM, 2018. doi:10.1145/3212734.3212773.

6 Bertie Ancona, Keren Censor-Hillel, Mina Dalirrooyfard, Yuval Efron, and Virginia Vassilevska Williams. Distributed distance approximation. *CoRR*, abs/2011.05066, 2020. arXiv:2011.05066.

7 Arash Asadi, Vincenzo Mancuso, and Rohit Gupta. An sdr-based experimental study of outband D2D communications. In *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*, pages 1–9. IEEE, 2016. doi:10.1109/INFOCOM.2016.7524372.

8 John Augustine, Mohsen Ghaffari, Robert Gmyr, Kristian Hinnenthal, Christian Scheideler, Fabian Kuhn, and Jason Li. Distributed computation in node-capacitated networks. In Christian Scheideler and Petra Berenbrink, editors, *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 69–79. ACM, 2019. doi:10.1145/3323165.3323195.

9 John Augustine, Kristian Hinnenthal, Fabian Kuhn, Christian Scheideler, and Philipp Schneider. Shortest paths in a hybrid network model. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1280–1299. SIAM, 2020. doi:10.1137/1.9781611975994.78.

10 Ruben Becker, Andreas Karrenbauer, Sebastian Krinninger, and Christoph Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPIcs*, pages 7:1–7:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.DISC.2017.7.

11 Aaron Bernstein and Danupon Nanongkai. Distributed exact weighted all-pairs shortest paths in near-linear time. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 334–342. ACM, 2019. doi:10.1145/3313276.3316326.

12 Keren Censor-Hillel, Michal Dory, Janne H. Korhonen, and Dean Leitersdorf. Fast approximate shortest paths in the congested clique. 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 74–83. ACM, 2019.

13 Keren Censor-Hillel, Michal Dory, Janne H. Korhonen, and Dean Leitersdorf. Fast approximate shortest paths in the congested clique. 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 74–83. ACM, 2019.

14 Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. *Distributed Comput.*, 32(6):461–478, 2019. doi:10.1007/s00446-016-0270-2.

15 Keren Censor-Hillel, Dean Leitersdorf, and Volodymyr Polosukhin. Distance computations in the hybrid network model via oracle simulations. *CoRR*, abs/2010.13831, 2020. arXiv:2010.13831.

16 Keren Censor-Hillel, Dean Leitersdorf, and Elia Turner. Sparse matrix multiplication and triangle listing in the congested clique model. *Theor. Comput. Sci.*, 809:45–60, 2020. doi:10.1016/j.tcs.2019.11.006.

17 Yong Cui, Hongyi Wang, and Xiuzhen Cheng. Channel allocation in wireless data center networks. In *INFOCOM 2011. 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 10-15 April 2011, Shanghai, China*, pages 1395–1403. IEEE, 2011. doi:10.1109/INFCOM.2011.5934925.

18 Michal Dory and Merav Parter. Exponentially faster shortest paths in the congested clique. In Yuval Emek and Christian Cachin, editors, *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, pages 59–68. ACM, 2020. doi:10.1145/3382734.3405711.

19 Michael Elkin. Distributed exact shortest paths in sublinear time. *J. ACM*, 67(3):15:1–15:36, 2020. doi:10.1145/3387161.

20 Michael Feldmann, Kristian Hinnenthal, and Christian Scheideler. Fast hybrid network algorithms for shortest paths in sparse graphs. *CoRR*, abs/2007.01191, 2020. arXiv:2007.01191.

21 Sebastian Forster and Danupon Nanongkai. A faster distributed single-source shortest paths algorithm. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 686–697. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00071.

22 Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. Networks cannot compute their diameter in sublinear time. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1150–1162. SIAM, 2012. doi:10.1137/1.9781611973099.91.

23 François Le Gall. Further algebraic algorithms in the congested clique model and applications to graph-theoretic problems. In Cyril Gavoille and David Ilcinkas, editors, *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, volume 9888 of *Lecture Notes in Computer Science*, pages 57–70. Springer, 2016. doi:10.1007/978-3-662-53426-7_5.

24 Mohsen Ghaffari and Jason Li. Improved distributed algorithms for exact shortest paths. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 431–444. ACM, 2018. doi:10.1145/3188745.3188948.

25 Robert Gmyr, Kristian Hinnenthal, Christian Scheideler, and Christian Sohler. Distributed monitoring of network properties: The power of hybrid networks. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 137:1–137:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.137.

26 Kai Han, Zhiming Hu, Jun Luo, and Liu Xiang. RUSH: routing and scheduling for hybrid data center networks. In *2015 IEEE Conference on Computer Communications, INFOCOM 2015, Kowloon, Hong Kong, April 26 - May 1, 2015*, pages 415–423. IEEE, 2015. doi:10.1109/INFOCOM.2015.7218407.

27 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A deterministic almost-tight distributed algorithm for approximating single-source shortest paths. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 489–498. ACM, 2016. doi:10.1145/2897518.2897638.

28 Stephan Holzer and Nathan Pinsker. Approximation of distances and shortest paths in the broadcast congest clique. In Emmanuelle Anceaume, Christian Cachin, and Maria Gradinariu Potop-Butucaru, editors, *19th International Conference on Principles of Distributed Systems, OPODIS 2015, December 14-17, 2015, Rennes, France*, volume 46 of *LIPIcs*, pages 6:1–6:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.OPODIS.2015.6.

29 He Huang, Xiangke Liao, Shanshan Li, Shaoliang Peng, Xiaodong Liu, and Bin Lin. The architecture and traffic management of wireless collaborated hybrid data center network. In Dah Ming Chiu, Jia Wang, Paul Barford, and Srinivasan Seshan, editors, *ACM SIGCOMM 2013 Conference, SIGCOMM'13, Hong Kong, China, August 12-16, 2013*, pages 511–512. ACM, 2013. doi:10.1145/2486001.2491724.

30 Fabian Kuhn and Philipp Schneider. Computing shortest paths and diameter in the hybrid network model. In Yuval Emek and Christian Cachin, editors, *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, pages 109–118. ACM, 2020. doi:10.1145/3382734.3405719.

31 Christoph Lenzen. Optimal deterministic routing and sorting on the congested clique. In Panagiota Fatourou and Gadi Taubenfeld, editors, *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, pages 42–50. ACM, 2013. doi:10.1145/2484239.2501983.

32 Christoph Lenzen and Boaz Patt-Shamir. Fast partial distance estimation and applications. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 153–162. ACM, 2015. doi:10.1145/2767386.2767398.

33 Christoph Lenzen, Boaz Patt-Shamir, and David Peleg. Distributed distance computation and routing with small messages. *Distributed Comput.*, 32(2):133–157, 2019. doi:10.1007/s00446-018-0326-6.

34 Christoph Lenzen and David Peleg. Efficient distributed source detection with limited bandwidth. In Panagiota Fatourou and Gadi Taubenfeld, editors, *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, pages 375–382. ACM, 2013. doi:10.1145/2484239.2484262.

35 Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 565–573. ACM, 2014. doi:10.1145/2591796.2591850.

36 David Peleg, Liam Roditty, and Elad Tal. Distributed algorithms for network diameter and girth. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, volume 7392 of *Lecture Notes in Computer Science*, pages 660–672. Springer, 2012. doi:10.1007/978-3-642-31585-5_58.

37 Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, 2011. doi:10.1007/s00453-010-9401-5.

38 Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM J. Comput.*, 41(5):1235–1265, 2012. doi:10.1137/11085178X.

39 Jeffrey D. Ullman and Mihalis Yannakakis. High-probability parallel transitive-closure algorithms. *SIAM J. Comput.*, 20(1):100–125, 1991. doi:10.1137/0220006.

40 Stefano Vissicchio, Laurent Vanbever, and Olivier Bonaventure. Opportunities and research challenges of hybrid software defined networks. *Comput. Commun. Rev.*, 44(2):70–75, 2014. doi:10.1145/2602204.2602216.

41 Guohui Wang, David G. Andersen, Michael Kaminsky, Konstantina Papagiannaki, T. S. Eugene Ng, Michael Kozuch, and Michael P. Ryan. c-through: part-time optics in data centers. In Shivkumar Kalyanaraman, Venkata N. Padmanabhan, K. K. Ramakrishnan, Rajeev Shorey, and Geoffrey M. Voelker, editors, *Proceedings of the ACM SIGCOMM 2010 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, New Delhi, India, August 30 -September 3, 2010*, pages 327–338. ACM, 2010. doi:10.1145/1851182.1851222.

Simple Multi-Pass Streaming Algorithms for Skyline Points and Extreme Points

Timothy M. Chan
Dept. of Computer Science, University of Illinois at Urbana-Champaign, IL, USA

Saladi Rahul
Dept. of Computer Science and Automation, Indian Institute of Science Bangalore, India

Abstract

In this paper, we present simple randomized multi-pass streaming algorithms for fundamental computational geometry problems of finding the skyline (maximal) points and the extreme points of the convex hull. For the skyline problem, one of our algorithm occupies $O(h)$ space and performs $O(\log n)$ passes, where h is the number of skyline points. This improves the space bound of the currently best known result by Das Sarma, Lall, Nanongkai, and Xu [VLDB'09] by a logarithmic factor. For the extreme points problem, we present the first non-trivial result for any constant dimension greater than two: an $O(h \log^{O(1)} n)$ space and $O(\log^d n)$ pass algorithm, where h is the number of extreme points. Finally, we argue why randomization seems unavoidable for these problems, by proving lower bounds on the performance of deterministic algorithms for a related problem of finding maximal elements in a poset.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases multi-pass streaming algorithms, skyline, convex hull, extreme points, randomized algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.22

Funding *Timothy M. Chan*: Work supported in part by NSF Grant CCF-1814026.
Saladi Rahul: Work supported by IISc start-up research grant.

1 Introduction

1.1 Multi-pass streaming model

The *streaming* model has emerged as a popular model to handle massive data. Unfortunately, streaming algorithms for geometric problems that make a single pass over the input and work with a small amount of space are typically unable to give exact solutions. This motivates the *multi-pass* streaming model, where the algorithm is allowed to make multiple passes over the input. The input sequence remains unchanged in each pass. The goal is to minimize the amount of working space (or memory) and the number of passes. The data is assumed to be explicitly stored either in a disk or in a cloud, which facilitates multiple passes over it, but since each pass is costly it is essential to minimize the passes.

Summarization queries are the most widely studied class of problems in the streaming model. The focus of this paper is *geometric* summarization queries in the multi-pass streaming model. Specifically, we study two fundamental geometric summarization problems: the *skyline* problem, asking for the "dominating" points in the data, and the *extreme points* problem, asking for the vertices of the convex hull, which succinctly represents the shape of the point cloud. The goal is to design algorithms which are *output-sensitive* in space (i.e., near $O(h)$ space when there are h skyline/extreme points) and perform few passes. Note that $O(h)$ space[1] is indeed the best possible if we want to store the skyline/extreme points in memory, rather than write to an output stream.

[1] Throughout the paper, all space bounds are measured in words, not bits. A word may store one input point, or an $O(\log n)$-bit number.

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 22; pp. 22:1–22:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We will focus on algorithms which are optimized to work efficiently even for the worst-case input. For the problems of interest in this paper (such as the skyline problem), one could argue that an incremental algorithm which updates the skyline as each new element is inserted will work well for randomly generated inputs (for example, for points uniformly distributed in a square, any prefix of the input has an expected $O(\log n)$ number of skyline points, and so it is not difficult to obtain a solution using $O(\log n)$ expected space, with just one pass). However, point sets encountered in practice may not be randomly distributed.

1.2 Skyline points

Let P be a set of n points lying in $\mathbb{R}^d$ for a constant d. A point $p = (p_1, \ldots, p_d)$ *dominates* another point $q = (q_1, \ldots, q_d)$ if $p_i > q_i$ for all $i \in \{1, \ldots, d\}$. In the *skyline* (also called *maxima*) problem, the goal is to find all points $p \in P$ such that p is not dominated by any other point in P. The problem has been extensively studied by the computational geometry [20] and the database community (e.g., see [22] and the references therein). Currently, the best known result in the word-RAM model is an $O(n \log^{d-3} n)$-time algorithm by Chan, Larsen and Pătraşcu [6] (also see [1] and [14] for the best-known output-sensitive algorithms in the word-RAM model and the I/O model, respectively, and [2] for instance-optimal algorithms in $\mathbb{R}^2$ and $\mathbb{R}^3$).

The formal study of the skyline problem in the multi-pass streaming model was initiated by Das Sarma, Lall, Nanongkai, and Xu [11]. The naive $O(nh)$-time output-sensitive algorithm (e.g., see [8]) can be implemented in the multi-pass setting with $O(h)$ space but requires $O(h)$ passes, where h is the number of skyline points of P. Das Sarma et al. proposed a new randomized algorithm using significantly fewer number of passes: it requires $O(h \log n)$ space and just $O(\log n)$ passes, with high probability,[2] for any constant dimension d.

Alternatively, it is not difficult to obtain a deterministic algorithm with $O(h \log n)$ space and $O(\log^{d-1} n)$ passes, by adapting Kirkpatrick and Seidel's output-sensitive skyline algorithm [16] in the multi-pass setting, similar to Chan and Chen's multi-pass adaptation [5] of Kirkpatrick and Seidel's output-sensitive 2-d convex hull algorithm [17]; see the appendix. However, with this approach, the number of logarithmic factors grows as the dimension increases.

New randomized algorithms. Our first result is a variant of Das Sarma et al.'s algorithm that solves the d-dimensional skyline problem using $O(\log n)$ passes and $O(h)$ space – this improves space by a logarithmic factor. Our bounds also hold with high probability. Although the improvement is not big, the highlight here is the simplicity of our analysis compared to the longer and more complicated analysis by Das Sarma et al. [11]. (The simpler analysis is well-suited for teaching purposes.) Also, unlike their analysis, which is specialized to the skyline problem, our analysis naturally extends to the extreme points problem, as we will discuss later.

Our algorithm can also achieve a trade-off: by increasing the space bound to $O(bh)$ for a parameter b, the number of passes can be lowered to $O(\log_b n)$. For example, setting $b = n^\delta$ gives $O(hn^\delta)$ space and $O(1/\delta)$ passes, for any $\delta \in (0, 1]$. Setting $b = \log^\delta n$ for an arbitrarily small constant $\delta > 0$ gives $O(h \log^\delta n)$ space and $O\left(\frac{\log n}{\log \log n}\right)$ passes.

In the $O(h)$-space regime, we also describe a refinement of the algorithm that further reduces the number of passes from $O(\log n)$ to $O\left(\log h + \frac{\log n}{\log \log n}\right)$, which is slightly sublogarithmic, assuming that h is not too big. These bounds hold in expectation.

[2] Throughout this paper, "with high probability" will imply "with probability at least $1 - \frac{1}{n^c}$", where c is a sufficiently large constant.

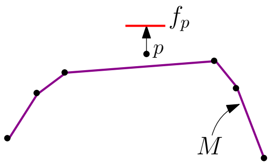
Our randomized algorithms for skyline points, like previous work [11], extends (with the same bounds) to the general setting of a partially ordered set, or *poset*. A poset is a pair $(P, \succ)$, where P is the set of n elements and $\succ$ is an irreflexive, transitive binary relation on the elements of P. The problem here is to find all the *maximal* elements in a poset, i.e., elements $a \in P$ such that there is no element $b \in P$ with $b \succ a$. We only assume an oracle that can test whether $a \succ b$ for any two given elements a and b.

Is randomization essential? A natural question is whether randomization is essential for the algorithms proposed in this paper. At least for the poset problem we can answer this question. We show that any deterministic algorithm which uses $O(h)$ space to find all maximal points in a poset has to perform $\Omega(h)$ passes. In other words, among the class of deterministic algorithms to compute maximal elements in a poset, the naive idea of finding one maximal element per pass is the best possible algorithm. Therefore, randomization is not just necessary, but in fact leads to dramatically improved results. Our lower bound proof is based on a new, interesting adversarial argument.

1.3 Extreme points

Given a set P of n points lying in $\mathbb{R}^d$, a point $p \in P$ is an *extreme point* if $conv(P) \neq conv(P \setminus \{p\})$, where $conv(P)$ denotes the convex hull of P. In the multi-pass setting, this problem was first studied in $\mathbb{R}^2$ by Chan and Chen [5], who obtained an algorithm with $O(h \log^2 n)$ space and $O(\log^2 n)$ passes, and then recently by Farach-Colton, Li, and Tsai [12], who improved the bounds to $O(h \log^2 n)$ space and $O(\log n)$ passes. These solutions are based on the output-sensitive divide-and-conquer algorithms of Kirkpatrick and Seidel [17] and Chan, Snoeyink, and Yap [7], and hence, they inherently work only in $\mathbb{R}^2$ (and possibly in $\mathbb{R}^3$) – in dimension greater than three, these divide-and-conquer algorithms have complexity at least the number of hull facets, which can be much larger than the number of hull vertices.

In this work, we present the first non-trivial result for any constant dimension greater than two: an $O(h \log^{O(1)} n)$-space, $O(\log^d n)$-pass algorithm. Our solution requires extending our skyline algorithm in a nontrivial fashion.



In the non-streaming setting, $O(nh)$-time output-sensitive algorithms for the extreme points problem were reported independently by Clarkson [8], Chan [3], Ottmann et al. [19], and Dula and Helagason [15]. These algorithms are all similar and work by incrementally building a subset $M \subseteq P$ of the extreme points of the upper hull. For each input point $p \in P$, we first check if p lies below or above the current upper hull of M – this step reduces to a linear program with $O(h)$ constraints. If p lies below, then it can be removed. Otherwise, p may not necessarily be extreme in P, but we can shoot an upward ray from p to hit a facet $f_p \in conv(P)$ – this step reduces to a linear program with n constraints.[3] The $d+1$ vertices defining f_p are extreme in P and can be added to M. The whole algorithm requires solving

[3] A version of the algorithm by Clarkson [8] and Dula and Helagason [15] avoids linear programming in this step and instead finds a point extreme in the direction orthogonal to f_p, and adds to M. If p is still above the current upper hull of M, we repeat.

$O(n)$ linear programs with $O(h)$ constraints and $O(h)$ linear programs with n constraints, and thus takes $O(nh)$ time by known linear-time linear programming algorithms in constant dimensions [4, 9, 21].[4] Because it finds a constant number of extreme points per iteration, the algorithm is inherently *sequential* and requires $\Omega(h)$ passes.

In our solution, we use randomization instead to find more extreme points at each iteration, by generating $O(h)$ linear programs with n constraints that can be solved *in parallel*, for each of logarithmically many rounds.

The rest of the paper is organized as follows. In Sections 2 and 3, we present our algorithms for computing skyline points and extreme points, respectively. In Section 4 we prove a lower bound on the performance of any deterministic algorithm for finding the maximal elements in a poset.

2 Randomized algorithms for skyline points in $\mathbb{R}^d$

2.1 Main algorithm

In this section we will prove the following result.

▶ **Theorem 1.** *Let P be a set of n points in $\mathbb{R}^d$, and h be the number of skyline points. Then, with high probability, there is an $O(bh)$-space, $O(\log_b n)$-pass algorithm to compute the skyline points, where b is a parameter in the range $[2, n/h]$.*

Our algorithm is a refinement of Das Sarma et al.'s algorithm [11]. The general idea is to use random sampling to somehow prune a fraction of the input points in each round. The right sample size requires knowledge of h, which we guess by repeated doubling.

Let P be a set of n points in $\mathbb{R}^d$. The algorithm consists of two stages. In Stage I, the goal is to handle the case when $h \leq \log_b n$. Here, we can just run a known naive algorithm (e.g., see [8]) which finds one skyline point per pass and thus requires $h \leq \log_b n$ passes. For example, having found skyline points $p_1, \ldots, p_{i-1}$ in the first $i-1$ passes, we can find the point, $p_i \in P$, with the largest x-coordinate value among the points of P not dominated by $p_1, \ldots, p_{i-1}$, in the next pass. If some skyline points have not been found after $\log_b n$ passes, we proceed to Stage II. Let c be a sufficiently large constant.

The set M maintains the current list of skyline points found by the algorithm. In Stage-II, an *iteration* will consist of two passes. Let P_i denote the set of unclassified points at the beginning of the i-th iteration (where $i \geq 1$), i.e., the points of P_i which have not yet been labeled as skyline or non-skyline. In the first pass of the i-th iteration (step 3a), we independently sample each point of P_i with probability $\frac{cr_i}{|P_i|}$, where c is a sufficiently large constant. Let $R_i \subseteq P_i$ be the sampled set of points. In the second pass of the i-th iteration (step 3b), the goal is to find a set of skyline points R_i^+ which dominate *all* the points in R_i (i.e., every point in R_i is dominated by some point in R_i^+). This is achieved as follows: initially, set $R_i^+ \leftarrow R_i$. Then for each point $p \in P_i$, if p does not dominate any point in R_i^+, then we do nothing. Otherwise, we add p to R_i^+ and remove the points of R_i^+ which are dominated by p. Note that this step will not increase the size of R_i^+, although it could potentially decrease the size of R_i^+. At the end of the second pass, the claim is that the points in R_i^+ are all skyline points (see Lemma 2). The algorithm terminates when all the points have been classified.

[4] With range searching data structures, the overall running time can be lowered to $O(n \log^{O(1)} h + (nh)^{1-1/(\lfloor d/2 \rfloor + 1)} \log^{O(1)} n)$ [3]. We will ignore bounds of this flavor, since they do not improve upon $O(nh)$ by much as d increases.

Algorithm 1 Finding Skyline Points.

```
0. $M \leftarrow \emptyset$.
//Stage I to handle $h \leq \log_b n$.
1. For the first $\log_b n$ passes, run the naive algorithm of finding one skyline point per pass.

//Stage II to handle $h > \log_b n$.
2. $P_1 \leftarrow P$, $r_1 \leftarrow c\log_b n$, and $i \leftarrow 1$.
3. Repeat till $|P_i| < r_i$:
3a.   Find a sample $R_i \subseteq P_i$ of size $r_i$. Set $R_i^+ \leftarrow R_i$.
3b.   For each point $p \in P_i$:   //Finding new skyline points.
-         Add $p$ to $R_i^+$ if $p$ dominates at least one point in $R_i^+$.
-         Remove the points in $R_i^+$ which are dominated by $p$.
3c.   $M \leftarrow M \cup R_i^+$.
3d.   $P_{i+1} \leftarrow P_i \setminus (R_i^+ \cup \{$points of $P_i$ dominated by $R_i^+\})$.
3e(i).   If $|P_{i+1}| \geq |P_i|/b$, then $r_{i+1} \leftarrow br_i$.   // Ineffective iteration.
3e(ii).  Otherwise, $r_{i+1} \leftarrow r_i$.
3f.   $i \leftarrow i + 1$.

4. Compute the skyline points of the $O(r_i)$ points in $P_i$ and add them to $M$.
```

▶ Remark. Since the size of the sets P_i can be significantly larger than h, we cannot afford to store them explicitly. To overcome this issue, we will instead use M to *implicitly* maintain P_i: observe that whenever a point p in the stream arrives, if none of the points in M dominate p, then $p \in P_i$.

Analysis. Stage I of the algorithm requires $O(\log_b n)$ space and $O(\log_b n)$ passes. From now on we will focus on Stage II and assume $h > \log_b n$. The i-th iteration is labeled *effective* if $|P_{i+1}| < |P_i|/b$. We start by proving a simple fact.

▶ **Lemma 2.** *At the end of the i-th iteration, all the points in R_i^+ are skyline points.*

Proof. For the sake of contradiction, assume that a point $p \in R_i^+$ is not a skyline point and let $q \in P_i$ be a point dominating it. If p appears before q in the stream, then it is easy to observe that p will not survive in R_i^+. On the other hand, if q arrives before p in the stream, then there are two cases:

- $p \in R_i^+$ at the end of the first pass. In that case, we know that in the second pass there is a point which will come before p and remove p from R_i^+. Once that happens, then p cannot be added back to R_i^+ in the second pass.
- $p \notin R_i^+$ in the first pass, but $p \in R_i^+$ at the end of the second pass. This implies that there is a point $p' \in R_i^+$ at the end of the first pass and p' is dominated by p; but p' would have been dominated by q as well, and hence p cannot be part of R_i^+ at the end of the second pass. ◀

The following lemma is the crux of our argument.

▶ **Lemma 3.** *When $r_i \geq cb^2h$ for a sufficiently large constant c, then all further iterations will be effective with high probability.*

Proof. Let M^* be the skyline points of P. Consider any i-th iteration in which $r_i \geq cb^2h$. Given a sample R_i, in step 3b we have constructed a set $R_i^+ \subseteq M^*$ which dominates all the points in R_i. The main question is this:

> What is the probability that the number of points of $P_i \setminus R_i^+$ not dominated by R_i^+ is more than $|P_i|/b$?

This probability seems hard to bound directly. We turn the question around:

> Fix a subset $A \subseteq M^*$, where the number of points of $P_i \setminus A$ not dominated by A is more than $|P_i|/b$. What is the probability that $R_i^+ = A$?

Observe that if any point $p \in P_i \setminus A$ not dominated by A is chosen to be in R_i, then p would be dominated by some point in R_i^+, making it impossible for $R_i^+ = A$. Using this observation, we get the following upper bound:

$$\begin{aligned}\Pr[R_i^+ = A] &\leq \Pr[\text{every point of } P_i \setminus A \text{ not dominated by } A \text{ is not in } R_i] \\ &\leq \left(1 - \frac{r_i}{|P_i|}\right)^{|P_i|/b} \\ &\leq e^{-r_i/b}.\end{aligned}$$

A trivial upper bound on the number of candidates for A is 2^h, since $A \subseteq M^*$. It turns out that this trivial bound is sufficient for our purposes. By the union bound,

$$\begin{aligned}&\Pr[\text{the number of points of } P_i \setminus R_i^+ \text{ not dominated by } R_i^+ \text{ is more than } |P_i|/b] \\ &\quad\leq\ 2^h \cdot e^{-r_i/b} \\ &\quad<\ 2^{-\Omega(cbh)} \qquad \text{since } r_i \geq cb^2 h \\ &\quad=\ n^{-\Omega(c)} \qquad \text{since } h > \log_b n.\end{aligned}$$

It follows that an iteration is effective with high probability. ◀

Readers familiar with ε-nets or standard geometric Clarkson–Shor-style sampling analysis [10, 18] may find the preceding analysis similar to known arguments, but there is one interesting, key difference: the set system we are dealing with does not have constant VC dimension, but rather has dimension $\Theta(h)$. We use the 2^h upper bound on the number of possible sets A, instead of a more usual polynomial bound. (In dimension 2 and 3, one could decompose the region not dominated by $O(h)$ points into $O(h)$ cells of constant complexity, and could therefore use a more standard Clarkson-Shor–style argument, but the size of such decomposition blows up in dimension beyond 3.)

▶ **Lemma 4.** *The number of passes performed by the algorithm is $O(\log_b n)$ and the space occupied by the algorithm is $O(bh)$. Both bounds hold with high probability.*

Proof. The number of effective iterations is $O(\log_b n)$, since in each effective iteration the number of unclassified points go down by a factor of at least b. Now we will bound the number of iterations which are *ineffective*. By Lemma 3, with high probability, an ineffective iteration can only happen when $r_i < cb^2 h$. Since the value of r is increased by a factor of b after each ineffective iteration, the total number of ineffective iterations will be bounded by $O(\log_b(b^2 h))$. Therefore, with high probability the number of passes performed by the algorithm is $O(\log_b n + \log_b(b^2 h)) = O(\log_b n)$.

Since an ineffective iteration can only happen when $r_i < cb^2 h$, with high probability, the space occupied will be $O(b^2 h)$. By choosing $b' = \sqrt{b}$, the space becomes $O(b'h)$ and the number of passes becomes $O(\log_{b'} n)$. ◀

Comparison. Compared to Das Sarma et al.'s algorithm [11], our use of a different algorithm in Stage I ensures that Stage II is only invoked to handle the case where h is sufficiently large, and as a result, we could afford a sample size smaller by a logarithmic factor than the sample size used by [11] and still obtain high probability bounds.

Compared to our analysis, Das Sarma et al.'s analysis [11] is longer and more complicated. It starts by constructing a graph consisting of h components (one per skyline point). To argue that in each pass a constant fraction of the points gets classified, the h components are categorized into *heavy* and *light*, and then a separate analysis is performed on the heavy and the light components. Also, their analysis is specialized to the skyline problem, whereas our analysis will extend to the extreme points problem as well.

Running time. Naively, each pass in Stage II can be implemented in $O(b^2nh)$ time, yielding a total running time of $O(b^2nh\log_b n)$, with high probability. In constant dimensions, we can use orthogonal range searching data structures to implement step 3b, and the total running time can be reduced to $O(n\log^{O(1)} h\log_b n)$.

Posets. The algorithm for skyline points naturally extends to the problem of finding maximal points in a poset by suitably adapting the definition of domination. The only modification needed is in Stage I: we used geometry (the x-coordinate values) to find one skyline point per pass. For poset, replace it with another naive algorithm which finds one maximal point per pass.

2.2 Further refinement

In this subsection, we present an interesting variant of the algorithm which slightly reduces the expected number of passes to sublogarithmic, if h is not too big, while using only $O(h)$ expected space.

▶ **Theorem 5.** *Let P be a set of n points in $\mathbb{R}^d$, and h be the number of skyline points. Then, there is an $O(h)$-space, $O\left(\log h + \frac{\log n}{\log\log n}\right)$-pass algorithm to compute the skyline points. Both bounds hold in expectation.*

The new algorithm is similar to our algorithm in Section 2.1, but with one key difference. An iteration will now be considered ineffective if the sample does not prune away a large number of points (this is as before), *nor* does it discover a large number of new skyline points (this is new). Another minor difference is that in case of an ineffective iteration, we will double the sample size (instead of lying by b). More precisely, the only change in the pseudocode is to replace step 3e(i) with the following:

```
3e(i). If $|P_{i+1}| \geq |P_i|/b$ and $|R_i^+| < h/b^2$, then $r_{i+1} \leftarrow 2r_i$   // Ineffective iteration.
```

We will fix the parameter b so that $\log_b n = b^2$ (and thus $b = \Theta\left(\sqrt{\frac{\log n}{\log\log n}}\right)$).

The following lemma shows that an ineffective iteration is not very likely to happen when the sample size is sufficiently large.

▶ **Lemma 6.** $\Pr[\text{iteration } i \text{ is ineffective} \mid r_i \geq h] \;\leq\; e^{-\Omega(b)}$.

Proof. We modify the proof of Lemma 3.

We have already shown that $\Pr[R_i^+ = A] \le e^{-r_i/b}$. Before, we trivially bound the number of candidates for A by 2^h. This time, we will give a sharper upper bound. An ineffective iteration guarantees that $|A| < h/b^2$, and hence, the number of candidates for A is at most

$$\sum_{k=1}^{h/b^2} \binom{h}{k} \le \frac{h}{b^2} \cdot \binom{h}{h/b^2} \le \frac{h}{b^2} \cdot \left(\frac{eh}{h/b^2}\right)^{h/b^2} = b^{O(h/b^2)}.$$

By the union bound,

$$\begin{aligned}&\Pr\left[\text{the number of points of } P_i \setminus R_i^+ \text{ not dominated by } R_i^+ \text{ is more than } |P_i|/b\right]\\ &\le\ b^{O(h/b^2)} \cdot e^{-r_i/b}\\ &\le e^{-\Omega(\frac{h}{b})} \quad \text{since } r_i \ge h\\ &\le e^{-\Omega(b)} \quad \text{since } h \ge b^2.\end{aligned}$$

◀

▶ **Lemma 7.** *The expected number of passes performed by the algorithm is* $O\left(\log h + \frac{\log n}{\log\log n}\right)$.

Proof. An effective iteration with $|R_i^+| \ge h/b^2$ can happen at most b^2 times. An effective iteration with $|P_{i+1}| < |P_i|/b$ can happen only $O(\log_b n)$ times. Therefore, effective iterations happen $O(\log_b n + b^2) = O\left(\frac{\log n}{\log\log n}\right)$ times.

Let us classify the ineffective iterations into two categories: (a) when $r_i < h$, and (b) when $r_i \ge h$.

The number of ineffective iterations of category (a) is $O(\log h)$, since r_i doubles during each ineffective iteration.

By Lemma 6, in expectation, between two consecutive effective iterations, there can be only $O(1)$ ineffective iterations of category (b). Therefore, the expected number of ineffective iterations of category (b) is $O\left(\frac{\log n}{\log\log n}\right)$. This finishes the proof. ◀

▶ **Lemma 8.** *The expected space used by the algorithm is* $O(h)$.

Proof. Let Y be the number of ineffective iterations in which $r_i \ge h$. The space used is bounded by $r^* = \max_i r_i$, which is at most $h \cdot 2^Y$. It thus remains to show that $\mathbf{E}[2^Y] = O(1)$.

To this end, we consider the following probability exercise:

> Let t be an integer and $\rho \le 1/(8t)$. Consider a sequence of independent tosses of a biased coin, where the probability of heads is ρ. Stop the process when we encounter t tails. Let H be the number of heads encountered. Show that $\mathbf{E}[2^H] = O(1)$.

It is straightforward to see that $\Pr[H = j] \le \binom{t+j}{j}\rho^j$. If $j < t$, this probability is at most $(2t)^j \rho^j \le 1/4^j$. If $j \ge t$, the probability is at most $(2j)^t \rho^j \le 1/4^j$, since the function $f(x) = (2x)^t \rho^x \cdot 4^x$ is decreasing for $x \ge t$ and has value at most 1 at $x = t$. Thus, $\mathbf{E}[2^H] \le \sum_j 2^j \cdot 1/4^j = O(1)$.

The result now follows, by associating heads with ineffective iterations and tails with effective iterations, where $t = O(\log_b n + b^2) = O\left(\frac{\log n}{\log\log n}\right)$ (from the proof of Lemma 7) and $\rho = e^{-\Omega(b)}$ (by Lemma 6). ◀

3 Extreme points in $\mathbb{R}^d$

In this section we build on the ideas used for the skyline algorithm to solve the extreme points problem. The following result is obtained.

▶ **Theorem 9.** *Let P be a set of n points in $\mathbb{R}^d$, and h be the number of extreme points. Then, with high probability, there is an $O(h \log^{O(1)} n)$ space and $O(\log^d n)$ pass algorithm to compute the extreme points.*

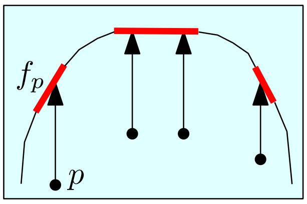
It suffices to focus on computing the extreme points on the upper hull of P (finding the extreme points on the lower hull is symmetric). Our algorithm for the extreme points problem will also work in two stages. In Stage I, we will use the expensive $O(nh)$-time algorithm mentioned in the Introduction and let it run for $O(\log n)$ passes. If $h < \log n$, then $O(\log n)$ passes will be enough to find all the extreme points. Otherwise, we go to Stage II.

Algorithm 2 Finding Extreme Points.

```
0. $M \leftarrow \emptyset$.
//Stage I to handle $h \leq \log n$.
1. For the first $O(\log n)$ passes, run the expensive $O(nh)$-time algorithm.

//Stage II to handle $h > \log n$.
2. $P_1 \leftarrow P$, $r \leftarrow c \log n$ and $i \leftarrow 1$.
3. Repeat till $|P_i| < r$:
3a.   Find a sample $R_i \subseteq P_i$ of size $r$. Set $R_i^+ \leftarrow \emptyset$.
3b.   For each point $p \in R_i$:   //Finding new extreme points.
-         Shoot a vertical ray upwards from $p$ to hit a facet $f_p \in conv(P)$.
-         Add the $d+1$ vertices defining $f_p$ into $R_i^+$.
3c.   $M \leftarrow M \cup R_i^+$.
3d.   $P_{i+1} \leftarrow P_i \setminus \{$points of $P_i$ that are strictly below the upper hull of $R_i^+\}$.
3e.   If $|P_{i+1}| \geq |P_i|/2$, then $r \leftarrow 2r$.     // Ineffective iteration.
3f.   $i \leftarrow i + 1$.

4. Output the extreme points of the $O(r)$ points in $P_i$.
```



Unlike the skyline algorithm, there is no notion of domination for the extreme points problem. Therefore, step 3b of the skyline algorithm cannot be used here. Instead, we perform the following operation: from each point $p \in R_i$, shoot a vertical ray upwards to hit a facet $f_p \in conv(P)$, where $conv(P)$ is the convex hull of P. This operation reduces to linear programming on the dual halfspaces of P. There is a known multi-pass streaming algorithm of Chan and Chen [5] which can solve a linear program in any constant dimension d using $O(\log^{O(1)} n)$ space and $O(\log^{d-1} n)$ passes. We can execute all the r linear programming queries *simultaneously*. This will not hurt the number of passes, but instead increase the space to $O(r \log^{O(1)} n)$. At the end of step 3b, we ensure that the upper hull of R_i^+ "covers" the points in R_i.

Analysis. The analysis follows the same steps as in our analysis of the skyline algorithm. The space used is $O(h \log^{O(1)} n)$, and since $O(\log n)$ iterations are performed, the total number of passes required are $O(\log^d n)$.

To prove an equivalent statement as Lemma 3, let M^* be the extreme points on the upper hull of P and let $b \leftarrow 2$. Fix a subset $A \subseteq M^*$, where the number of points of P_i above the upper hull of A is more than $|P_i|/2$. If any point $p \in P_i \setminus A$ above the upper hull of A is chosen to be in R_i, then p will be "covered" by the upper hull of R_i^+, making $R_i^+ = A$ impossible. So, the same argument as before shows $\Pr[R_i^+ = A] \le e^{-\Omega(ch)}$. Thus, as before, the probability that the number of points of P_i above the upper hull of R_i^+ is more than $|P_i|/2$ is at most $n^{-\Omega(c)}$.

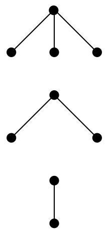
Running time and trade-offs. Each pass in Stage II can be implemented in $O(nh\log^{O(1)} n)$ time, since with Chan and Chen's algorithm [5], the r linear programs take $O(nr\log^{O(1)} n)$ time. Recall that P_i is represented implicitly; in each pass, we can test whether a point p is in P_i by testing whether p is covered by the upper hull of M, which reduces to solving a linear program on $O(h)$ points. The extra cost is $O(nh)$ per pass. The total time is thus $O(nh\log^{O(1)} n)$. (In the traditional non-streaming setting, the total running time is actually $O(nh)$, as it can be bounded by a geometric series.)

As before, it is possible to adapt the algorithm to achieve a trade-off, with $O(b^{O(1)}h\log^{O(1)} n)$ space and $O((\log_b n)^{O(1)})$ passes for a parameter b, since Chan and Chen's multi-pass linear programming algorithm [5] supports a trade-off. For example, setting $b = n^{\Theta(\delta)}$ gives $O(hn^{\delta})$ space and $O((1/\delta)^{O(1)})$ passes.

4 Why randomized algorithms?

We finish by proving that there does *not* exist any efficient deterministic algorithm for the problem of finding maximal elements of a poset. This justifies the use of randomization in the paper (at least for the poset problem). Our lower bound proof is based on a new and self-contained adversarial argument.

▶ **Theorem 10.** *Let $h = \Omega(1)$ and $p \cdot h \ll n$, where p is the number of passes made by an algorithm. Assume that the only operations on the input elements are pairwise comparisons. Then any deterministic algorithm which uses $O(h)$ space has to perform $p \ge \frac{h}{3} + 1$ passes to decide whether the number of maximal elements in a poset is $h+1$, or $h+2, \ldots,$ or $h+6$.*



Let P be the elements in our partially ordered set (poset). The queries asked by the algorithm will be of the form $q(a, b)$, where $a \in P$ is currently stored in the memory and $b \in P$ is the current element in the stream. The response of the adversary will either be $a \succ b$ which implies a *dominates* b, or $b \succ a$ which implies b dominates a, or $a \not\succ b$ which implies a and b are *incomparable*. The responses of the adversary has be *consistent*, i.e., once it responds to a query $q(a, b)$, then the answer to it cannot change later. Before the algorithm begins, the adversary will maintain that all the elements are incomparable. Each time, after seeing $n/3$ elements in the stream, the adversary will create dominance relationship between some pairs of elements by revealing a two-level tree (examples of two-level trees shown in the figure on the right), where the root element dominates its child elements. Therefore, the root of each tree is a maximal element, and hence, the number of maximal points in P will be equal to the number of trees constructed by the adversary.

If an element belongs to a tree revealed till now by the adversary, then it will be labelled *locked*; otherwise, it will be labelled *unlocked*. We will show that if the number of passes performed by the algorithm is less than or equal to $h/3$, then the adversary can arrange the unlocked elements in at least two consistent ways, each having different number of maximal elements. This implies that the execution of the algorithm is exactly the same for two different inputs, which is a contradiction. Now we will present the technical details.

Adversary's strategy. We will need a couple of definitions to set up adversary's strategy. A *time-unit* corresponds to processing a single element in the stream. Each pass is divided into three *phases*, with each phase lasting $n/3$ time-units. For a p-pass algorithm, this naturally leads to a labelling of the phases as $1, 2, 3, \ldots, 3p{-}2, 3p{-}1, 3p$. At the end of the i-th phase, a tree T_i is created by the adversary.

The elements are partitioned into three equal-sized slabs: slab 0 consists of the first $n/3$ elements in the stream, slab 1 consists of the middle $n/3$ elements in the stream, and slab 2 consists of the last $n/3$ elements in the stream. Before the algorithm begins, all the elements are called *short-lived*, and if at any point an element remains in memory continuously for $n/3$ time-units, then we start calling it *long-lived*.

Now we are ready to describe the construction of a tree T_i. The dominated elements in T_i will be those elements in slab $(i-2) \bmod 3$ which were short-lived till the end of the $(i-1)$-th phase, but became long-lived at the end of the i-th phase. Next, we describe the strategy for picking the maximal element of T_i. The adversary will arbitrarily pick *one* among all the elements which satisfy the following conditions. The element should

1. belong to slab $(i \bmod 3)$,
2. not belong to any of the trees already constructed, and
3. not be present in the memory at the end of the previous phases.

The reason for imposing these conditions will become clear in the proof of Lemma 12.

▶ **Lemma 11.** *There always exists an element which satisfies the above conditions. In fact, at least $n/6$ elements in a slab satisfy the above conditions.*

Proof. We claim that the number of long-lived elements are $O(ph)$. The key observation is that for an element to become long-lived, it has to be stored in memory at the end of at least one phase. Since the number of phases are $O(p)$, there can be at most $O(ph)$ long-lived elements. Therefore, the number of elements of slab $i \bmod 3$ which belong to the trees already constructed are $O(ph) + O(h) \ll n/12$. Also, the number of elements in slab $i \bmod 3$ which are present in the memory at the end of any phase is $O(ph) \ll n/12$. Since slab $i \bmod 3$ consists of $n/3$ elements, there will be at least $n/3 - n/6 = n/6$ elements satisfying the above conditions. ◀

When the algorithm asks a query $q(a, b)$, the adversary reports $a \succ b$ or $b \succ a$ if that relation holds in any of the trees constructed till now; otherwise it reports $a \not\succ b$. Next, we argue that the responses of the adversary to the queries are consistent.

▶ **Lemma 12.** *If the adversary places a relation $a \succ b$ in the poset, then the algorithm must not have asked the query $q(a, b)$ or $q(b, a)$ till then. This ensures that responses of the adversary are consistent.*

Proof. Without loss of generality, assume that the element b is in slab 0 and the element a in slab 2 (the other cases can be handled symmetrically). Note that this satisfies the condition that an element from slab $(i-2) \bmod 3$ is dominated only by an element from slab $i \bmod 3$. For a query $q(a,b)$ to be asked during the j-th pass, a should be stored in memory at the beginning of the j-th pass. This implies that a is stored in memory at the end of the $3(j-1)$-th phase, which violates condition 3 for picking the maximal element.

Now we prove that the query $q(b,a)$ was not asked. In a given pass, at the end of which phase does b newly become long-lived? It turns out to be the end of the second phase. Then, let j be the smallest index such that at the end of the second phase in the j-th pass, b was still in memory. This is when $a \succ b$ will be created by the adversary, since b has newly become long-lived. Now, if $q(b,a)$ was asked (say, in the i-th pass) before $a \succ b$ was placed, then b should be in memory at the end of the second phase of the i-th pass and in fact, it should be in memory in the third phase of the i-th pass till a is processed. This implies that b becomes long-lived in the i-th pass, which contradicts that j is the smallest index. ◀

Handling unlocked elements. Since each tree corresponds to revealing only one maximal element, this strategy of the adversary will ensure that the algorithm is forced to perform $h/3$ passes at which point h maximal elements will be revealed. If further passes are not performed, then the algorithm will have the same outcome for more than two inputs. The details follow next.

Let b, b' be any two unlocked elements in slab 2 which satisfy the three conditions stated above for being a maximal element (by Lemma 11 we know at least two such elements in slab 2 still exist). Also, observe that all the unlocked elements in slab 0 are short-lived and by using an argument similar to Lemma 12, it can be shown that queries of the form $q(b,a)$ or $q(a,b)$ or $q(b',a)$ or $q(a,b')$ were not asked by the algorithm, where a is a short-lived unlocked element in slab 0. As a result, the adversary will be consistent if it declares that the unlocked elements in slab 0 are dominated by either b or b'. Now the adversary has two choices: (i) either make only b or b' the root of a tree and all the unlocked elements in slab 0 the leaves of that tree, or (ii) make two trees with b and b' as the root of those trees, and the unlocked elements in slab 0 are partitioned to be the leaves of the two trees.

A similar argument holds when maximal elements are chosen from slab 0 and slab 1. Therefore, the number of maximal elements can be anywhere in the range $(h, h+6]$, if the number of passes performed are at most $h/3$.

▶ Remark. The assumption that $p \cdot h \ll n$ is needed, since for large h, there is a trivial deterministic algorithm with $O(h)$ space and $O(n/h)$ passes (we can divide the input sequence into $O(n/h)$ blocks of h elements, and in the i-th iteration, load the i-th block in memory and test which of the h elements in the block are maximal).

References

1 Peyman Afshani. Fast computation of output-sensitive maxima in a word RAM. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1414–1423, 2014.

2 Peyman Afshani, Jérémy Barbay, and Timothy M. Chan. Instance-optimal geometric algorithms. *Journal of the ACM*, 64(1):3:1–3:38, 2017.

3 Timothy M. Chan. Output-sensitive results on convex hulls, extreme points, and related problems. *Discrete & Computational Geometry*, 16(4):369–387, 1996.

4 Timothy M. Chan. Improved deterministic algorithms for linear programming in low dimensions. *ACM Trans. Algorithms*, 14(3):30:1–30:10, 2018. doi:10.1145/3155312.

5 Timothy M. Chan and Eric Y. Chen. Multi-pass geometric algorithms. *Discrete & Computational Geometry*, 37(1):79–102, 2007. doi:10.1007/s00454-006-1275-6.

6 Timothy M. Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proceedings of Symposium on Computational Geometry (SoCG)*, pages 1–10, 2011.

7 Timothy M. Chan, Jack Snoeyink, and Chee-Keng Yap. Primal dividing and dual pruning: Output-sensitive construction of four-dimensional polytopes and three-dimensional Voronoi diagrams. *Discrete & Computational Geometry*, 18(4):433–454, 1997.

8 Kenneth L. Clarkson. More output-sensitive geometric algorithms. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 695–702, 1994.

9 Kenneth L. Clarkson. Las Vegas algorithms for linear and integer programming when the dimension is small. *Journal of the ACM*, 42(2):488–499, 1995. doi:10.1145/201019.201036.

10 Kenneth L. Clarkson and Peter W. Shor. Application of random sampling in computational geometry, II. *Discret. Comput. Geom.*, 4:387–421, 1989. doi:10.1007/BF02187740.

11 Atish Das Sarma, Ashwin Lall, Danupon Nanongkai, and Jun (Jim) Xu. Randomized multi-pass streaming skyline algorithms. *PVLDB*, 2(1):85–96, 2009. doi:10.14778/1687627.1687638.

12 Martin Farach-Colton, Meng Li, and Meng-Tsung Tsai. Streaming algorithms for planar convex hulls. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 47:1–47:13, 2018. doi:10.4230/LIPIcs.ISAAC.2018.47.

13 Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 58–66, 2001.

14 Xiaocheng Hu, Cheng Sheng, Yufei Tao, Yi Yang, and Shuigeng Zhou. Output-sensitive skyline algorithms in external memory. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 887–900, 2013.

15 J.H. Dula J and R.V. Helgason. A new procedure for identifying the frame of the convex hull of a finite collection of points in multidimensional space. *European Journal of Operational Research*, 92(2):352–367, 1996.

16 David G. Kirkpatrick and Raimund Seidel. Output-size sensitive algorithms for finding maximal vectors. In *Proceedings of Symposium on Computational Geometry (SoCG)*, pages 89–96, 1985. doi:10.1145/323233.323246.

17 David G. Kirkpatrick and Raimund Seidel. The ultimate planar convex hull algorithm? *SIAM Journal of Computing*, 15(1):287–299, 1986. doi:10.1137/0215021.

18 Ketan Mulmuley. *Computational Geometry: An Introduction through Randomized Algorithms*. Prentice Hall, 1994.

19 Thomas Ottmann, Sven Schuierer, and Subbiah Soundaralakshmi. Enumerating extreme points in higher dimensions. *Nordic Journal of Computing*, 8(2):179–192, 2001.

20 F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer–Verlag, 1985.

21 Raimund Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete & Computational Geometry*, 6:423–434, 1991. doi:10.1007/BF02574699.

22 Cheng Sheng and Yufei Tao. On finding skylines in external memory. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 107–116, 2011. doi:10.1145/1989284.1989298.

A A deterministic algorithm for skyline points in $\mathbb{R}^d$

In this appendix, we briefly describe a deterministic algorithm to compute skyline points in $\mathbb{R}^d$ using $O(h \log n)$ space and $O(\log^{d-1} n)$ passes (as we have mentioned in the introduction). Though we are not aware of an explicit reference of this result, it follows from a straightforward adaptation of Kirkpatrick and Seidel's output-sensitive divide-and-conquer algorithm [16], but reimplemented in the multi-pass setting (analogous to Chan and Chen's multi-pass reimplementation [5] of Kirkpatrick and Seidel's output-sensitive 2-d convex hull algorithm [17]).

Given point sets P and M in $\mathbb{R}^d$, let $P \ominus M$ denote the "filtered" subset of all points $p \in P$ that are not dominated by any points in M. Let $p\!\downarrow$ denote the projection of p onto the first $d-1$ coordinates, and let $P\!\downarrow = \{p\!\downarrow : p \in P\}$.

Below is a variant or reinterpretation of Kirkpatrick and Seidel's algorithm for computing the skyline of $P \ominus M$ (initially, we set $M = \emptyset$):

Algorithm 3 Skyline$_d(P, M)$.

```
1. If $|P \ominus M| \le 1$, then return $P \ominus M$.
2. Partition $P$ into the left and the right halves $P_\ell$ and $P_r$ using an approximate median $d$-th coordinate.
3. Compute $M_r = \textbf{Skyline}_{d-1}((P_r \ominus M)\!\downarrow, \emptyset)$. Add $\{p : p\!\downarrow \in M_r\}$ to $M$.
4. Return $\textbf{Skyline}_d(P_\ell, M) \cup \textbf{Skyline}_d(P_r, M) \cup \{p : p\!\downarrow \in M_r\}$.
```

(In the original algorithm, points dominated by M_r are pruned from P_ℓ before recursion. With the filtering operation $\ominus$, explicit pruning is avoided.)

In the multi-pass setting, we will execute the recursion level by level. The recursion tree for **Skyline**$_d$ has $O(\log n)$ levels. We maintain one global set M and do filtering with respect to this global set M (this does not affect correctness); the size of the set is $O(h)$. Consider the next level of the tree. There are at most $O(h)$ nodes in the level. Each subset P can be encoded by an interval in the d-th coordinate. As we make a pass over the input and encounter a point p, we can identify the subset P containing p, and test whether it is in $P \ominus M$ by checking whether it is dominated by any point in M (in $O(h)$ time naively, or in polylogarithmic time by storing M in an orthogonal range searching data structure). The approximate median computation in step 2 can be done by a known one-pass, $O(\log n)$-space algorithm of Greenwald and Khanna [13]. All $O(h)$ invocations to this approximate median algorithm are done simultaneously, and so the total space used is $O(h \log n)$. Step 3 invokes a $(d-1)$-dimensional skyline algorithm. Again, these invocations are done simultaneously; the total output size in these calls is $O(h)$.

Let $P_d(n)$ be the number of passes in our d-dimensional skyline algorithm, and let $s_d(n)$ be the space used per output point (i.e., the total space is $h \cdot s_d(n)$). Then

$$P_d(n) = O(\log n) \cdot (P_{d-1}(n) + O(1)) \qquad \text{and} \qquad s_d(n) = s_{d-1}(n) + O(\log n).$$

With the base case $P_1(n) = 1$ and $s_1(n) = O(1)$, we get $P_d(n) = O(\log^{d-1} n)$ and $s_d(n) = O(\log n)$ as desired.

One-Tape Turing Machine and Branching Program Lower Bounds for MCSP

Mahdi Cheraghchi
Department of EECS, University of Michigan, Ann Arbor, MI, USA

Shuichi Hirahara
National Institute of Informatics, Tokyo, Japan

Dimitrios Myrisiotis
Department of Computing, Imperial College London, London, UK

Yuichi Yoshida
National Institute of Informatics, Tokyo, Japan

Abstract

For a size parameter $s\colon \mathbb{N} \to \mathbb{N}$, the Minimum Circuit Size Problem (denoted by MCSP$[s(n)]$) is the problem of deciding whether the minimum circuit size of a given function $f\colon \{0,1\}^n \to \{0,1\}$ (represented by a string of length $N := 2^n$) is at most a threshold $s(n)$. A recent line of work exhibited “hardness magnification” phenomena for MCSP: A very weak lower bound for MCSP implies a breakthrough result in complexity theory. For example, McKay, Murray, and Williams (STOC 2019) implicitly showed that, for some constant $\mu_1 > 0$, if MCSP$[2^{\mu_1 \cdot n}]$ cannot be computed by a one-tape Turing machine (with an additional one-way read-only input tape) running in time $N^{1.01}$, then $\mathrm{P} \neq \mathrm{NP}$.

In this paper, we present the following new lower bounds against one-tape Turing machines and branching programs:

1. A randomized two-sided error one-tape Turing machine (with an additional one-way read-only input tape) cannot compute MCSP$[2^{\mu_2 \cdot n}]$ in time $N^{1.99}$, for some constant $\mu_2 > \mu_1$.
2. A non-deterministic (or parity) branching program of size $o(N^{1.5}/\log N)$ cannot compute MKTP, which is a time-bounded Kolmogorov complexity analogue of MCSP. This is shown by directly applying the Nečiporuk method to MKTP, which previously appeared to be difficult.
3. The size of any non-deterministic, co-non-deterministic, or parity branching program computing MCSP is at least $N^{1.5-o(1)}$.

These results are the first non-trivial lower bounds for MCSP and MKTP against one-tape Turing machines and non-deterministic branching programs, and essentially match the best-known lower bounds for any explicit functions against these computational models.

The first result is based on recent constructions of pseudorandom generators for read-once oblivious branching programs (ROBPs) and combinatorial rectangles (Forbes and Kelley, FOCS 2018; Viola 2019). En route, we obtain several related results:

1. There exists a (local) hitting set generator with seed length $\widetilde{O}(\sqrt{N})$ secure against read-once polynomial-size non-deterministic branching programs on N-bit inputs.
2. Any read-once co-non-deterministic branching program computing MCSP must have size at least $2^{\widetilde{\Omega}(N)}$.

2012 ACM Subject Classification Theory of computation → Circuit complexity; Theory of computation → Pseudorandomness and derandomization

Keywords and phrases Minimum Circuit Size Problem, Kolmogorov Complexity, One-Tape Turing Machines, Branching Programs, Lower Bounds, Pseudorandom Generators, Hitting Set Generators

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.23

Related Version *Full Version*: https://eccc.weizmann.ac.il/report/2020/103/

Funding *Mahdi Cheraghchi*: M. Cheraghchi’s research is supported in part by the NSF award CCF-2006455.

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 23; pp. 23:1–23:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements We would like to express our gratitude to Emanuele Viola and Osamu Watanabe for bringing to our attention the works by Kalyanasundaram and Schnitger [26] and Watanabe [39], respectively, and for helpful discussions. In particular, we thank Emanuele Viola for explaining to us his works [15, 38]. We thank Rahul Santhanam for pointing out that Nečiporuk's method can be applied to not only MKtP but also MKTP. We thank Chin Ho Lee for answering our questions regarding his work [27]. We thank Paul Beame for bringing his work [6] to our attention. We thank Valentine Kabanets, Zhenjian Lu, Igor C. Oliveira, and Ninad Rajgopal for illuminating discussions. Finally, we would like to thank the anonymous reviewers for their constructive feedback.

1 Introduction

The Minimum Circuit Size Problem (MCSP) asks whether a given Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ can be computed by some Boolean circuit of size at most a given threshold s. Here the function f is represented by the truth table of f, i.e., the string of length $N := 2^n$ that is obtained by concatenating all the outputs of f. For a size parameter $s\colon \mathbb{N} \to \mathbb{N}$, its parameterized version is denoted by MCSP$[s]$: That is, MCSP$[s]$ asks if the minimum circuit size of a function $f\colon \{0,1\}^n \to \{0,1\}$ is at most $s(n)$.

MCSP is one of the most fundamental problems in complexity theory, because of its connection to various research areas, such as circuit complexity [35, 25, 22, 31, 21, 2], learning theory [8], and cryptography [35, 16, 18]. It is easy to see that MCSP $\in \mathsf{NP}$ because, given a circuit C of size s as an NP certificate, one can check whether C computes the given function f in time $N^{O(1)}$. On the other hand, its NP-completeness is a long-standing open question, which dates back to the introduction of the theory of NP-completeness (cf. [4]), and it has an application to the equivalence between the worst-case and average-case complexity of NP (cf. [18]).

Recently, a line of work exhibited surprising connections between very weak lower bounds of MCSP and important open questions of complexity theory, informally termed as "hardness magnification" phenomena. Oliveira and Santhanam [34] (later with Pich [33]) showed that, if an approximation version of MCSP cannot be computed by a circuit of size $N^{1.01}$, then $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ (in particular, $\mathsf{P} \neq \mathsf{NP}$ follows). Similarly, McKay, Murray, and Williams [30] showed that, if MCSP$[s(n)]$ cannot be computed by a 1-pass streaming algorithm of poly$(s(n))$ space and poly$(s(n))$ update time, then $\mathsf{P} \neq \mathsf{NP}$. Therefore, in order to obtain a breakthrough result, it is sufficient to obtain a very weak lower bound for MCSP.

Are hardness magnification phenomena plausible approaches for resolving the P versus NP question? We do not know the answer yet. However, it should be noted that, as argued in [3, 34], hardness magnification phenomena appear to bypass the *natural proof* barrier of Razborov and Rudich [35], which is one of the major barriers of complexity theory for resolving the P versus NP question. Most of lower bound proof techniques of complexity theory are "natural" in the following sense: Given a lower bound proof for a circuit class $\mathfrak{C}$, one can interpret it as an efficient average-case algorithm for solving $\mathfrak{C}$-MCSP (i.e., one can efficiently decide whether a given Boolean function f can be computed by a small $\mathfrak{C}$-circuit when the input f is chosen uniformly at random; cf. Hirahara and Santhanam [20]). Razborov and Rudich [35] showed that such a "natural proof" technique is unlikely to resolve $\mathsf{NP} \not\subseteq \mathsf{P/poly}$; thus we need to develop fundamentally new proof techniques. There seems to be no simple argument that naturalizes proof techniques of hardness magnification phenomena; hence, investigating hardness magnification phenomena could lead us to a new non-natural proof technique.

1.1 Our results

1.1.1 Lower bounds against one-tape Turing machines

Motivated by hardness magnification phenomena, we study the time required to compute MCSP by using a one-tape Turing machine. We first observe that the hardness magnification phenomena of [30] imply that a barely superlinear time lower bound for a one-tape Turing machine is sufficient for resolving the P versus NP question.

▶ **Theorem 1** (A corollary of McKay, Murray, and Williams [30]; see the full version). *There exists a small constant $\mu > 0$ such that if $\mathrm{MCSP}[2^{\mu \cdot n}] \notin \mathsf{DTIME}_1\left[N^{1.01}\right]$, then $\mathsf{P} \neq \mathsf{NP}$.*

Here, we denote by $\mathsf{DTIME}_1[t(N)]$ the class of languages that can be computed by a Turing machine equipped with a one-way read-only input tape and a two-way read/write work tape running in time $O(t(N))$ on inputs of length N. We note that it is rather counter-intuitive that there is a *universal* constant $\mu > 0$; it is instructive to state Theorem 1 in the following logically equivalent way: If $\mathrm{MCSP}[2^{\mu \cdot n}] \notin \mathsf{DTIME}_1\left[N^{1.01}\right]$ for *all constants* $\mu > 0$, then $\mathsf{P} \neq \mathsf{NP}$.[1]

One of our main results is a nearly quadratic lower bound on the time complexity of a *randomized* one-tape Turing machine (with one additional read-only one-way input tape) computing MCSP.

▶ **Theorem 2.** *There exists some constant $0 < \mu < 1$ such that $\mathrm{MCSP}[2^{\mu \cdot n}]$ is not in $\mathsf{BPTIME}_1\left[N^{1.99}\right]$.*

Here, $\mathsf{BPTIME}_1[t(N)]$ denotes the class of languages that can be computed by a *two-sided-error randomized* Turing machine equipped with a one-way read-only input tape and a two-way read/write work tape running in time $t(N)$ on inputs of length N; we say that a two-sided-error randomized algorithm *computes* a problem if it outputs a correct answer with high probability (say, with probability at least 2/3) over the internal randomness of the algorithm.

Previously, no non-trivial lower bound on the time complexity required for computing MCSP by a Turing machine was known. Moreover, Theorem 2 essentially matches the best-known lower bound for this computational model; namely, the lower bound due to Kalyanasundaram and Schnitger [26], who showed that Element Distinctness is not in $\mathsf{BPTIME}_1\left[o\left(N^2/\log N\right)\right]$.

Our lower bound against $\mathsf{BPTIME}_1\left[N^{1.99}\right]$ is much stronger than the required lower bound (i.e, $\mathsf{DTIME}_1\left[N^{1.01}\right]$) of the hardness magnification phenomenon of Theorem 1. However, Theorem 2 falls short of the hypothesis of the hardness magnification phenomenon of Theorem 1 because of the choice of the size parameter. In the hardness magnification phenomenon, we need to choose the size parameter to be $2^{\mu \cdot n}$ for some small constant $\mu > 0$, whereas, in our lower bound, we will choose μ to be some constant close to 1. That is, what is missing for proving $\mathsf{P} \neq \mathsf{NP}$ is to decrease the size parameter from $2^{(1-o(1))\cdot n}$ to $2^{o(n)}$ in Theorem 2, or to increase the size parameter from $2^{o(n)}$ to $2^{(1-o(1))\cdot n}$ in Theorem 1.

Next, we investigate the question of whether hardness magnification phenomena on $\mathrm{MCSP}[s(n)]$ such as Theorem 1 can be proved when the size parameter $s(n)$ is large, as posed by Chen, Jin, and Williams [10]. As observed in [9], most existing proof techniques on hardness magnification phenomena are shown by constructing an oracle algorithm which makes short queries to some oracle. For example, behind the hardness magnification

[1] Observe that $\exists \mu, (P(\mu) \Rightarrow Q)$ is logically equivalent to $\exists \mu, (\neg P(\mu) \vee Q)$, which is equivalent to $\neg(\forall \mu, P(\mu)) \vee Q$.

phenomena of Theorem 1 is a nearly-linear-time oracle algorithm that solves $\mathrm{MCSP}[2^{o(n)}]$ by making queries of length $2^{o(n)}$ to some PH oracle (see Corollary 18 for a formal statement). Chen, Hirahara, Oliveira, Pich, Rajgopal, and Santhanam [9] showed that most lower bound proof techniques can be generalized to such an oracle algorithm, thereby explaining the difficulty of combining hardness magnification phenomena with lower bound proof techniques. Following [9], we observe that our lower bound (Theorem 3) can be generalized to a lower bound against an oracle algorithm which makes short queries.

▶ **Theorem 3.** *Let $O \subseteq \{0,1\}^*$ be any oracle. Then, for every constant $1/2 < \mu < 1$, $\mathrm{MCSP}[2^{\mu \cdot n}]$ on truth tables of size $N := 2^n$ is not in $\mathsf{BPTIME}_1^O\left[N^{1+\mu'}\right]$ for some constant $\mu' > 0$, where all of the strings queried to O are of length $N^{o(1)}$.*

Theorem 3 can be seen as a partial answer to the question posed by [10]: It is impossible to extend the hardness magnification phenomena of Theorem 1 to $\mathrm{MCSP}[2^{\mu n}]$ for $\mu > 1/2$ by using similar techniques used in [30]. Recall that the proof techniques behind [30] are to construct a nearly-linear-time oracle algorithm that solves $\mathrm{MCSP}[2^{\mu n}]$ by making short queries to some oracle; the existence of such an oracle algorithm is ruled out by Theorem 3 when $\mu > 1/2$. Therefore, in order to obtain a hardness magnification phenomenon for $\mathrm{MCSP}[2^{0.51n}]$, one needs to develop a completely different proof technique that does not rely on constructing an oracle algorithm that makes short queries.

1.1.2 Lower bounds against branching programs

Another main result of this work is a lower bound against non-deterministic branching programs. We make use of *Nečiporuk's method*, which is a standard proof technique for proving a lower bound against branching programs. However, it appeared previously that Nečiporuk's method is not directly applicable to the problems such as MCSP [20]. In this paper, we develop a new proof technique for applying Nečiporuk's method to a variant of MCSP, called MKTP. MKTP is the problem of deciding whether $\mathrm{KT}(x) \leq s$ given (x, s) as input. Here $\mathrm{KT}(x)$ is defined as the minimum, over all programs M and integers t, of $|M| + t$ such that, for every i, M outputs the i-th bit of x in time t given an index i as input [1]. We prove lower bounds against general branching programs and non-deterministic branching programs by using Nečiporuk's method.

▶ **Theorem 4.** *The size of a branching program computing MKTP is at least $\Omega(N^2/\log^2 N)$. The size of a non-deterministic branching program or a parity branching program computing MKTP is at least $\Omega(N^{1.5}/\log N)$.*

Theorem 4 gives the first non-trivial lower bounds against non-deterministic and parity branching programs for MKTP and, in addition, these are the best lower bounds which can be obtained by using Nečiporuk's method (cf. [6]). Previously, by using a pseudorandom generator for branching programs constructed by [23], it was shown in [33, 11] that (deterministic) branching programs requires $N^{2-o(1)}$ size to compute MCSP and MKTP.[2] However, it is not known whether there is a pseudorandom generator for non-deterministic or parity branching programs. As a consequence, no non-trivial lower bound for MKTP (nor its exponential-time version denoted by MKtP) against these models was known before. Surprisingly, Theorem 4 is proved without using a pseudorandom generator nor a weaker

[2] It is worthy of note that Theorem 4 mildly improves the lower bounds of [33, 11] to $\Omega(N^2/\log^2 N)$ by directly applying Nečiporuk's method, which matches the state-of-the-art lower bound for any explicit function up to a constant factor.

object called a hitting set generator. We emphasize that it is surprising that a lower bound for MKtP can be obtained without using a hitting set generator; indeed, the complexity of MKtP is closely related to a hitting set generator, and in many settings (especially when the computational model is capable of computing XOR), a lower bound for MKtP and the existence of a hitting set generator are equivalent [18, 19].

The proof technique of Theorem 4 is applicable to problems of computing various resource-bounded Kolmogorov complexity measures, such as MKtP. However, we fail to apply Nečiporuk's method to MCSP, despite that circuit complexity can also be regarded as a version of resource-bounded Kolmogorov complexity. The KT-complexity of the truth table of a function f and the minimum circuit size of f are polynomially related to each other [1]; unfortunately, the relationship between circuit complexity and KT-complexity is not tight enough for our argument to work. Nevertheless, we were able to use a different approach to present the first non-trivial lower bound for MCSP against non-deterministic branching programs.

▶ **Theorem 5.** *The size of any non-deterministic, co-non-deterministic, or parity branching program computing* MCSP *is at least* $N^{1.5-o(1)}$.

The proof of Theorem 5 is based on a pseudorandom generator construction of Impagliazzo, Meka, and Zuckerman [23]. We show that their construction actually provides a pseudorandom generator of seed length $s^{2/3+o(1)}$ that fools non-deterministic, co-non-deterministic, and parity branching programs of size s.

Along the way, we obtain several new results regarding a lower bound for MCSP and a hitting set generator. A *hitting set generator* (HSG) $H\colon \{0,1\}^{\lambda(N)} \to \{0,1\}^N$ for a circuit class $\mathfrak{C}$ is a function such that, for any circuit C from $\mathfrak{C}$ that accepts at least $(1/2)\cdot 2^N$ strings of length N, there exists some seed $z \in \{0,1\}^{\lambda(N)}$ such that C accepts $H(z)$. We present a hitting set generator secure against read-once non-deterministic branching programs, based on a pseudorandom generator constructed by Forbes and Kelley [13].

▶ **Theorem 6.** *There exists an explicit construction of a (local) hitting set generator* $H\colon \{0,1\}^{\widetilde{O}\left(\sqrt{N\cdot\log s}\right)} \to \{0,1\}^N$ *for read-once non-deterministic branching programs of size* s.

Previously, Andreev, Baskakov, Clementi, and Rolim [5] constructed a hitting set generator with non-trivial seed length for read-k-times non-deterministic branching programs, but their seed length is as large as $N - o(N)$. Theorem 6 improves the seed length to $\widetilde{O}(\sqrt{N\cdot\log s})$. As an immediate corollary, we obtain a lower bound for MCSP against read-once non-deterministic branching programs.

▶ **Corollary 7.** *Any read-once co-non-deterministic branching program that computes* MCSP *must have size at least* $2^{\widetilde{\Omega}(N)}$.

1.2 Our techniques

1.2.1 Local HSGs for MCSP lower bounds

For a circuit class $\mathfrak{C}$, a general approach for obtaining a $\mathfrak{C}$-lower bound for MCSP is by constructing a "local" hitting set generator (or a pseudorandom generator (PRG), which is a stronger notion) secure against $\mathfrak{C}$. Here, we say that a function $G\colon \{0,1\}^s \to \{0,1\}^N$ is *local* if, for every z, the ith bit of $G(z)$ is "easy to compute" from the index i; more precisely, for every seed z, there exists some circuit C of size at most s such that C outputs the ith bit of $G(z)$ on input $i \in [N]$. Note here that $G(z)$ is a YES instance of MCSP$[s]$, whereas a string w

chosen uniformly at random is a NO instance of MCSP[s] with high probability. This means that any $\mathfrak{C}$-algorithm that computes MCSP[s] distinguishes the pseudorandom distribution $G(z)$ from the uniform distribution w, and hence the existence of $\mathfrak{C}$-algorithm for MCSP[s] implies that there exists no local hitting set generator secure against $\mathfrak{C}$. This approach has been used in several previous works, e.g., [35, 1, 20, 11]. In fact, it is worthy of note that, in some sense, this is *the only approach* – at least for a general polynomial-size circuit class $\mathfrak{C} = \mathsf{P/poly}$, because Hirahara [18] showed that a lower bound for an approximation version of MCSP is equivalent to the existence of a local HSG.

At the core of our results is the recent breakthrough result of Forbes and Kelley [13], who constructed the first pseudorandom generator with $\mathsf{polylog}(n)$ seed length that fools unknown-order read-once oblivious branching programs. Viola [38] used their construction to obtain a pseudorandom generator that fools *deterministic* Turing machines (DTMs). Herein, we generalize his result to the case of *randomized* Turing machine (RTMs), and the case of *two-sided-error* randomized Turing machine ($\mathsf{BPTIME}_1[t(N)]$).[3] At a high level, our crucial idea is that Viola's proof does not exploit the uniformity of Turing machines, and hence a good coin flip sequence of a randomized oracle algorithm and all of its [small enough] oracle queries and corresponding answers can be fixed as non-uniformity (Lemma 22). In addition, by a careful examination of the Forbes-Kelley PRG, we show that their PRG is local; this gives rise to a local PRG that fools $\mathsf{BPTIME}_1[t(N)]$, which will complete a proof of our main result (Theorem 3).

We note that the proof above implicitly shows an exponential-size lower bound for MCSP against read-once oblivious branching programs, which was previously not known. Corollary 7 generalizes this lower bound to the case of co-non-deterministic read-once (not necessarily oblivious) branching program. In order to prove this, we make use of PRGs that fool combinatorial rectangles (e.g., [13, 27]). We present a general transformation from a PRG for combinatorial rectangles into a HSG for non-deterministic read-once branching program, by using the proof technique of Borodin, Razborov, and Smolensky [7]; see Theorem 6.

1.2.2 Nečiporuk's method for MKTP lower bounds

In order to apply Nečiporuk's method to MKTP, we need to give a lower bound on the number of distinct subfunctions that can be obtained by fixing all but $O(\log n)$ bits.

The idea of counting distinct subfunctions of MKTP is to show that a random restriction which leaves $O(\log n)$ variables free induces different subfunctions with high probability. Specifically, partition the input variables $[n]$ into $m := n/O(\log n)$ blocks, pick $m-1$ strings $\rho := \rho_2 \cdots \rho_m \in (\{0,1\}^{O(\log n)})^{m-1}$ randomly, and consider the restricted function $f{\restriction}_\rho(\rho_1) := \mathrm{MKTP}(\rho_1\rho_2\cdots\rho_m, \theta)$ for some threshold function θ to be chosen later. Then, the string $\rho_i\rho_2\cdots\rho_m$ is compressible when $i \in \{2, \cdots, m\}$ whereas the string $\rho_1\rho_2\cdots\rho_m$ is not compressible when ρ_1 is chosen randomly. This holds as, in the former case, there exists a $k \in \{2, \dots, m\}$ such that $\rho_i = \rho_k$ and this yields a description for the string $\rho_i\rho_2\cdots\rho_m$ that is shorter than most of its descriptions in the latter case. Let now θ be an upper bound on the KT complexity of $\rho_i\rho_2\cdots\rho_m$ in the case where $i \in \{2, \cdots, m\}$. Therefore, $f{\restriction}_\rho(\rho_i) = 1$ for any ρ and $i \in \{2, \cdots, m\}$, and $f{\restriction}_\rho(\rho_1) = 0$ with high probability over random ρ and ρ_1. This implies that, with high probability over the random restrictions ρ and ρ', it is the case that $f{\restriction}_\rho \not\equiv f{\restriction}_{\rho'}$. This is so as, for every $i \in \{2, \dots, m\}$, the probability over the random

[3] We emphasize that the notion of PRGs secure against these three computational models is different. See Definition 11, Definition 13, and Lemma 15.

restrictions ρ and ρ' that the string ρ_i is such that $f{\restriction}_{\rho'}(\rho_i) = f{\restriction}_{\rho}(\rho_i)$ is small, by the fact that $f{\restriction}_{\rho}(\rho_i) = 1$ for any ρ and the fact that $f{\restriction}_{\rho'}(\rho_i) = 0$ with high probability over random ρ_i and ρ' [and therefore with high probability over random ρ and ρ' as well].

Unfortunately, the probability that $f{\restriction}_{\rho} \equiv f{\restriction}_{\rho'}$ holds may not be exponentially small. As a consequence, a lower bound on the number of distinct subfunctions that can be directly obtained from this fact may not be exponential. In contrast, we need to prove an exponential lower bound on the number of distinct subfunctions in order to obtain the state-of-the-art lower bound via Nečiporuk's method.

In order to make the argument work, we exploit symmetry of information for (resource-unbounded) Kolmogorov complexity and Kolmogorov-randomness. Instead of picking ρ and ρ' randomly, we keep a set P which contains restrictions ρ that induce distinct subfunctions. Starting from $P := \emptyset$, we add one Kolmogorov-random restriction ρ to P so that the property of P is preserved. By using symmetry of information for Kolmogorov complexity, we can argue that one can add a restriction to P until P becomes as large as $2^{\Omega(n)}$, which proves that the number of distinct subfunctions of MKTP is exponentially large. Details can be found in Section 4.

1.3 Related work

Chen, Jin, and Williams [10] generalized hardness magnification phenomena to arbitrary sparse languages in NP. Note that MCSP$[2^{\mu n}]$ is a *sparse* language in the sense that the number of YES instances of MCSP$[2^{\mu n}]$ is at most $2^{\widetilde{O}(2^{\mu n})}$, which is much smaller than the number 2^{2^n} of all the instances of length 2^n. Hirahara [19] proved that a super-linear-size lower bound on co-non-deterministic branching programs for computing an approximation and space-bounded variant of MKtP implies the existence of a hitting set generator secure against read-once branching programs (and, in particular, $\mathsf{RL} = \mathsf{L}$).

Regarding unconditional lower bounds for MCSP, Razborov and Rudich [35] showed that there exists no AC^0-natural property useful against $\mathsf{AC}^0[\oplus]$, which in particular implies that MCSP $\notin \mathsf{AC}^0$; otherwise, the complement of MCSP would yield an AC^0-natural property useful against $\mathsf{P/poly} \supseteq \mathsf{AC}^0[\oplus]$. Hirahara and Santhanam [20] proved that MCSP essentially requires quadratic-size de Morgan formulas. Cheraghchi, Kabanets, Lu, and Myrisiotis [11] proved that MCSP essentially requires cubic-size de Morgan formulas as well as quadratic-size (general, unconstrained) branching programs. Golovnev, Ilango, Impagliazzo, Kabanets, Kolokolova, and Tal [14] proved that, for any prime p, MCSP requires constant-depth circuits, that are augmented with MOD$_p$ gates, of weakly-exponential size.

The state-of-the-art time lower bound against DTMs on inputs of size n is $\Omega(n^2)$, proved by Maass [28], for the Polydromes function (which is a generalization of Palindromes). Regarding the case when the considered DTMs have a two-way read-only input tape, Maass and Schorr [29] proved that there is some problem in $\Sigma_2\mathsf{TIME}[n]$ that requires $\Omega(n^{3/2}/\log^6 n)$ time to compute on such machines. As mentioned earlier, in Section 1.1, the state-of-the-art time lower bound against RTMs is due to Kalyanasundaram and Schnitger [26], who showed that Element Distinctness is not in $\mathsf{BPTIME}_1\left[o(N^2/\log N)\right]$.

Viola [38] gave a PRG that fools RTMs that run in time $n^{1+\Omega(1)}$; this also yields a $n^{1+\Omega(1)}$ time lower bound against such machines. To do this, Viola extended prior work [29, 37] on simulating any RTM by a sum of ROBPs [see Lemma 20] and then employed the PRG by Haramaty, Lee, and Viola [15] that fools ROBPs;[4] it is a straightforward observation [38],

[4] It should be noted that before Haramaty, Lee, and Viola [15] and Viola [38], the problem of designing PRGs of polynomial stretch that fool RTMs was wide open despite intense research efforts.

then, that the Forbes-Kelley PRG [13] [which appeared afterwards and was inspired by the PRG by Haramaty, Lee, and Viola] yields a PRG of nearly quadratic stretch that fools RTMs and, therefore, a nearly quadratic lower bound against the same model as well. Moreover, Viola [38] showed that there exists some problem in $\Sigma_3\mathsf{TIME}[n]$ that requires $n^{1+\Omega(1)}$ time to compute on any RTM that has the extra feature of a two-way read-only input tape; one of the ingredients of this result, is again the PRG by Haramaty, Lee, and Viola [15].

For the case of one-tape TMs with no extra tapes, Hennie [17] proved in 1965 that the Palindromes function requires $\Omega(n^2)$ time to compute. Van Melkebeek and Raz [37] observed fixed-polynomial time lower bounds for SAT against non-deterministic TMs with a d-dimensional read/write two-way work tape and a random access read-only input tape; these lower bounds depend on d.

1.4 Organization

In Section 2, we give the necessary background. We prove Theorem 3 in Section 3, and Theorem 4 in Section 4. The proofs of the rest of our results appear in the full version.

2 Preliminaries

2.1 Circuit complexity

Let $f : \{0,1\}^n \to \{0,1\}$. We define the *circuit complexity of* f, denoted by $\mathrm{CC}(f)$, to be equal to the size (i.e., the number of gates) of the smallest bounded fan-in unbounded fan-out Boolean circuit, over the $\{\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}\} = \{\wedge, \vee, \neg\}$ basis, that, on input x, outputs $f(x)$. For a string $y \in \{0,1\}^{2^n}$, we denote by $\mathrm{CC}(y)$ the circuit complexity of the function $f_y : \{0,1\}^n \to \{0,1\}$ encoded by y; i.e., $f_y(x) = y_x$, for any $x \in \{0,1\}^n$.

A standard counting argument shows that a random function attains nearly maximum circuit complexity with high probability.

▶ **Proposition 8** ([36]). *For any function $s\colon \mathbb{N} \to \mathbb{N}$ with $s(n) = o(2^n/n)$, it holds that*

$$\mathop{\mathbf{Pr}}_{x \sim \{0,1\}^{2^n}}[\mathrm{CC}(x) \le s(n)] = o(1),$$

for all large $n \in \mathbb{N}$.

▶ **Definition 9** (Minimum Circuit Size Problem [25]). *We define* MCSP *as*

$$\mathrm{MCSP} := \left\{(x, \theta) \in \{0,1\}^{2^n} \times \{0,1\}^n \mid \mathrm{CC}(x) \le \theta\right\}_{n \in \mathbb{N}},$$

and its parameterized version *as*

$$\mathrm{MCSP}[s(n)] := \left\{x \in \{0,1\}^{2^n} \mid \mathrm{CC}(x) \le s(n)\right\}_{n \in \mathbb{N}},$$

for a size parameter $s\colon \mathbb{N} \to \mathbb{N}$.

2.2 Turing machines

Throughout this paper, we consider a Turing machine that has one work tape and a one-way input tape. In this context, "one-way" means that the tape-head may move only from left to right.

A *deterministic Turing machine (DTM)* is a Turing machine with two tapes: A two-way read/write work tape and a one-way read-only input tape. Let $x \in \{0,1\}^*$ and M be a DTM; we write $M(x)$ to denote the output of M when its input tape is initialized with x and its work tape is empty. Let $t\colon \mathbb{N} \to \mathbb{N}$ be time-constructible. The class of languages $L \subseteq \{0,1\}^*$ decided by some $O(1)$-state time-t DTM is denoted by $\mathsf{DTIME}_1[t]$.

We also consider a randomized variant of DTMs. A *randomized Turing machine (RTM)* is a Turing machine with three tapes: A two-way read/write work tape, a one-way read-only input tape, and a one-way read-only random tape. Let $x, r \in \{0,1\}^*$ and M be a RTM; we write $M(x,r)$ to denote the output of M when its input tape contains x, its work tape is empty, and its random tape contains r. Let $t\colon \mathbb{N} \to \mathbb{N}$ be time-constructible. For a language $L \subseteq \{0,1\}^*$ and a RTM M, we say that M *decides* L *with two-sided error* if $\Pr_r[M(x,r) = 1] \geq \frac{2}{3}$ for every input $x \in L$ and $\Pr_r[M(x,r) = 0] \geq \frac{2}{3}$ for every input $x \notin L$. The class of languages $L \subseteq \{0,1\}^*$ decided by some $O(1)$-state time-t RTM with two-sided error is denoted by $\mathsf{BPTIME}_1[t]$.

A *randomized oracle Turing machine (oracle RTM)* is a Turing machine with four tapes: A two-way read/write work tape, a one-way read-only input tape, a one-way read-only random tape, and an oracle tape. This model is identical to the randomized Turing machine model apart from the oracle tape, which is a standard oracle tape. The class of languages $L \subseteq \{0,1\}^*$ decided by some $O(1)$-state time-t oracle RTM, with access to some oracle $O \subseteq \{0,1\}^*$, with two-sided error is denoted by $\mathsf{BPTIME}_1^O[t]$.

2.3 Streaming algorithms

A *space-$s(n)$ streaming algorithm* with update time $u(n)$ on an input $x \in \{0,1\}^n$ has a working storage of $s(n)$ bits. At any point the algorithm can either choose to perform one operation on $O(1)$ bits in storage or it can choose to read the next bit from the input. The total time between two next-bit reads is at most $u(n)$ and the final outcome is reported in $O(u(n))$ time.

▶ **Lemma 10.** *Any one-pass streaming algorithm with $t(N)$ update time, on inputs of length N, can be simulated by a one-tape Turing machine with a one-way read-only input tape running in time $O(N \cdot \mathrm{poly}(t(N)))$.*

Proof. Recall that a streaming algorithm reads one bit of its input from left to right, and each consecutive read operation occurs within $t(N)$ time steps. Thus, it takes $N \cdot \mathrm{poly}(t(N))$ time-steps in total to finish the computation on inputs of length N in the standard multi-tape Turing machine model, as the size of the input is N and $\mathrm{poly}(t(N))$ time-steps suffice for some multi-tape Turing machine to perform an update [12]. For any time constructible function $T : \mathbb{N} \to \mathbb{N}$, a one-tape Turing machine can simulate a $T(n)$-time multi-tape Turing machine within $O(T(n)^2)$ steps. Thus, a streaming algorithm can be simulated in time $N \cdot (\mathrm{poly}(t(N)))^2 = N \cdot \mathrm{poly}(t(N))$ by a one-tape Turing machine. ◀

2.4 Branching programs

A *branching program (BP)* is a directed acyclic graph with three special vertices: a start vertex s (the *source*) and two finish vertices, namely an accepting vertex h_1 and a rejecting vertex h_0 (the *sinks*).

On input $x \in \{0,1\}^n$, the computation starts at s and follows a directed path from s to some h_b, with $b \in \{0,1\}$. On this occasion, the output of the computation is b. In each step, the computation queries some input x_i, for $i \in [n]$, and then visits some other node, depending on the value of the variable just queried, namely 0 or 1, through an edge with label "$x_i = 0$" or "$x_i = 1$," respectively.

A branching program P *decides a language* $L \subseteq \{0,1\}^*$ in the natural way, i.e., $x \in L$ if and only if, on input x, the computation path that P follows starts at s and finishes at h_1. If the branching program is layered and the variable queried within each layer is the same, then the branching program is called *oblivious*. If the branching program queries each variable at most once, then the branching program is called a *read-once branching program (ROBP)*. If the branching program is oblivious and always queries the variables in some known order, where it is known beforehand which variable is queried at each layer, then the branching program is called *known-order*, else it is called *unknown-order*.

A branching program is called *non-deterministic* if some of its vertices have an arbitrary number of outgoing edges (i.e., if this number is not 2) or if some of its vertices have edges that do not refer to the same input variable. Non-deterministic branching programs may also have *unlabelled* edges, as well. Due to the nature of a non-deterministic branching program, it is possible that a computation never reaches either h_0 or h_1 as there can be some node with edges that their labels are all false according to the input at hand; in this case, we assume that the computation halts in a rejecting state.

A *non-deterministic branching program computes a function* $f\colon \{0,1\}^n \to \{0,1\}$ if, for every $x \in \{0,1\}^n$ such that $f(x) = 1$, there is some s-h_1 path and for every $x \in \{0,1\}^n$ such that $f(x) = 0$, all computations end in a rejecting state.

A *co-non-deterministic branching program computes a function* $f\colon \{0,1\}^n \to \{0,1\}$ if, for every $x \in \{0,1\}^n$ such that $f(x) = 1$, all source-to-sink paths are s-h_1 paths and for every $x \in \{0,1\}^n$ such that $f(x) = 0$, there exists some rejecting computation.

A *parity branching program* is a branching program that has counting semantics. That is, a parity branching program computes a function $f\colon \{0,1\}^n \to \{0,1\}$ if, for every $x \in \{0,1\}^n$ such that $f(x) = 1$, there is an odd number of s-h_1 paths and for every $x \in \{0,1\}^n$ such that $f(x) = 0$, there is an even number of s-h_1 paths.

We define the *size* of a branching program to be the number of its labelled edges.

2.5 Pseudorandom generators and hitting set generators

We recall the standard notions of pseudorandom generators and hitting set generators.

► **Definition 11.** *Let* $s\colon \mathbb{N} \to \mathbb{N}$ *be a function,* $\mathfrak{C}$ *be a circuit class, and* $0 < \varepsilon < 1$. *A* pseudorandom generator (PRG) that ε-fools $\mathfrak{C}$ *is a function* $G\colon \{0,1\}^{s(n)} \to \{0,1\}^n$ *such that*

$$\left| \underset{x\sim\{0,1\}^n}{\mathbf{Exp}}[f(x)] - \underset{y\sim\{0,1\}^{s(n)}}{\mathbf{Exp}}[f(G(y))] \right| \le \varepsilon,$$

for any circuit $C \in \mathfrak{C}$. *The value* $s(n)$ *is referred to as the* seed length *of* G.

► **Definition 12.** *Let* $s\colon \mathbb{N} \to \mathbb{N}$ *be a function,* $\mathfrak{C}$ *be a circuit class, and* $0 < \varepsilon < 1$. *A* hitting set generator (HSG) ε-secure against $\mathfrak{C}$ *is a function* $G\colon \{0,1\}^{s(n)} \to \{0,1\}^n$ *such that*

$$\underset{x\sim\{0,1\}^n}{\mathbf{Pr}}[C(x) = 1] \ge \varepsilon \implies C(H(y)) = 1 \textit{ for some } y \in \{0,1\}^{s(n)},$$

for any circuit $C \in \mathfrak{C}$. *By default, we choose* $\varepsilon := 1/2$.

For our purpose, it is useful to extend the notion of PRG to a pseudorandom generator that fools *randomized* algorithms.

▶ **Definition 13.** *For a function $s\colon \mathbb{N} \to \mathbb{N}$ and a parameter $0 < \varepsilon < 1$, a function $G\colon \{0,1\}^{s(n)} \to \{0,1\}^n$ is said to be a pseudorandom generator that* ε-fools q-state time-t RTMs *if*

$$\left| \mathop{\mathbf{Exp}}_{\substack{x\sim\{0,1\}^n,\\ r\sim\{0,1\}^t}}[M(x,r)] - \mathop{\mathbf{Exp}}_{\substack{y\sim\{0,1\}^{s(n)},\\ r\sim\{0,1\}^t}}[M(G(y),r)] \right| \le \varepsilon,$$

for any q-state time-t RTM M.

2.6 MCSP lower bounds from local HSGs

For a function $G\colon \{0,1\}^s \to \{0,1\}^n$, we say that G is *local* [11] if $\mathrm{CC}(G(z)) \le s$ for every string $z \in \{0,1\}^s$. We make use of the following standard fact.

▶ **Lemma 14.** *Let $s\colon \mathbb{N} \to \mathbb{N}$ be a function such that $s(n) = o(2^n/n)$, and $N := 2^n$. Suppose that there exists a local hitting set generator $H\colon \{0,1\}^{s(n)} \to \{0,1\}^N$ for a circuit class $\mathfrak{C}$. Then,* $\mathrm{MCSP}[s(n)] \notin \mathsf{co}\mathfrak{C}$.

Proof. We prove the contrapositive. Let $C \in \mathsf{co}\mathfrak{C}$ be a circuit that computes $\mathrm{MCSP}[s(n)]$. Since $\mathrm{CC}(H(z)) \le s(n)$, we have $H(z) \in \mathrm{MCSP}[s(n)]$; thus $C(H(z)) = 1$, for every $z \in \{0,1\}^{s(n)}$. For a random $w \sim \{0,1\}^N$, it follows from Proposition 8 that $w \notin \mathrm{MCSP}[s(n)]$ with probability $1 - o(1)$; hence $C(w) = 0$ for most w. Therefore, $\neg C \in \mathfrak{C}$ accepts at least a half of $\{0,1\}^N$ but rejects every string in the range of H, which contradicts the security of the hitting set generator H. ◀

We observe that a local pseudorandom generator for time-t RTMs also "fools" $\mathsf{BPTIME}_1[t(N)]$ in the following sense.

▶ **Lemma 15.** *Let $s, t\colon \mathbb{N} \to \mathbb{N}$ be functions, such that $s(n) = o(2^n/n)$, and $N := 2^n$. Suppose that there is a family of local pseudorandom generators $G = \{G_n : \{0,1\}^{s(n)} \to \{0,1\}^N\}_{n\in\mathbb{N}}$ such that, for every $n \in \mathbb{N}$, G_n $(1/6)$-fools time-$t(N)$ RTMs. Then,* $\mathrm{MCSP}[s(n)]$ *is not in* $\mathsf{BPTIME}_1[t(N)]$.

Proof. We prove the contrapositive. Let M be a time-t RTM that decides $\mathrm{MCSP}[s(n)]$. Fix any $n \in \mathbb{N}$. For any seed $z \in \{0,1\}^{s(n)}$, we have $G_n(z) \in \mathrm{MCSP}[s(n)]$ since G_n is local. Thus, $\Pr_r[M(G_n(z), r) = 1] \ge 2/3$. On the other hand, pick a string $w \in \{0,1\}^N$ chosen uniformly at random. By the counting argument of Proposition 8, we get $\Pr_w[w \notin \mathrm{MCSP}[s(n)]] \ge 1 - o(1)$. Thus, we have $\Pr_{w,r}[M(w,r) = 1] \le o(1) + 1/3 < 1/2$. Therefore,

$$\Pr_{z,r}[M(G_n(z), r) = 1] - \Pr_{w,r}[M(w,r) = 1] > \frac{2}{3} - \frac{1}{2} = \frac{1}{6},$$

which means that G_n does not fool RTMs. ◀

3 MCSP lower bounds against one-tape oracle RTMs

In this section, we present a proof of our main result.

▶ **Theorem 16** (Theorem 3, restated). *Let $O \subseteq \{0,1\}^*$ be any language. Then, for every constant $1/2 < \mu < 1$, $\mathrm{MCSP}[2^{\mu\cdot n}]$ on truth tables of size $N := 2^n$ is not in* $\mathsf{BPTIME}_1^O\left[N^{2\cdot(\mu' - o(1))}\right]$ *for all $1/2 < \mu' < \mu$, where all of the strings queried to O are of length $N^{o(1)}$.*

3.1 Connections to hardness magnification

As discussed in Section 1.1.1, Theorem 16 implies that establishing hardness magnification phenomena for MCSP, when the circuit size threshold parameter is $2^{0.51n}$, would require the development of new techniques; see Remark 19. To explain why this is true, we shall first require the following result by McKay, Murray, and Williams [30] that gives an oracle streaming algorithm for MCSP.

▶ **Lemma 17** ([30, Theorem 1.2]). *Let $s : \mathbb{N} \to \mathbb{N}$ be a size function, with $s(n) \geq n$ for all n, and $N := 2^n$. Then, there is a one-pass streaming algorithm for $\mathrm{MCSP}[s(n)]$ on N-bit inputs running in $N \cdot \widetilde{O}(s(n))$ time with $\widetilde{O}\left(s(n)^2\right)$ update time and $\widetilde{O}(s(n))$ space, using an oracle for $\Sigma_3\mathrm{SAT}$ with queries of length $\widetilde{O}(s(n))$.*

A corollary of Lemma 17 and Lemma 10 is the following.

▶ **Corollary 18** (Consequences of hardness magnification from currently known techniques). *Let $s : \mathbb{N} \to \mathbb{N}$ be a size function. Then, $\mathrm{MCSP}[s(n)]$ on truth tables of length $N := 2^n$ is in $\mathsf{DTIME}_1^O[N \cdot \mathrm{poly}(s(n))]$, for some $O \in \Sigma_3^{\mathsf{P}}$, where all of the strings queried to O are of length at most $\mathrm{poly}(s(n))$.*

The following remark summarizes the main idea of this subsection.

▶ Remark 19. By Corollary 18, we see that if $s(n) = 2^{\mu \cdot n}$, for $\mu = o(1)$, then $\mathrm{MCSP}[s(n)]$ is in $\mathsf{DTIME}_1^O\left[N^{1+o(1)}\right]$, where all of the strings queried to O are of length $N^{o(1)}$. In light of this observation, Theorem 16 is important for the following reason. As $\mathsf{DTIME}_1^O\left[N^{1+o(1)}\right]$ is a subset of $\mathsf{BPTIME}_1^O\left[N^{2\cdot(\mu'-o(1))}\right]$ for all $1/2 < \mu' < 1$ and all languages $O \subseteq \{0,1\}^*$, Theorem 16 shows that establishing hardness magnification phenomena for $\mathrm{MCSP}[s(n)]$ like that of Theorem 1, when $s(n) = 2^{\mu \cdot n}$ for any constant $1/2 < \mu < 1$, would require the development of techniques that do not rely on designing oracle algorithms that make short oracle queries.

3.1.1 Comparison with the locality barrier

Chen, Hirahara, Oliveira, Pich, Rajgopal, and Santhanam [9] introduced the "locality barrier" to explain why it will be difficult to acquire a major complexity breakthrough through the lens of hardness magnification. Their reasoning goes as follows:

> Existing magnification theorems unconditionally show that problems, against which some circuit lower bound implies a complexity-theoretic breakthrough, admit highly efficient small fan-in oracle circuits, while lower bound techniques against weak circuit models quite often easily extend to circuits containing such oracles.

Our Remark 19, therefore, is close in spirit to the results of Chen et al. [9]: We make use of a lower bound (Theorem 16) to motivate the development of new techniques for proving hardness magnification phenomena while Chen et al. make use of hardness magnification phenomena to motivate the development of new techniques for acquiring lower bounds; a notable difference is that we consider one-tape Turing machines while they consider Boolean circuits.

3.2 Proof of Theorem 16

In order to prove Theorem 16, our goal is to construct a local pseudorandom generator that fools oracle RTMs and then apply Lemma 15. Viola [38] constructed a pseudorandom generator that fools the one-tape Turing machine model (DTM).[5] We will show that, in fact, the same construction fools oracle RTMs as well. In order to do so, we recall the idea of Viola [38]. The idea is that, in order to fool DTMs, it is sufficient to use a PRG that ε-fools ROBPs for an exponentially small ε. This is because time-t DTMs can be written as the sum of an exponential number of ROBPs.

▶ **Lemma 20** (Viola [38]). *Let $n \in \mathbb{N}$ and M be a q-state time-t DTM. Then, there is a family $\{P_\alpha\}_{\alpha \in A}$ of n-input ROBPs of width $\exp\bigl(O\bigl(\sqrt{t} \cdot \log(tq)\bigr)\bigr)$ such that, for any $x \in \{0,1\}^n$,*

$$M(x) = \sum_{\alpha \in A} P_\alpha(x),$$

where $|A| \le (tq)^{O(\sqrt{t})}$.

By a simple calculation, any pseudorandom generator that $\varepsilon/|A|$-fools ROBPs also ε-fools DTMs. Viola [38] then used the pseudorandom generator of Forbes and Kelley [13] that fools ROBPs. By a careful examination, we will show that the Forbes-Kelley pseudorandom generator is local; see the full version.

▶ **Theorem 21** (Forbes-Kelley PRG is local). *There exists a local pseudorandom generator with seed length $\widetilde{O}\bigl((\sqrt{t} + \log(1/\varepsilon)) \cdot \log q\bigr)$ that ε-fools q-state time-t n-input DTMs for any $t \ge n$.*

Our main idea for obtaining an oracle randomized Turing machine lower bound is that Viola's reduction can be applied to *non-uniform computational models*, i.e., q-state Turing machines where q can become large as the input length becomes large. More specifically, it is possible to incorporate all possible oracle queries [along with their answers] and any good coin flip sequence r into the internal states of DTMs.

▶ **Lemma 22.** *For an input length $n \in \mathbb{N}$, for any q-state time-t oracle RTM M, that only queries strings of length at most ℓ to its oracle O, and a coin flip sequence $r \in \{0,1\}^t$, there exists some $\bigl(q \cdot 2^\ell \cdot t\bigr)$-state time-$t$ DTM M' such that $M'(x) = M^O(x, r)$ for every input $x \in \{0,1\}^n$.*

Proof. Let Q_M denote the set of the states of M. We define the set of the states of M' as

$$Q_{M'} := \Bigl\{(q, s, b, i) \in Q_M \times \{0,1\}^\ell \times \{0,1\} \times [t] \mid O(s) = b\Bigr\}.$$

The transition from the state $(q, s, b, i) \in Q_{M'}$ can be defined in a natural way, by using the i-th bit of r, namely r_i, the state q, and the fact that $O(s) = b$. ◀

▶ **Corollary 23.** *There exists a local pseudorandom generator with seed length $\sigma(t, q, \varepsilon) = \widetilde{O}\bigl((\sqrt{t} + \log(1/\varepsilon)) \cdot \log(q \cdot 2^\ell \cdot t)\bigr)$ that ε-fools q-state time-t n-input oracle RTMs that may only query strings of length at most ℓ to their oracle, for any $t \ge n$.*

[5] We note that our definition of PRG is different from that of [38] in that a random tape is not regarded as an input tape.

Proof. We hard-code the oracle queries and their answers in the internal states and, moreover, we use an averaging argument to fix one good coin flip sequence r. Let M be any q-state time-t oracle RTM that may query to its oracle O strings of length at most ℓ. Let G be a PRG from Theorem 21. We have that

$$\begin{aligned}
&\left| \mathop{\mathbf{Exp}}_{r\sim\{0,1\}^t}\left[\mathop{\mathbf{Exp}}_{x\sim\{0,1\}^n}\left[M^O(x,r)\right]\right] - \mathop{\mathbf{Exp}}_{r\sim\{0,1\}^t}\left[\mathop{\mathbf{Exp}}_{y\sim\{0,1\}^{\sigma(t,q,\varepsilon)}}\left[M^O(G(y),r)\right]\right]\right| \\
&\qquad = \left| \mathop{\mathbf{Exp}}_{r}\left[\mathop{\mathbf{Exp}}_{x}\left[M^O(x,r)\right] - \mathop{\mathbf{Exp}}_{y}\left[M^O(G(y),r)\right]\right]\right| \\
&\qquad \le \mathop{\mathbf{Exp}}_{r}\left[\left| \mathop{\mathbf{Exp}}_{x}\left[M^O(x,r)\right] - \mathop{\mathbf{Exp}}_{y}\left[M^O(G(y),r)\right]\right|\right] \\
&\qquad \le \left| \mathop{\mathbf{Exp}}_{x}\left[M^O(x,r^*)\right] - \mathop{\mathbf{Exp}}_{y}\left[M^O(G(y),r^*)\right]\right|,
\end{aligned}$$

for some $r^* \in \{0,1\}^t$, by an averaging argument. By applying Lemma 22, for M^O, O, and r^*, we obtain an equivalent $\left(q \cdot 2^\ell \cdot t\right)$-state time-$t$ DTM M'. The result now follows from Theorem 21. Specifically,

$$\left| \mathop{\mathbf{Exp}}_{x}\left[M^O(x,r^*)\right] - \mathop{\mathbf{Exp}}_{y}\left[M^O(G(y),r^*)\right]\right| = \left| \mathop{\mathbf{Exp}}_{x}[M'(x)] - \mathop{\mathbf{Exp}}_{y}[M'(G(y))]\right| \le \varepsilon. \qquad \blacktriangleleft$$

Proof of Theorem 16. Take the local pseudorandom generator G of Corollary 23 with parameter $\varepsilon := 1/6$. Let $1/2 < \mu' < \mu < 1$ be arbitrary constants. Let $t, s, \ell \colon \mathbb{N} \to \mathbb{N}$ be functions such that $t(N) = N^{2\cdot\left(\mu' - o(1)\right)}$, $s(n) = 2^{\mu\cdot n}$, and $\ell(n) = 2^{o(n)}$. Then, the seed length of G is at most

$$\widetilde{O}\left(\sqrt{t(N)} \cdot (\log q + \ell(n))\right) \le \widetilde{O}(N^{\mu' - o(1) + o(1)}) \le s(n),$$

where $N = 2^n$. Since $s(n) = o(2^n/n)$, by Lemma 15, we obtain that $\mathrm{MCSP}[s(n)] \notin \mathsf{BPTIME}_1^O[t(N)]$, where all of the strings queried to O are of length $N^{o(1)}$. ◀

4 MKTP lower bounds against branching programs

In this section, we develop a proof technique for applying Nečiporuk's method to MKTP and prove Theorem 4. The KT-complexity is formally defined as follows.

▶ **Definition 24.** *Let U be an efficient universal Turing machine. For a string $x \in \{0,1\}^*$, the* KT-complexity *of x is defined as follows.*

$$\mathrm{KT}(x) := \min\{|d| + t \mid U^d(i) \textit{ outputs } x_i \textit{ in time } t \textit{ for every } i \in [|x|+1]\}.$$

Here we define x_i as the ith bit of x if $i \le |x|$ and $\perp$ otherwise.

For a threshold $\theta \colon \mathbb{N} \to \mathbb{N}$, we denote by $\mathrm{MKTP}[\theta]$ the problem of deciding whether $\mathrm{KT}(x) \le \theta(|x|)$ given a string $x \in \{0,1\}^*$ as input.

Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function and $\rho \in \{0,1,*\}^n$ a restriction. The *ρ-restricted version of f* is a function, denoted by $f\!\upharpoonright_\rho$, such that for any $x \in \{0,1\}^n$ it is the case that $f\!\upharpoonright_\rho(x) := f(y)$ where $y \in \{0,1\}^n$ and, for all $1 \le i \le n$, $y_i := \rho(i)$ if $\rho(i) \in \{0,1\}$, else $y_i := x_i$.

For a function $f\colon \{0,1\}^n \to \{0,1\}$, we partition the input variables $[n]$ into disjoint blocks $V_1, \cdots, V_m$, where $|V_i| = v$ for each $i \in [m]$ and $n = vm$. ($v = O(\log n)$ will be chosen later.) The idea of the Nečiporuk's method is to lower-bound the number of subfunctions. For each $i \in [m]$, we define $c_i(f)$ to be the number of distinct functions $f\lceil_\rho$ such that $\rho\colon [n] \to \{0,1,*\}$ is a restriction with $\rho^{-1}(*) = V_i$.

The Nečiporuk method can be then summarized as follows.

▶ **Theorem 25** (Nečiporuk [32]; cf. [24, Theorem 15.1]). *The size of a branching program computing f is at least $\Omega\left(\sum_{i=1}^m \log c_i(f)/\log\log c_i(f)\right)$. The size of a non-deterministic branching program or a parity branching program computing f is at least $\Omega\left(\sum_{i=1}^m \sqrt{\log c_i(f)}\right)$.*

Our main technical result of this section is the following.

▶ **Theorem 26.** *Let $f\colon \{0,1\}^n \to \{0,1\}$ be $\mathrm{MKTP}[\theta]$ on n-bit inputs for $\theta := n - 3c\log n - 4$, where $c > 0$ is a universal constant. Then, for every $i \in [m]$, it holds that $c_i(f) = 2^{\Omega(n)}$.*

The lower bounds for branching programs (Theorem 4) immediately follow from Theorem 26 and Theorem 25.

In our proof of Theorem 26, we only need the following two properties of KT-complexity.

1. The resource-unbounded Kolmogorov complexity[6] provides a lower bound on the KT-complexity. That is, $\mathrm{K}(x) \le \mathrm{KT}(x)$ for any $x \in \{0,1\}^*$.
2. For any strings $\rho_1, \cdots, \rho_m \in \{0,1\}^v$ such that there exist distinct indices $i \neq j \in [m]$ such that $\rho_i = \rho_j$, we have $\mathrm{KT}(\rho_1 \cdots \rho_m) \le (m-1)\cdot v + O(\log n)$. This is because each bit of the string $\rho_1 \cdots \rho_m$ can be described by the strings $\{\rho_1, \cdots, \rho_m\} \setminus \{\rho_j\}$ and the index $j \in [m]$ in time $O(\log n)$.[7]

For simplicity, we focus on the case when $i = 1$; the other case can be proved similarly. The idea of the proof is the following. Imagine that we pick $\rho \in \{*\}^{V_1} \times \{0,1\}^{V_2 \cup \cdots \cup V_m}$ uniformly at random. (Here we identify a restriction with a string in $\{0,1,*\}^{[n]}$.) We denote by $\rho_2 \in \{0,1\}^{V_2}, \cdots, \rho_m \in \{0,1\}^{V_m}$ the random bits such that $\rho = *^{V_1}\rho_2 \cdots \rho_m$. We will sometimes identify $\rho_2 \cdots \rho_m$ with ρ.

Consider the function $f\lceil_\rho\colon \{0,1\}^{V_1} \to \{0,1\}$ obtained by restricting f by ρ. Then, we expect that $f\lceil_\rho(\rho_i) = 1$ for any $i \in \{2, \cdots, m\}$ since $\mathrm{KT}(\rho_i\rho_2 \cdots \rho_m)$ is small, whereas $f\lceil_\rho(U) = 0$ for a random $U \sim \{0,1\}^{V_1}$ with high probability. Thus, the function $f\lceil_\rho$ is likely to be distinct for a randomly chosen ρ.

In order to make the argument formal, we proceed as follows. Pick ρ randomly. Then we add it to a set P while keeping the promise that the map $\rho \in P \mapsto f\lceil_\rho$ is injective. We will show that one can keep adding ρ until the size of P becomes exponentially large.

We will make use of symmetry of information of (resource-unbounded) Kolmogorov complexity.

▶ **Lemma 27.** *There exists a constant $c > 0$ such that, for any strings $x, y \in \{0,1\}^*$,*

$$\mathrm{K}(xy) \ge \mathrm{K}(x) + \mathrm{K}(y \mid x) - c\log \mathrm{K}(xy)\,.$$

[6] Let U be an efficient universal Turing machine. For a string $x \in \{0,1\}^*$, the *resource-unbounded Kolmogorov complexity of x* is defined as $\mathrm{K}(x) := \min\{|d| \mid U^d(i) \text{ outputs } x_i \text{ for every } i \in [|x|+1]\}$.

[7] Here we assume that the universal Turing machine is efficient. If the universal Turing machine is slower and the time is $\mathrm{polylog}(n)$, we obtain a branching program size lower bound of $n^2/\mathrm{polylog}(n)$.

We focus on restrictions ρ such that ρ is Kolmogorov-random. To this end, define

$$R := \{\rho \in \{0,1\}^{V_2 \cup \cdots \cup V_m} \mid \mathrm{K}(\rho) \geq |\rho| - 1\}$$

as the set of Kolmogorov-random restrictions ρ. By the standard counting argument, we have

$$\Pr_{\rho}[\rho \notin R] \leq \sum_{i=1}^{|\rho|-2} 2^i/2^{|\rho|} \leq \frac{1}{2}.$$

The following lemma is the key for counting the number of distinct subfunctions.

▶ **Lemma 28.** *Let $\rho' \in R$ be an arbitrary restriction and define $\theta := n - v + c\log n$. If $f{\restriction_\rho} \equiv f{\restriction_{\rho'}}$, then $\mathrm{K}(\rho_i \mid \rho') \leq 2c\log n + 1$ for any $i \in \{2, \cdots, m\}$.*

Proof. For each $i \in [m] \setminus \{1\}$,

$$\mathrm{KT}(\rho_i \rho_2 \cdots \rho_m) \leq |\rho_2| + \cdots + |\rho_m| + O(\log n) \leq (m-1) \cdot v + c\log n \leq \theta.$$

This means that $\rho_i \rho_2 \cdots \rho_m$ is a YES instance of MKTP$[\theta]$. Therefore, we have $1 = f{\restriction_\rho}(\rho_i) = f{\restriction_{\rho'}}(\rho_i)$, which implies that $\mathrm{KT}(\rho_i \rho'_2 \cdots \rho'_m) \leq \theta$. By the symmetry of information,

$$\theta \geq \mathrm{KT}(\rho_i \rho'_2 \cdots \rho'_m) \geq \mathrm{K}(\rho_i \rho'_2 \cdots \rho'_m) \geq \mathrm{K}(\rho'_2 \cdots \rho'_m) + \mathrm{K}(\rho_i \mid \rho'_2 \cdots \rho'_m) - c\log n.$$

Since $\rho' \in R$, we have $\mathrm{K}(\rho'_2 \cdots \rho'_m) \geq v(m-1) - 1 = n - v - 1$. Therefore,

$$\mathrm{K}(\rho_i \mid \rho'_2 \cdots \rho'_m) \leq \theta + c\log n - (n - v - 1) = 2c\log n + 1.$$ ◀

Now we set $v := 4c\log n + 4$. Then, for any $\rho' \in R$,

$$\begin{aligned}\Pr_{\rho}[f{\restriction_\rho} \equiv f{\restriction_{\rho'}}] &\leq \Pr[\forall i \in [m] \setminus \{1\},\ \mathrm{K}(\rho_i \mid \rho') \leq v/2 - 1] \\ &\leq (2^{v/2}/2^v)^{m-1} \\ &= 2^{-n/2 + v/2} \\ &\leq 2^{-n/3}.\end{aligned}$$

In particular, for any $P \subseteq R$, by the union bound, we obtain

$$\Pr_{\rho}[\exists \rho' \in P,\ f{\restriction_\rho} \equiv f{\restriction_{\rho'}}] \leq |P| \cdot 2^{-n/3}.$$

Therefore,

$$\Pr_{\rho}[\rho \notin R \text{ or } \exists \rho' \in P,\ f{\restriction_\rho} \equiv f{\restriction_{\rho'}}] \leq 1/2 + |P| \cdot 2^{-n/3},$$

which is strictly less than 1 if $|P| < 2^{n/3-1}$. To summarize, we established the following property.

▶ **Corollary 29.** *For any $P \subseteq R$ such that $|P| < 2^{n/3-1}$, there exists a restriction ρ such that $\rho \in R$ and $f{\restriction_\rho} \not\equiv f{\restriction_{\rho'}}$ for any $\rho' \in P$.*

In light of this, we can construct a large set P such that the map $\rho \in P \mapsto f{\restriction_\rho}$ is injective as follows: Starting from $P := \emptyset$, add a restriction $\rho \in R$ such that $f{\restriction_\rho} \not\equiv f{\restriction_{\rho'}}$ for any $\rho' \in P$, whose existence is guaranteed by Corollary 29 if $|P| < 2^{n/3-1}$. In this way, we obtain a set P such that $|P| \geq 2^{n/3-1}$ and each $f{\restriction_\rho}$ is distinct for any $\rho \in P$. We conclude that $c_1(f) \geq |P| \geq 2^{n/3-1}$. This completes the proof of Theorem 26.

References

1 Eric Allender, Harry Buhrman, Michal Koucký, Dieter van Melkebeek, and Detlef Ronneburger. Power from random strings. *SIAM J. Comput.*, 35(6):1467–1493, 2006.

2 Eric Allender and Shuichi Hirahara. New insights on the (non-)hardness of circuit minimization and related problems. In *Proceedings of the 42nd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 54:1–54:14, 2017.

3 Eric Allender and Michal Koucký. Amplifying lower bounds by means of self-reducibility. *J. ACM*, 57(3):14:1–14:36, 2010.

4 Eric Allender, Michal Koucký, Detlef Ronneburger, and Sambuddha Roy. The pervasive reach of resource-bounded Kolmogorov complexity in computational complexity theory. *J. Comput. Syst. Sci.*, 77(1):14–40, 2011.

5 Alexander E. Andreev, Juri L. Baskakov, Andrea E. F. Clementi, and José D. P. Rolim. Small pseudo-random sets yield hard functions: New tight explicit lower bounds for branching programs. In *Proceedings of the 26th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 179–189, 1999.

6 Paul Beame, Nathan Grosshans, Pierre McKenzie, and Luc Segoufin. Nondeterminism and an abstract formulation of Nečiporuk's lower bound method. *ACM Trans. Comput. Theory*, 9(1):5:1–5:34, 2016.

7 Allan Borodin, Alexander A. Razborov, and Roman Smolensky. On lower bounds for read-k-times branching programs. *Computational Complexity*, 3:1–18, 1993.

8 Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Learning algorithms from natural proofs. In *Proceedings of the 31st Conference on Computational Complexity (CCC)*, pages 10:1–10:24, 2016.

9 Lijie Chen, Shuichi Hirahara, Igor Carboni Oliveira, Ján Pich, Ninad Rajgopal, and Rahul Santhanam. Beyond natural proofs: Hardness magnification and locality. In *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, pages 70:1–70:48, 2020.

10 Lijie Chen, Ce Jin, and R. Ryan Williams. Hardness magnification for all sparse NP languages. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1240–1255, 2019.

11 Mahdi Cheraghchi, Valentine Kabanets, Zhenjian Lu, and Dimitrios Myrisiotis. Circuit lower bounds for MCSP from local pseudorandom generators. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 39:1–39:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

12 Stephen A. Cook and Robert A. Reckhow. Time-bounded random access machines. In Patrick C. Fischer, H. Paul Zeiger, Jeffrey D. Ullman, and Arnold L. Rosenberg, editors, *Proceedings of the 4th Annual ACM Symposium on Theory of Computing, May 1-3, 1972, Denver, Colorado, USA*, pages 73–80. ACM, 1972.

13 Michael A. Forbes and Zander Kelley. Pseudorandom generators for read-once branching programs, in any order. In *Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 946–955, 2018.

14 Alexander Golovnev, Rahul Ilango, Russell Impagliazzo, Valentine Kabanets, Antonina Kolokolova, and Avishay Tal. $\mathsf{AC}^0[p]$ lower bounds against MCSP via the coin problem. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 66:1–66:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

15 Elad Haramaty, Chin Ho Lee, and Emanuele Viola. Bounded independence plus noise fools products. *SIAM J. Comput.*, 47(2):493–523, 2018.

16 Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

17 F. C. Hennie. One-tape, off-line turing machine computations. *Inf. Control.*, 8(6):553–578, 1965.

18 Shuichi Hirahara. Non-black-box worst-case to average-case reductions within NP. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 247–258, 2018.

19 Shuichi Hirahara. Non-Disjoint Promise Problems from Meta-Computational View of Pseudorandom Generator Constructions, 2020. Manuscript.

20 Shuichi Hirahara and Rahul Santhanam. On the average-case complexity of MCSP and its variants. In *Proceedings of the 32nd Computational Complexity Conference (CCC)*, pages 7:1–7:20, 2017.

21 Shuichi Hirahara and Osamu Watanabe. Limits of minimum circuit size problem as oracle. In *Proceedings of the 31st Conference on Computational Complexity (CCC)*, pages 18:1–18:20, 2016.

22 John M. Hitchcock and Aduri Pavan. On the NP-completeness of the minimum circuit size problem. In *Proceedings of the 35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 236–245, 2015.

23 Russell Impagliazzo, Raghu Meka, and David Zuckerman. Pseudorandomness from Shrinkage. *J. ACM*, 66(2):11:1–11:16, 2019.

24 Stasys Jukna. *Boolean Function Complexity - Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer, 2012.

25 Valentine Kabanets and Jin-yi Cai. Circuit minimization problem. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 73–79, 2000.

26 Bala Kalyanasundaram and Georg Schnitger. Communication complexity and lower bounds for sequential computation. In *Informatik, Festschrift zum 60. Geburtstag von Günter Hotz*, pages 253–268. Teubner / Springer, 1992.

27 Chin Ho Lee. Fourier bounds and pseudorandom generators for product tests. In *Proceedings of the 34th Computational Complexity Conference (CCC)*, pages 7:1–7:25, 2019.

28 Wolfgang Maass. Quadratic lower bounds for deterministic and nondeterministic one-tape Turing machines (extended abstract). In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing (STOC)*, pages 401–408, 1984.

29 Wolfgang Maass and Amir Schorr. Speed-up of Turing machines with one work tape and a two-way input tape. *SIAM J. Comput.*, 16(1):195–202, 1987.

30 Dylan M. McKay, Cody D. Murray, and R. Ryan Williams. Weak lower bounds on resource-bounded compression imply strong separations of complexity classes. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 1215–1225, 2019.

31 Cody D. Murray and Richard Ryan Williams. On the (non) NP-hardness of computing circuit complexity. In *Proceedings of the 30th Conference on Computational Complexity (CCC)*, pages 365–380, 2015.

32 E.I. Nečiporuk. On a Boolean function. *Doklady Akademii Nauk SSSR*, 169(4):765–766, 1966. English translation in Soviet Mathematics Doklady.

33 Igor Carboni Oliveira, Ján Pich, and Rahul Santhanam. Hardness magnification near state-of-the-art lower bounds. In *Proceedings of the 34th Computational Complexity Conference (CCC)*, pages 27:1–27:29, 2019.

34 Igor Carboni Oliveira and Rahul Santhanam. Hardness magnification for natural problems. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 65–76, 2018.

35 Alexander A. Razborov and Steven Rudich. Natural proofs. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 204–213, 1994.

36 Claude E. Shannon. The synthesis of two-terminal switching circuits. *Bell Systems Technical Journal*, 28:59–98, 1949.

37 Dieter van Melkebeek and Ran Raz. A time lower bound for satisfiability. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, volume 3142 of *Lecture Notes in Computer Science*, pages 971–982. Springer, 2004.

38 Emanuele Viola. Pseudorandom bits and lower bounds for randomized Turing machines. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:51, 2019.

39 Osamu Watanabe. The time-precision tradeoff problem on on-line probabilistic Turing machines. *Theor. Comput. Sci.*, 24:105–117, 1983.

Inference and Mutual Information on Random Factor Graphs

Amin Coja-Oghlan ✉
Mathematics Institute, Goethe Universität Frankfurt am Main, Germany

Max Hahn-Klimroth ✉
Mathematics Institute, Goethe Universität Frankfurt am Main, Germany

Philipp Loick ✉
Mathematics Institute, Goethe Universität Frankfurt am Main, Germany

Noela Müller ✉
Mathematics Institute, University of Munich, Germany

Konstantinos Panagiotou ✉
Mathematics Institute, University of Munich, Germany

Matija Pasch ✉
Mathematics Institute, University of Munich, Germany

Abstract

Random factor graphs provide a powerful framework for the study of inference problems such as decoding problems or the stochastic block model. Information-theoretically the key quantity of interest is the mutual information between the observed factor graph and the underlying ground truth around which the factor graph was created; in the stochastic block model, this would be the planted partition. The mutual information gauges whether and how well the ground truth can be inferred from the observable data. For a very general model of random factor graphs we verify a formula for the mutual information predicted by physics techniques. As an application we prove a conjecture about low-density generator matrix codes from [Montanari: IEEE Transactions on Information Theory 2005]. Further applications include phase transitions of the stochastic block model and the mixed $\boldsymbol{k}$-spin model from physics.

2012 ACM Subject Classification Mathematics of computing → Probabilistic inference problems

Keywords and phrases Information theory, random factor graphs, inference problems, phase transitions

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.24

Related Version *Full Version*: `https://arxiv.org/abs/2007.07494`

Funding *Amin Coja-Oghlan*: Supported by DFG CO 646/3.
Max Hahn-Klimroth: Supported by Stiftung Polytechnische Gesellschaft and DFG FOR 2975.
Philipp Loick: Supported by DFG CO 646/3.
Noela Müller: Supported by the European Research Council, ERC Grant Agreement 772606–PTRCSP.
Konstantinos Panagiotou: Supported by the European Research Council, ERC Grant Agreement 772606–PTRCSP.
Matija Pasch: Supported by the European Research Council, ERC Grant Agreement 772606–PTRCSP.

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 24; pp. 24:1–24:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

1.1 Background and motivation

Since the 1990s there has been an immense interest in inference and learning problems on random graphs. One motivation has been to seize upon random graphs as benchmarks for inference algorithms of all creeds and denominations. An excellent example of this is the stochastic block model; the impressive literature on this model alone is surveyed in [1]. A second, no less salient motivation has been the use of random graphs in probabilistic constructions. Concrete examples include powerful error correcting codes such as low density generator matrix or low density parity check codes, which have since found their way into modern communications standards [20, 31]. Further prominent recent applications include compressed sensing and group testing [2, 14, 15]. It appears hardly a stretch to claim that in terms of real world impact these constructions occupy top ranks among applications of the probabilistic method and, indeed, modern combinatorics generally.

Yet many applications of the probabilistic method to inference problems still lack a satisfactory rigorous justification. Some are supported primarily by empirical evidence, i.e., not much more than a bunch of computer experiments. Quite a few others have been inspired by a versatile but non-rigorous approach from physics known as the "cavity method". But while there has been progress in recent years, vast gaps between the physics predictions and their rigorous vindications remain. One important reason for this is that the random graph models used in practical inference tend to be significantly more intricate than, say, a classical binomial random graph. For instance, a highly popular breed of low-density parity check codes use delicately tailored degree distributions for both the variable nodes and the check nodes of the Tanner graph [31].

In this paper we significantly advance the rigorous state of the art by corroborating important cavity method predictions wholesale for a rich class of inference problems that accommodates the very general choices of degree distributions of interest in high-dimensional Bayesian inference problems and coding theory. Generally, the objective in such inference problems is to recover the ground truth from the observable data. Think, for instance, of retrieving the hidden communities in the stochastic block model or of reconstructing the original message from a noisy codeword. For this broad class of models we rigorously establish the formulas that the cavity method predicts for the mutual information, which is the key information-theoretic potential that gauges precisely how much it is possible in principle to learn about the ground truth. Technically we build upon and extend the methods developed in [11] for random graph models of Erdős-Rényi type. While we follow a similar general proof strategy, the greater generality of the present results necessitates significant upgrades to virtually all of the moving parts. For example, due to the more rigid combinatorial structure of graphs with given degrees many of the manoeuvres that are straightforward for binomial random graphs now require delicate coupling arguments.

We proceed to highlight applications of our main results to three specific problems that have each received a great deal of attention in their own right: low-density generator matrix codes, the stochastic block model and the mixed $\boldsymbol{k}$-spin model, which hails from mathematical physics. Then in Section 2 we state the main results concerning the general class of random factor graph models. Section 3 contains an overview of the proof strategy and a detailed comparison with prior work.

1.2 Low-density generator matrix codes

A powerful and instructive class of error-correcting codes, low-density generator matrix ("ldgm") codes are based on random bipartite graphs with given degree distributions. Specifically, let $\boldsymbol{d}, \boldsymbol{k} \geq 0$ be bounded integer-valued random variables, let n be an integer and let $\boldsymbol{m} \sim \mathrm{Po}(n\mathbb{E}[\boldsymbol{d}]/\mathbb{E}[\boldsymbol{k}])$ be a Poisson variable. One vertex class $V = \{x_1, \ldots, x_n\}$ of the graph represents the bits of the original message. The other class $F = \{a_1, \ldots, a_{\boldsymbol{m}}\}$ represents the rows of the code's generator matrix. To obtain the random graph $\boldsymbol{G}$ create for each variable node x_i an independent copy $\boldsymbol{d}_i$ of $\boldsymbol{d}$. Similarly, create an independent copy $\boldsymbol{k}_i$ of $\boldsymbol{k}$ for each check node a_i. Then given the event $\left\{\sum_{i=1}^n \boldsymbol{d}_i = \sum_{i=1}^{\boldsymbol{m}} \boldsymbol{k}_i\right\}$ that the total degrees on both sides match let $\boldsymbol{G}$ be a random bipartite graph where every x_i has degree $\boldsymbol{d}_i$ and every a_i has degree $\boldsymbol{k}_i$. We tacitly restrict to n such that this event has positive probability.

The generator matrix of the ldgm code is now precisely the $\boldsymbol{m} \times n$ biadjacency matrix $A(\boldsymbol{G})$ of $\boldsymbol{G}$, viewed as a matrix over $\mathbb{F}_2$. Thus, the rows of $A(\boldsymbol{G})$ correspond to the check nodes $a_1, \ldots, a_{\boldsymbol{m}}$, the columns correspond to $x_1, \ldots, x_n$ and the (i,j)-entry equals one iff a_i and x_j are adjacent. For a given message $\boldsymbol{x} \in \mathbb{F}_2^n$ the corresponding codeword reads $\boldsymbol{y} = A(\boldsymbol{G})\boldsymbol{x} \in \mathbb{F}_2^{\boldsymbol{m}}$. The receiver on the other end of a noisy channel observes a scrambled version $\boldsymbol{y}^*$ of $\boldsymbol{y}$. Specifically, $\boldsymbol{y}^*$ is obtained from $\boldsymbol{y}$ by flipping every bit with probability $\eta \in (0, 1/2)$ independently. To gauge the potential of the code, the key question is how much information about the original $\boldsymbol{x}$ the receiver can possibly extract from $\boldsymbol{y}^*$. Naturally, the receiver also knows $\boldsymbol{G}$. Hence, we aim to work out the conditional mutual information

$$I(\boldsymbol{x}, \boldsymbol{y}^* \mid \boldsymbol{G}) = \sum_{x \in \mathbb{F}_2^n, y \in \mathbb{F}_2^{\boldsymbol{m}}} \mathbb{P}\left[\boldsymbol{x} = x, \boldsymbol{y}^* = y \mid \boldsymbol{G}\right] \log \frac{\mathbb{P}\left[\boldsymbol{x} = x, \boldsymbol{y}^* = y \mid \boldsymbol{G}\right]}{2^n \mathbb{P}\left[\boldsymbol{y}^* = y \mid \boldsymbol{G}\right]}.$$

A precise prediction as to its asymptotical value was put forward on the basis of the physicists' cavity method. As most such predictions, the formula comes as a variational problem that asks to optimise a functional called the Bethe free entropy over a space of probability measures. Specifically, let $\mathfrak{P}_*([-1,1])$ be the space of all probability measures ρ on the interval $[-1,1]$ with mean zero. Let $(\boldsymbol{\theta}_{i,j,\rho})_{i,j \geq 1} \subseteq [-1,1]$ be a family of samples from ρ. Further, let $(\boldsymbol{J}_i)_{i \geq 1}$ be Rademacher variables, i.e., $\mathbb{P}\left[\boldsymbol{J}_i = 1\right] = \mathbb{P}\left[\boldsymbol{J}_i = -1\right] = 1/2$. In addition, let $(\hat{\boldsymbol{k}}_i)_{i \geq 1}$ be random variables with distribution

$$\mathbb{P}\left[\hat{\boldsymbol{k}}_i = \ell\right] = \frac{\ell \mathbb{P}\left[\boldsymbol{k} = \ell\right]}{\mathbb{E}[\boldsymbol{k}]} \qquad (\ell \geq 0). \tag{1}$$

All of these are independent. Finally, let $\Lambda(z) = z \log(z)$. Then the Bethe free entropy reads

$$\begin{aligned}\mathcal{B}_{\mathrm{ldgm}}(\rho, \eta) = \mathbb{E}\left[\frac{1}{2}\Lambda\left(\sum_{\sigma \in \{0,1\}} \prod_{i=1}^{\boldsymbol{d}} 1 + (-1)^{\sigma} \boldsymbol{J}_i (1 - 2\eta) \prod_{j=1}^{\hat{\boldsymbol{k}}_i - 1} \boldsymbol{\theta}_{i,j,\rho}\right)\right] \\ - \frac{\mathbb{E}[\boldsymbol{d}]}{\mathbb{E}[\boldsymbol{k}]} \mathbb{E}\left[(\boldsymbol{k} - 1)\Lambda\left(1 + \boldsymbol{J}_1(1 - 2\eta) \prod_{j=1}^{\boldsymbol{k}} \boldsymbol{\theta}_{1,j,\rho}\right)\right].\end{aligned}$$

▶ **Theorem 1.** *For any* $\boldsymbol{d}, \boldsymbol{k}$ *and for all* $\eta \in (0,1)$ *we have*

$$\begin{aligned}\lim_{n \to \infty} \frac{1}{n} I(\boldsymbol{x}, \boldsymbol{y}^* \mid \boldsymbol{G}) = \left(1 + \frac{\mathbb{E}[\boldsymbol{d}]}{\mathbb{E}[\boldsymbol{k}]}\right) \log(2) + \frac{\mathbb{E}[\boldsymbol{d}]}{\mathbb{E}[\boldsymbol{k}]} (\eta \log(\eta) + (1 - \eta) \log(1 - \eta)) \\ - \sup_{\rho \in \mathfrak{P}_*[-1,1]} \mathcal{B}_{\mathrm{ldgm}}(\rho, \eta) \qquad \textit{in probability.}\end{aligned}$$

Theorem 1 completely solves a well known conjecture [25, Conjecture 1] and significantly extends the results from [32, 11], which required the restrictive assumption that the check degree $\boldsymbol{k}$ be constant.

A possible objection to a result such as Theorem 1 might be that the resulting formula appears exceedingly complicated as it leaves us with a potentially difficult variational problem. Yet two points are to be made in defense. First, by vindicating the precise formula predicted by the cavity method, the theorem and its proof show that this technique and the ideas behind it do indeed get to the bottom of the problem. Second, since the formula involves a supremum, any $\rho \in \mathfrak{P}_*[-1,1]$ yields an upper bound on the mutual information. Hence, the heuristic population dynamics algorithm deemed to produce good candidate maximisers and beloved of physicists, can be harnessed to get rigorous bounds in one direction. Finally, in some cases it is possible to precisely identify the maximiser analytically [6, 9].

1.3 The stochastic block model

An instructive model of graph clustering, the stochastic block model presumes that a random graph is created in two steps. First each of the n vertices $\{x_1,\ldots,x_n\}$ receives one of $q \geq 2$ possible colours $\boldsymbol{\sigma}^*_{x_i} \in [q]$ uniformly and independently. Then a sparse random graph is created where vertices with the same colour are either more likely to be connected by an edge (assortative case), or less likely (disassortative). Different versions of this model have been proposed. While in the simplest one edges are inserted independently, here we consider a model from [27] that produces a d-regular graph. Hence, let $d \geq 3$ be an integer and let $\boldsymbol{G} = \boldsymbol{G}(n,d)$ be a random d-regular graph. Further, given a parameter $\beta > 0$ let $\boldsymbol{G}^* = \boldsymbol{G}^*(n,d,\boldsymbol{\sigma}^*)$ be a random graph drawn from the distribution

$$\mathbb{P}\left[\boldsymbol{G}^* = G \mid \boldsymbol{\sigma}^*\right] \propto \exp\left[-\beta \sum_{vw \in E(G)} \mathbf{1}\left\{\boldsymbol{\sigma}^*_v = \boldsymbol{\sigma}^*_w\right\}\right], \tag{2}$$

with the $\propto$-symbol hiding the normalisation required to obtain a probability distribution. Thus, the parameter β tunes the penalty that we impose on monochromatic edges by comparison to the null model $\boldsymbol{G}$. At $\beta = 0$ there is no such penalty and $\boldsymbol{G}^*$ and $\boldsymbol{G}$ are identical. But even for positive β the random graphs $\boldsymbol{G}, \boldsymbol{G}^*$ may still be indistinguishable and in effect recovering $\boldsymbol{\sigma}^*$ may be impossible. Hence, a fundamental question is for what q, d, β it is possible to discriminate between $\boldsymbol{G}, \boldsymbol{G}^*$. Formally, we recall that the *Kullback-Leibler divergence* of $\boldsymbol{G}^*, \boldsymbol{G}$ is defined as $D_{\mathrm{KL}}\left(\boldsymbol{G}^* \| \boldsymbol{G}\right) = \sum_G \mathbb{P}\left[\boldsymbol{G}^* = G\right] \log \frac{\mathbb{P}[\boldsymbol{G}^*=G]}{\mathbb{P}[\boldsymbol{G}=G]}$. The Kullback-Leibler divergence is an information-theoretic potential that gauges the similarity of two random graph models. In particular, if $D_{\mathrm{KL}}\left(\boldsymbol{G}^* \| \boldsymbol{G}\right) = \Omega(n)$, then $\boldsymbol{G}, \boldsymbol{G}^*$ can be told apart because natural observables will take vastly different values on the two models. Whether $D_{\mathrm{KL}}\left(\boldsymbol{G}^* \| \boldsymbol{G}\right) = \Omega(n)$ depends on the value of the Bethe free entropy for the stochastic block model. To be precise, let $\mathcal{P}([q])$ be the set of all probability distributions $(\mu(1),\ldots,\mu(q))$ on $[q]$. We identify $\mathcal{P}([q])$ with the standard simplex in $\mathbb{R}^q$. Further, let $\mathfrak{P}_*([q])$ be the set of all probability measures π on $\mathcal{P}([q])$ such that $\int \mu(\sigma)\mathrm{d}\pi(\mu) = 1/q$ for every $\sigma \in [q]$. In other words, the mean of π is the barycenter of the simplex. Let $(\boldsymbol{\mu}_{i,\pi})_{i \geq 1}$ be a family of independent samples from π and let

$$\begin{aligned}\mathcal{B}_{\mathtt{sbm}}(\pi,\beta) = \mathbb{E}&\left[\frac{\Lambda\left(\sum_{\sigma=1}^{q}\prod_{i=1}^{d} 1-(1-\mathrm{e}^{-\beta})\boldsymbol{\mu}_{i,\pi}(\sigma)\right)}{q\left(1-(1-\mathrm{e}^{-\beta})/q\right)^{d}}\right]\\ &-\mathbb{E}\left[\frac{d\Lambda\left(1-(1-\mathrm{e}^{-\beta})\sum_{\sigma=1}^{q}\boldsymbol{\mu}_{1,\pi}(\sigma)\boldsymbol{\mu}_{2,\pi}(\sigma)\right)}{2\left(1-(1-\mathrm{e}^{-\beta})/q\right)}\right].\end{aligned}$$

▶ **Theorem 2.** *Let*

$$\beta^* = \inf\left\{\beta > 0 : \sup_{\pi \in \mathfrak{P}_*([q])} \mathcal{B}_{\mathtt{sbm}}(\pi, \beta) > \log(q) + \frac{d}{2}\log\left(1 - (1 - \mathrm{e}^{-\beta})/q\right)\right\}.$$

(i) *If $\beta < \beta^*$, then $\lim_{n\to\infty} \frac{1}{n} D_{\mathrm{KL}}\left(\boldsymbol{G}^* \| \boldsymbol{G}\right) = 0$.*

(ii) *If $\beta > \beta^*$, then $\lim_{n\to\infty} \frac{1}{n} D_{\mathrm{KL}}\left(\boldsymbol{G}^* \| \boldsymbol{G}\right) > 0$.*

Theorem 2 easily implies that for $\beta > \beta^*$ it is information-theoretically possible to recover a non-trivial approximation to $\boldsymbol{\sigma}^*$ from $\boldsymbol{G}^*$. In other words, there exists an exponential time algorithm that likely outputs a colouring τ of the vertices that has a significantly greater overlap with the ground truth $\boldsymbol{\sigma}^*$ than a random guess. An open question is whether for $\beta > \beta^*$ this problem can even be solved by a polynomial time algorithm. The going conjecture is that in general the answer is "no" and that efficient recoverability kicks in only at a second threshold $\beta^{**} > \beta^*$ for many interesting choices of q, d [12].

1.4 The mixed k-spin model

Not only do the main results of this paper facilitate rigorous proofs of physics predictions for problems in computer science, but also, conversely, do we obtain new theorems on problems of keen interest in statistical physics. For example, the mixed $\boldsymbol{k}$-spin model is an important spin glass model [28]; its purpose is to describe the magnetic interactions in metallic alloys. To define the model let $\boldsymbol{k} \geq 2$ be an integer-valued random variable such that $\mathbb{E}[\boldsymbol{k}^{2+\varepsilon}] < \infty$ for some $\varepsilon > 0$ and $\mathbb{P}[\boldsymbol{k} = 2] > 0$. Let $(\boldsymbol{k}_i)_{i\geq 1}$ be a sequence of independent copies of $\boldsymbol{k}$. Moreover, let $d > 0$ and let $\boldsymbol{H} = \boldsymbol{H}_{\boldsymbol{k}}(n, \boldsymbol{m})$ be a (non-uniform) random hypergraph on $V_n = \{x_1, \ldots, x_n\}$ with $\boldsymbol{m} = \mathrm{Po}(dn/\mathbb{E}[\boldsymbol{k}])$ independent hyperedges $a_1, \ldots, a_{\boldsymbol{m}}$ such that a_i comprises $\boldsymbol{k}_i$ vertices, drawn uniformly without replacement. Thus, in the special case that $\boldsymbol{k}$ is constant we obtain the classical binomial random hypergraph. To turn this random hypergraph into a spin glass model we draw for each of its edges a_i an independent standard Gaussian $\boldsymbol{J}_i$. Additionally, let $\beta > 0$ be a parameter, commonly coined the inverse temperature. Then the *Boltzmann distribution* of the model is the probability distribution on $\{\pm 1\}^{V_n}$ defined by

$$\mu_{\boldsymbol{H},\boldsymbol{J},\beta}(\sigma) = \frac{\exp\left(\beta \sum_{i=1}^{\boldsymbol{m}} \boldsymbol{J}_i \prod_{x \in a_i} \sigma_x\right)}{Z(\boldsymbol{H}, \boldsymbol{J}, \beta)} \qquad (\sigma \in \{\pm 1\}^{V_n}),$$

where $Z(\boldsymbol{H}, \boldsymbol{J}, \beta) = \sum_{\tau \in \{\pm 1\}^{V_n}} \exp\left(\beta \sum_{i=1}^{\boldsymbol{m}} \boldsymbol{J}_i \prod_{x \in a_i} \tau_x\right)$. The normalising term $Z(\boldsymbol{H}, \boldsymbol{J}, \beta)$ is known as the *partition function.*

A key question is whether for given $d, \beta, \boldsymbol{k}$ there occur long-range correlations between the magnetic "spins" observed at $x_1, \ldots, x_n$. Formally, let $\boldsymbol{\sigma} \in \{\pm 1\}^{V_n}$ signify a sample from the Boltzmann distribution. Then we say that *long-range correlations are absent* if

$$\lim_{n\to\infty} \frac{1}{n^2} \sum_{x,y \in V_n} \mathbb{E}\left|\mu_{\boldsymbol{H},\boldsymbol{J},\beta}(\{\boldsymbol{\sigma}_x = \boldsymbol{\sigma}_y = 1\}) - \mu_{\boldsymbol{H},\boldsymbol{J},\beta}(\{\boldsymbol{\sigma}_x = 1\})\mu_{\boldsymbol{H},\boldsymbol{J},\beta}(\{\boldsymbol{\sigma}_y = 1\})\right| = 0.$$

In words, the equation expresses that for most pairs x, y of vertices the spins $\boldsymbol{\sigma}_x, \boldsymbol{\sigma}_y$ are essentially independent. If this is violated, we say that long-range correlations are present.

According to physics predictions for a given $\beta > 0$ long-range correlations emerge at a critical value $d_{\beta,\boldsymbol{k}}$ that can be determined in terms of the Bethe free entropy [19, 24]. The methods developed in this paper enable us to corroborate this formula rigorously. Specifically, let $\mathfrak{P}_*([-1,1])$ be the space of all probability measures on $[-1,1]$ with mean zero. Given $\pi \in \mathfrak{P}_*([-1,1])$ let $(\boldsymbol{\mu}_{i,j,\pi})_{i,j\geq 1}$ be a family of independent samples from π. Additionally, let $(\hat{\boldsymbol{k}}_i)_{i\geq 1}$ be a family of independent random variables with point masses (1) and let $\boldsymbol{d} = \mathrm{Po}(d)$. Then the Bethe free entropy $\mathcal{B}_{\boldsymbol{k}-\mathrm{spin}}(\pi)$ of the $\boldsymbol{k}$-spin model is given by the expression

$$\frac{1}{2}\mathbb{E}\left[\Lambda\left(\sum_{\sigma_1\in\{\pm1\}}\prod_{i=1}^{\boldsymbol{d}}\left(1+\sum_{\sigma_2,\ldots,\sigma_{\hat{\boldsymbol{k}}_i}\in\{\pm1\}}\tanh\left(\beta\boldsymbol{J}_j\prod_{j\in[\hat{\boldsymbol{k}}_i]}\sigma_j\right)\prod_{j=2}^{\hat{\boldsymbol{k}}_i}\frac{1+\sigma_j\boldsymbol{\mu}_{i,j,\pi}}{2}\right)\right)\right]$$
$$-\frac{d}{\mathbb{E}[\boldsymbol{k}]}\mathbb{E}\left[(\boldsymbol{k}-1)\Lambda\left(1+\sum_{\sigma\in\{\pm1\}^{\boldsymbol{k}}}\tanh\left(\beta\boldsymbol{J}_1\prod_{i=1}^{\boldsymbol{k}}\sigma_j\right)\prod_{i=1}^{\boldsymbol{k}}\frac{1+\sigma_i\boldsymbol{\mu}_{1,i,\pi}}{2}\right)\right].$$

▶ **Theorem 3.** *Let* $d_{\beta,\boldsymbol{k}} = \inf\left\{d > 0 : \sup_{\pi\in\mathfrak{P}_*([-1,1])}\mathcal{B}_{\boldsymbol{k}-\mathrm{spin}}(\pi) > \log 2\right\}$.

(i) *Long-range correlations are absent for* $d < d_{\beta,\boldsymbol{k}}$*.*

(ii) *For any* $\varepsilon > 0$ *there exists* $d_{\beta,\boldsymbol{k}} < d < d_{\beta,\boldsymbol{k}} + \varepsilon$ *where long-range correlations are present.*

Thus, the point $d_{\beta,\boldsymbol{k}}$, characterised by the Bethe variational principle, marks the onset of complex magnetic interactions in the mixed $\boldsymbol{k}$-spin model. This critical value is known as the *replica symmetry breaking* phase transition in physics jargon. As a further application of the main results we can pinpoint the so-called condensation phase transition of the Potts antiferromagnet on random d-regular graphs, another problem of interest in mathematical physics. More details can be found in Section 16 of the full version.

2 The mutual information of random factor graphs

The theorems quoted in Section 1 are easy consequences of results on general random factor graph models. These more general theorems, one of which we present next, constitute the main results of the paper.

2.1 Random factor graph models

Remarkably many classical problems from combinatorics, statistics and physics can be expressed conveniently in the language of factor graph models [24, 29, 34]. A factor graph G is a bipartite graph whose vertex classes are variable nodes $V(G)$ and factor nodes $F(G)$. The former represent the variables of the combinatorial problem in question, such as the individual bits of a codeword. Generally we assume that these variables range over a domain Ω of size $q = |\Omega| \geq 2$. Moreover, the factor nodes encode the interactions between the variables, such as the linear relations imposed by the check matrix of a code. Each factor node $a \in F(G)$ comes with a function $\psi_a : \Omega^{\partial a} \to (0,\infty)$ that assigns a positive weight to value combinations of the adjacent variables ∂a. The factor graph gives rise to a probability distribution

$$\mu_G(\sigma) = \frac{\psi_G(\sigma)}{Z_G}, \text{ where } \psi_G(\sigma) = \prod_{a\in F(G)}\psi_a(\sigma_{\partial a}) \text{ and } Z_G = \sum_{\tau\in\Omega^{V(G)}}\psi_G(\tau) \quad (\sigma \in \Omega^{V(G)}). \tag{3}$$

To describe problems such as the ones from Section 1 we introduce models where the factor graph itself is random. Specifically, let $\boldsymbol{d}, \boldsymbol{k} \geq 0$ be integer-valued random variables and let $(\boldsymbol{d}_i)_{i\geq 1}$, $(\boldsymbol{k}_i)_{i\geq 1}$ be independent copies of $\boldsymbol{d}, \boldsymbol{k}$. Further, for each k in the support of $\boldsymbol{k}$ let Ψ_k be a finite set of k-ary functions $\psi : \Omega^k \to (0, \infty)$. Let P_k be a probability distribution on Ψ_k and let us write $\boldsymbol{\psi}_k$ for a sample from P_k. Further, let $\boldsymbol{\psi}$ be a random variable distributed as $\boldsymbol{\psi}_{\boldsymbol{k}}$, let P be the distribution of $\boldsymbol{\psi}_{\boldsymbol{k}}$ and let k_ψ denote the arity of ψ.

Now, to construct a factor graph let $V_n = \{x_1, \ldots, x_n\}$ be a set of variable nodes and let $F_{\boldsymbol{m}} = \{a_1, \ldots, a_{\boldsymbol{m}}\}$ be a set of $\boldsymbol{m} \sim \mathrm{Po}(n\mathbb{E}[\boldsymbol{d}]/\mathbb{E}[\boldsymbol{k}])$ factor nodes. We obtain the random factor graph $\boldsymbol{G}$ as follows.

G1 given the event $\sum_{i=1}^{n} \boldsymbol{d}_i = \sum_{i=1}^{\boldsymbol{m}} \boldsymbol{k}_i$, choose a bipartite graph on variable and factor nodes such that every x_i has degree $\boldsymbol{d}_i$ and every a_j has degree $\boldsymbol{k}_j$ uniformly at random.

G2 choose for every factor node a_i a weight function ψ_{a_i} from the distribution $\boldsymbol{\psi}_{\boldsymbol{k}_i}$.

In the language of inference problems the random factor graph $\boldsymbol{G}$ is going to provide a null model because the weight functions in **G2** are independent of the graph structure from **G1**. For instance, in the context of the stochastic block model from Section 1.3, this model plays the role of the purely random graph without a particular underlying colouring.

2.2 The teacher-student scheme

The teacher-student scheme organically turns the null model into an inference problem. A helpful metaphor might be to imagine a teacher who attempts to convey a ground truth $\boldsymbol{\sigma}^*$ to a student by presenting examples. The ground truth itself is a random vector chosen uniformly from the space Ω^{V_n}. The set of examples corresponds to a factor graph $\boldsymbol{G}^*$.

To be precise, let $\mathfrak{D}$ be the σ-algebra generated by the degrees and the total number of factor nodes of the null model $\boldsymbol{G}$. Then the factor graph $\boldsymbol{G}^*$ is chosen from the distribution

$$\mathbb{P}\left[\boldsymbol{G}^* = G \mid \mathfrak{D}, \boldsymbol{\sigma}^*\right] = \frac{\mathbb{P}\left[\boldsymbol{G} = G \mid \mathfrak{D}\right] \psi_G(\boldsymbol{\sigma}^*)}{\mathbb{E}[\psi_{\boldsymbol{G}}(\boldsymbol{\sigma}^*) \mid \mathfrak{D}, \boldsymbol{\sigma}^*]}. \tag{4}$$

Hence, we reweigh the null model **G1**–**G2** according to the ground truth $\boldsymbol{\sigma}^*$, rewarding graphs under which $\boldsymbol{\sigma}^*$ receives a higher weight. In the case of the stochastic block model, $\boldsymbol{G}^*$ matches the reweighing (2) that prefers bichromatic edges. The obvious question is how much of an imprint $\boldsymbol{\sigma}^*$ leaves on the resulting factor graph $\boldsymbol{G}^*$? Before we answer this question in general let us illustrate how the examples from Section 1 fit into the general framework.

▶ **Example 4** (ldgm codes). Let $\Omega = \{+1, -1\}$ with $+1 = (-1)^0$ representing $0 \in \mathbb{F}_2$ and -1 representing $1 \in \mathbb{F}_2$. For every degree $k \geq 1$ there are two k-ary weight functions $\psi_{\eta,k,\pm 1}$ defined by $\psi_{\eta,k,J}(\sigma) = 1 - (1 - 2\eta) J \prod_{i=1}^{k} \sigma_i$ for $\sigma \in \Omega^k$.

The probability distribution P_k is defined by $P(\psi_{\eta,k,J}) = 1/2$. With this setup the bipartite graph structure of the null model $\boldsymbol{G}$ coincides with the bipartite graph introduced in Section 1.2. Moreover, the ± 1-labels of the weight functions (i.e., value of J such that $\psi_{a_i} = \psi_{\eta,\boldsymbol{k}_i,J}$) represent the entries of the vector $\boldsymbol{y}^*$. Thus, while in the null model $\boldsymbol{G}$ these vector entries are purely random, in the reweighted model $\boldsymbol{G}^*$ the labels are distributed precisely as the entries of the vector $\boldsymbol{y}^*$ from the ldgm model.

▶ **Example 5** (stochastic block model). Let $\Omega = [q]$ be a set of q colours. We introduce a single binary weight function $\psi_{\beta,q}(\sigma_1, \sigma_2) = \exp(-\beta \mathbf{1}\{\sigma_1 = \sigma_2\})$ and we let $\boldsymbol{d}$ be the constant random variable d. With this weight function the construction (4) coincides with the definition (2) of the stochastic block model.

The main theorem is going to provide a formula for the mutual information of $\boldsymbol{G}^*$ and the ground truth $\boldsymbol{\sigma}^*$, provided that the distribution P on weight functions satisfies a number of easy-to-check conditions. To state these conditions let us denote by $\mathcal{P}(\Omega)$ the set of all probability distributions on Ω, endowed with the topology inherited from Euclidean space. Moreover, let $\mathfrak{P}_*(\Omega)$ signify the space of all probability measures π on $\mathcal{P}(\Omega)$ such that $\int_{\mathcal{P}(\Omega)} \mu(\omega)\mathrm{d}\pi(\mu) = 1/q$ for all $\omega \in \Omega$. Finally, for a given $\pi \in \mathfrak{P}_*(\Omega)$ let $(\boldsymbol{\mu}_{i,\pi})_{i\geq 1}$ be independent samples from π and recall $\Lambda(x) = x\log x$. The assumptions read as follows.

DEG there exists $\varepsilon > 0$ such that $\mathbb{E}[\boldsymbol{d}^{2+\varepsilon}], \mathbb{E}[\boldsymbol{k}^{2+\varepsilon}] < \infty$.

SYM there exist reals $\varepsilon, \xi > 0$ such that for all $k \in \operatorname{supp} \boldsymbol{k}$, $\psi \in \Psi_k$, $j \in [k]$, $\omega \in \Omega$ we have

$$\sum_{\sigma\in\Omega^k} \mathbf{1}\{\sigma_j = \omega\}\,\psi(\sigma) = q^{k-1}\xi, \qquad\qquad \varepsilon < \psi(\sigma) < 1/\varepsilon \quad (\sigma \in \Omega^k).$$

BAL for every $k \in \operatorname{supp} \boldsymbol{k}$ the function $\mu \in \mathcal{P}(\Omega) \mapsto \sum_{\sigma\in\Omega^k} \mathbb{E}\left[\boldsymbol{\psi}_k(\sigma)\right] \prod_{i=1}^k \mu(\sigma_i)$ is concave and attains its maximum at the uniform distribution on Ω.

POS for any two probability distributions $\pi, \pi' \in \mathfrak{P}_*(\Omega)$ and any $k \in \operatorname{supp} \boldsymbol{k}$ we have

$$\mathbb{E}\left[\Lambda\left(\sum_{\tau\in\Omega^k} \boldsymbol{\psi}_k(\tau)\prod_{i=1}^k \boldsymbol{\mu}_{i,\pi}(\tau_i)\right)\right] + (k-1)\mathbb{E}\left[\Lambda\left(\sum_{\tau\in\Omega^k} \boldsymbol{\psi}_k(\tau)\prod_{i=1}^k \boldsymbol{\mu}_{i,\pi'}(\tau_i)\right)\right]$$
$$\geq \sum_{j=1}^k \mathbb{E}\left[\Lambda\left(\sum_{\tau\in\Omega^k} \boldsymbol{\psi}_k(\tau)\boldsymbol{\mu}_{j,\pi}(\tau_j)\prod_{i\neq j} \boldsymbol{\mu}_{i,\pi'}(\tau_i)\right)\right].$$

The first assumption **DEG** ensures that the factor graphs are "sparse" or, formally, locally finite. Yet **DEG** allows for very general degree distributions, including Poisson and power law distributions. Moreover, conditions **SYM** and **BAL** are symmetry conditions. Roughly speaking, they provide that all the values $\omega \in \Omega$ are on the same footing, i.e., there is no semantic preference for any value. Finally condition **POS** can be viewed as a convexity requirement. This assumption is needed for the technical reason of facilitating the interpolation method, a proof technique that we borrow from mathematical physics. The conditions are easily seen to be satisfied in many models of interest including, of course, the stochastic block model and ldgm codes. Crucially, the assumptions can be checked solely in terms of the weight functions; no random graphs considerations are required. [1]

2.3 The mutual information

The main result of the paper vindicates the physicists' hunch that the mutual information between the teacher's ground truth $\boldsymbol{\sigma}^*$ and the data $\boldsymbol{G}^*$ presented to the student is determined by the Bethe free entropy. To state the result we introduce the following generic version of the Bethe functional. Let $(\boldsymbol{\psi}_{k,i})_{k,i}$ be a family of independent random weight functions such that $\boldsymbol{\psi}_{k,i}$ is distributed as $\boldsymbol{\psi}_k$. Further, let $(\boldsymbol{h}_{k,i})_i$ with $\boldsymbol{h}_{k,i} \in [k]$ be a family of independent uniformly distributed indices. Given $\pi \in \mathfrak{P}_*(\Omega)$ let $(\mu_{i,j,\pi})_{i,j\geq 1}$ be a family of independent samples from π. Recalling that $(\hat{\boldsymbol{k}}_i)_{i\geq 1}$ are independent random variables with point masses (1), we define

[1] We point out that **POS** fails to hold in the case of the *assortative* stochastic block model.

$$\mathcal{B}(\pi) = \frac{1}{q}\mathbb{E}\left[\xi^{-\boldsymbol{d}}\Lambda\left(\sum_{\sigma\in\Omega}\prod_{i=1}^{\boldsymbol{d}}\sum_{\tau\in\Omega^{\hat{\boldsymbol{k}}_i}}\boldsymbol{1}\left\{\tau_{\boldsymbol{h}_{\hat{\boldsymbol{k}}_i,i}} = \sigma\right\}\boldsymbol{\psi}_{\hat{\boldsymbol{k}}_i,i}(\tau)\prod_{j\in[\hat{\boldsymbol{k}}_i]\setminus\{\boldsymbol{h}_{\hat{\boldsymbol{k}}_i,i}\}}\boldsymbol{\mu}_{i,j,\pi}(\tau_j)\right)\right] \\ - \frac{\mathbb{E}[\boldsymbol{d}]}{\xi\mathbb{E}[\boldsymbol{k}]}\mathbb{E}\left[(\boldsymbol{k}-1)\Lambda\left(\sum_{\tau\in\Omega^{\boldsymbol{k}}}\boldsymbol{\psi}_{\boldsymbol{k}}(\tau)\prod_{j=1}^{\boldsymbol{k}}\boldsymbol{\mu}_{1,j,\pi}(\tau_j)\right)\right]. \tag{5}$$

The following theorem expresses the mutual information of $\boldsymbol{G}^*$ and $\boldsymbol{\sigma}^*$ given the degrees and the total number of factor nodes as the variational problem of maximising the Bethe functional.

▶ **Theorem 6.** *For any random factor graph model that satisfies the conditions* **DEG**, **SYM**, **BAL** *and* **POS**,

$$\lim_{n\to\infty}\frac{1}{n}I(\boldsymbol{\sigma}^*, \boldsymbol{G}^* \mid \mathfrak{D}) = \log q + \frac{\mathbb{E}[\boldsymbol{d}]}{\xi\mathbb{E}[\boldsymbol{k}]}\mathbb{E}\left[q^{-k_{\boldsymbol{\psi}}}\sum_{\tau\in\Omega^{k_{\boldsymbol{\psi}}}}\Lambda(\boldsymbol{\psi}(\tau))\right] - \sup_{\pi\in\mathfrak{P}_*(\Omega)}\mathcal{B}(\pi) \tag{6}$$

in probability.

The formula (6) is in line with predictions from [33]. Moreover, the results quoted in Section 1 are immediate consequences of Theorem 6.

3 Proof strategy

In this section we survey the proof of Theorem 6. Subsequently we discuss how the strategy compares to prior work, particularly [11]. Throughout we tacitly assume that **DEG**, **SYM**, **BAL** and **POS** are satisfied.

3.1 The partition function

The starting point for computing the mutual information is to observe that this quantity is closely connected to the partition function of $\boldsymbol{G}^*$.

▶ **Proposition 7.** *W.h.p. we have*

$$I(\boldsymbol{\sigma}^*, \boldsymbol{G}^* \mid \mathfrak{D})/n = \log q + \frac{\mathbb{E}[\boldsymbol{d}]}{\xi\mathbb{E}[\boldsymbol{k}]}\mathbb{E}\left[q^{-k_{\boldsymbol{\psi}}}\sum_{\tau\in\Omega^{k_{\boldsymbol{\psi}}}}\Lambda(\boldsymbol{\psi}(\tau))\right] - \mathbb{E}[\log Z(\boldsymbol{G}^*)]/n + o(1).$$

Hence, Proposition 7 reduces our task to computing $\mathbb{E}[\log Z(\boldsymbol{G}^*)]$. This is still a formidable challenge because the logarithm sits inside the expectation; hence, routine techniques such as moment calculations do not bite. Instead we will combine two separate techniques. The first is a coupling argument known as the Aizenman-Sims-Starr scheme. This argument will show that $\mathbb{E}[\log Z(\boldsymbol{G}^*)]$ is upper bounded by $\sup_\pi \mathcal{B}(\pi)$. The second component, the interpolation method, will supply the matching lower bound.

What these techniques have in common is that they both boil down to "local" calculations. That is, we need to assess the impact on the partition function $Z(\boldsymbol{G}^*)$ of a small number of local changes such as addition of a few factor or variable nodes to $\boldsymbol{G}^*$. We will perform these computations by way of a probabilistic argument, namely by tracing how they affect the average weight of a sample from the Boltzmann distribution of $\boldsymbol{G}^*$. The key is a simple but powerful fact that trades as the Nishimori identity.

3.2 The Nishimori identity

To formulate this identity we need to introduce a slightly modified version of the random factor graph model $\boldsymbol{G}^*$. Recall from (4) that $\boldsymbol{G}^*$ was obtained by first drawing $\boldsymbol{\sigma}^*$ uniformly at random and then reweighting the null model $\boldsymbol{G}$ according to the weight of $\boldsymbol{\sigma}^*$. If we combine these two steps the net effect should be, at least roughly, that a specific G comes up with probability proportional to $Z(G)$, as every $\sigma \in \Omega^{V_n}$ provides G with a $\psi_G(\sigma)$ chance of being sampled. Thus, $\boldsymbol{G}^*$ should be roughly equivalent to the random factor graph model $\hat{\boldsymbol{G}}$ defined by $\mathbb{P}\left[\hat{\boldsymbol{G}} = G \mid \mathfrak{D}\right] \propto Z_G \mathbb{P}\left[\boldsymbol{G} = G \mid \mathfrak{D}\right]$. Indeed, this equivalence turns out to be exact if we make one minimal change. Namely, instead of drawing the ground truth $\boldsymbol{\sigma}^*$ uniformly at random, we draw a sample from the distribution $\mathbb{P}\left[\hat{\boldsymbol{\sigma}} = \sigma \mid \mathfrak{D}\right] \propto \mathbb{E}\left[\psi_{\boldsymbol{G}}(\sigma) \mid \mathfrak{D}\right]$ for $\sigma \in \Omega^{V_n}$. The following is an extension of [11, Proposition 3.10] to the present, more general class of factor graph models with given degrees.

▶ **Proposition 8.** *We have*

$$\mathbb{P}\left[\hat{\boldsymbol{G}} = G \mid \mathfrak{D}\right] \mu_G(\sigma) = \mathbb{P}\left[\hat{\boldsymbol{\sigma}} = \sigma \mid \mathfrak{D}\right] \mathbb{P}\left[\boldsymbol{G}^* = G \mid \mathfrak{D}, \boldsymbol{\sigma}^* = \sigma\right]. \tag{7}$$

Furthermore, $\hat{\boldsymbol{\sigma}}$ *and* $\boldsymbol{\sigma}^*$ *as well as* $\boldsymbol{G}^*, \hat{\boldsymbol{G}}$ *are mutually contiguous and* $\mathbb{E}[\log Z_{\boldsymbol{G}^*}] = \mathbb{E}[\log Z_{\hat{\boldsymbol{G}}}] + o(n)$.

The proof of Proposition 8 relies on Bayes' formula combined with a somewhat subtle application of local limit theorems and other probabilistic tools. The details can be found in Section 4 of the full version.

3.3 Degree pruning

A further preparation is degree pruning. Specifically, while in the random factor graph models $\boldsymbol{G}^*$ and $\hat{\boldsymbol{G}}$ may possess degrees as large as $n^{1/2-\varepsilon}$, the following proposition shows that it suffices to prove the main result (6) for bounded degree sequences.

▶ **Proposition 9.** *Assume that for any integer* $L > 0$ *and for any* $\boldsymbol{d}, \boldsymbol{k}$ *such that* $\boldsymbol{d}, \boldsymbol{k} \leq L$ *the statement* (6) *is true. Then* (6) *holds for all* $\boldsymbol{d}, \boldsymbol{k}$ *that satisfy* **DEG** *and for which* $\mathbb{E}\left[\boldsymbol{d}\right], \mathbb{E}\left[\boldsymbol{k}\right] > 0$.

The proof of Proposition 9 is based on concentration inequalities and coupling arguments for bipartite graphs with given degree sequences. Hence, we may assume from here on that $\boldsymbol{d}, \boldsymbol{k}$ are bounded.

3.4 Cavities and couplings

Two of the main steps towards the proof of Theorem 6, the Aizenman-Sims-Starr scheme and the interpolation method, hinge on comparing random factor graphs with slightly different parameters. For example, we will need to compare a random factor graph $\boldsymbol{G}^*$ with n variable and $\mathrm{Po}(\mathbb{E}[\boldsymbol{d}]n/\mathbb{E}[\boldsymbol{k}])$ factor nodes and a factor graph with $n+1$ variable and the commensurate number of $\mathrm{Po}(\mathbb{E}[\boldsymbol{d}](n+1)/\mathbb{E}[\boldsymbol{k}])$ factor nodes. In the classical case of binomial factor graphs as treated in [11] where factor nodes are drawn independently this coupling would be relatively straightforward. Indeed, we could just add a variable node and a few extra factor nodes to the graph with n variables. However, in the present setting of given degrees matters are much more delicate. For instance, how would you set up such a coupling for the d-regular stochastic block model from Section 1.3? Due to the given degrees the graph structure is too rigid to accommodate the necessary local changes.

To cope with this issue we first create a bit of wiggling room for ourselves by slightly reducing the number of factor nodes. This idea has been used in prior work on factor graphs with rigid degree distributions such as [9]. However, matters turn out to be rather more delicate here because we do not just work with purely random factor graphs, but with graphs drawn from the teacher-student model. Thus, we need to take care to meticulously implement the weight shifts in accordance with (4). Hence, for a small but fixed $\varepsilon > 0$ let $\boldsymbol{m}_\varepsilon = \mathrm{Po}((1-\varepsilon)\mathbb{E}[\boldsymbol{d}]n/\mathbb{E}[\boldsymbol{k}])$ be a Poisson variable with a slightly smaller mean than $\boldsymbol{m}$. Because we assume that all degrees are bounded, with probability $1-\exp(-\Omega(n))$ we have $\sum_{i=1}^{n}\boldsymbol{d}_i \geq \sum_{i=1}^{\boldsymbol{m}_\varepsilon}\boldsymbol{k}_i$. In fact, w.h.p. the total variable degree exceeds the total degree of the first $\boldsymbol{m}_\varepsilon$ factor nodes by $\Omega(n)$. Let $\boldsymbol{G}(n,\boldsymbol{m}_\varepsilon)$ be a random factor graph with variable nodes $x_1,\ldots,x_n$ and factor nodes $a_1,\ldots,a_{\boldsymbol{m}_\varepsilon}$ of degrees $\boldsymbol{k}_1,\ldots,\boldsymbol{k}_{\boldsymbol{m}_\varepsilon}$ drawn uniformly at random subject to the condition that the degree of each x_i remains bounded by $\boldsymbol{d}_i$. Thus, some of the variable nodes will likely have a degree strictly smaller than their "target degree" $\boldsymbol{d}_i$. We refer to these variable degrees as *cavities.* Further, given $\sigma \in \Omega^{V_n}$ let $\boldsymbol{G}^*(n,\boldsymbol{m}_\varepsilon,\sigma)$ be the random factor graph obtained as in (4), i.e., with $\mathfrak{D}_\varepsilon$ denoting the σ-algebra generated by the degrees and the total number of factors nodes of $\boldsymbol{G}(n,\boldsymbol{m}_\varepsilon)$ we let

$$\mathbb{P}\left[\boldsymbol{G}^*(n,\boldsymbol{m}_\varepsilon,\sigma)=G \mid \mathfrak{D}_\varepsilon\right] \propto \mathbb{P}\left[\boldsymbol{G}(n,\boldsymbol{m}_\varepsilon)=G \mid \mathfrak{D}_\varepsilon\right]\psi_G(\sigma).$$

The following proposition establishes that we can indeed think of $\boldsymbol{G}^*(n,\boldsymbol{m}_\varepsilon+1,\sigma)$ as being obtained from $\boldsymbol{G}^*(n,\boldsymbol{m}_\varepsilon,\sigma)$ by adding one extra factor node $a_{\boldsymbol{m}_\varepsilon+1}$. Further, for two factor graphs G, G' on the same set of nodes let $G\triangle G'$ be the symmetric difference of their edge sets.

▶ **Proposition 10.** *Assume that $|\sigma^{-1}(\omega)| = n/q + O(\sqrt{n}\log n)$ for all $\omega \in \Omega$. Then there exists a coupling of $\boldsymbol{G}^*(n,\boldsymbol{m}_\varepsilon,\sigma)$ and $\boldsymbol{G}^*(n,\boldsymbol{m}_\varepsilon+1,\sigma)$ such that*

$$\mathbb{P}\left[\boldsymbol{G}^*(n,\boldsymbol{m}_\varepsilon,\sigma)=\boldsymbol{G}^*(n,\boldsymbol{m}_\varepsilon+1,\sigma)-a_{\boldsymbol{m}_\varepsilon+1} \mid \mathfrak{D}_\varepsilon\right] = 1-\tilde{O}(1/n),$$
$$\mathbb{P}\left[|\boldsymbol{G}^*(n,\boldsymbol{m}_\varepsilon,\sigma)\triangle\boldsymbol{G}^*(n,\boldsymbol{m}_\varepsilon+1,\sigma)-a_{\boldsymbol{m}_\varepsilon+1}| > n^{2/3} \mid \mathfrak{D}_\varepsilon\right] = 1-\tilde{O}(1/n^2).$$

There is a similar coupling that accommodates the addition of an extra variable node.

▶ **Proposition 11.** *Assume that $|\sigma^{-1}(\omega)| = n/q + O(\sqrt{n}\log n)$ for all $\omega \in \Omega$. Given the degree $\boldsymbol{\gamma}$ of x_{n+1} in $\boldsymbol{G}^*(n+1,\boldsymbol{m}_\varepsilon+\boldsymbol{\gamma},\sigma)$ then there exists a coupling of $\boldsymbol{G}^*(n,\boldsymbol{m}_\varepsilon,\sigma)$ and $\boldsymbol{G}^*(n+1,\boldsymbol{m}_\varepsilon+\boldsymbol{\gamma},\sigma)$ such that*

$$\mathbb{P}\left[\boldsymbol{G}^*(n,\boldsymbol{m}_\varepsilon,\sigma)=\boldsymbol{G}^*(n+1,\boldsymbol{m}_\varepsilon+\boldsymbol{\gamma},\sigma)-x_{n+1}-\partial x_{n+1} \mid \mathfrak{D}_\varepsilon\right] = 1-\tilde{O}(1/n),$$
$$\mathbb{P}\left[|\boldsymbol{G}^*(n,\boldsymbol{m}_\varepsilon,\sigma)=\boldsymbol{G}^*(n+1,\boldsymbol{m}_\varepsilon+\boldsymbol{\gamma},\sigma)-x_{n+1}-\partial x_{n+1}| > n^{2/3} \mid \mathfrak{D}_\varepsilon\right] = 1-\tilde{O}(1/n^2).$$

The orders $\tilde{O}(1/n), \tilde{O}(1/n^2)$ of the error terms in Propositions 10 and 11 are vital to facilitate the computation of the partition function. On a technical level, the tools that we develop for proving these propositions, and particularly for dealing with the fragile combinatorics of the factor graph models with given degrees, constitute the main novelty of the paper. This is where we most visibly add to and improve over the machinery developed in prior work. The details can be found in Section 4.3 of the full version.

3.5 Aizenman-Sims-Starr and interpolation

Propositions 10 and 11 in combination with a trick known as the Aizenman-Sims-Starr scheme yield the desired upper bound on the partition function.

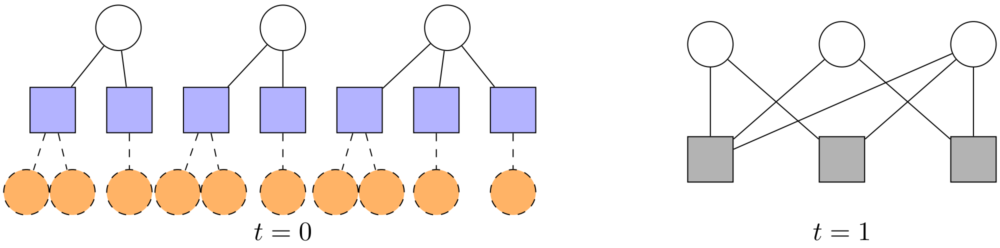


Figure 1 Illustration of the interpolation method at "times" $t = 0$ and $t = 1$.

▶ **Proposition 12.** *We have* $\mathbb{E}[\log Z(\boldsymbol{G}^*)] \leq n \sup_{\pi \in \mathfrak{P}_*(\Omega)} \mathcal{B}(\pi) + o(n)$.

To prove Proposition 12 it suffices to establish the corresponding upper bound for $\boldsymbol{G}^*(n, \boldsymbol{m}_\varepsilon, \boldsymbol{\sigma}^*)$. This is because similar but simpler arguments as in the proof of Proposition 10 show that $\mathbb{E}[\log Z(\boldsymbol{G}^*)] = \mathbb{E}[\log Z(\boldsymbol{G}^*(n, \boldsymbol{m}_\varepsilon, \boldsymbol{\sigma}^*)] + O(\varepsilon n)$. Its proof can be found in Section 13 of the full version. Now, the Aizenman-Sims-Starr scheme for calculating the latter quantity is to write a telescoping sum

$$\mathbb{E}[\log Z(\boldsymbol{G}^*(n, \boldsymbol{m}_\varepsilon, \boldsymbol{\sigma}^*))] \\ = \sum_{N=0}^{n-1} \mathbb{E}[\log Z(\boldsymbol{G}^*(N+1, \boldsymbol{m}_\varepsilon(N+1), \boldsymbol{\sigma}^*_{N+1}))] - \mathbb{E}[\log Z(\boldsymbol{G}^*(N, \boldsymbol{m}_\varepsilon(N), \boldsymbol{\sigma}^*_N))].$$

Hence, it suffices to bound the individual summands on the r.h.s., i.e., the differences

$$\mathbb{E}[\log Z(\boldsymbol{G}^*(n+1, \boldsymbol{m}_\varepsilon(n+1), \boldsymbol{\sigma}^*_{n+1}))] - \mathbb{E}[\log Z(\boldsymbol{G}^*(n, \boldsymbol{m}_\varepsilon(n), \boldsymbol{\sigma}^*_n))]. \tag{8}$$

To this end we couple these two random factor graphs. This is where Propositions 10 and 11 enter the fray. Specifically, we think of both these factor graphs as being obtained from a smaller factor graph $\boldsymbol{G}_0^*$ that with variables nodes $x_1, \ldots, x_n$ and slightly fewer factor nodes than either of the two target factor graphs. Then we obtain $\boldsymbol{G}^*(n, \boldsymbol{m}_\varepsilon(n), \boldsymbol{\sigma}^*_n)$ by adding a few random factors to $\boldsymbol{G}_0^*$. Similarly, we obtain $\boldsymbol{G}^*(n+1, \boldsymbol{m}_\varepsilon(n+1), \boldsymbol{\sigma}^*_{n+1})$ from $\boldsymbol{G}_0^*$ by adding a few new random factor nodes as well as a new variable node x_{n+1} along with a number of adjacent factor nodes. Crucially, Propositions 10 and 11 provide the necessary accuracy to trace the impact of these manipulations on the partition function, and the Bethe functional emerges organically as an upper bound on (8).

To obtain the matching lower bound we seize upon the interpolation method. The basic idea is to set up a family of random factor graph models parametrised by time $t \in [0, 1]$ such that the model at time $t = 1$ coincides with $\boldsymbol{G}^*(n, \boldsymbol{m}_\varepsilon, \boldsymbol{\sigma}^*)$ while the model at time $t = 0$ is so simple that its partition function can be read off easily. In fact, the partition function of the $t = 0$ model turns out to be the Bethe free entropy. To derive the desired lower bound we prove that the derivative of the log-partition function remains non-negative as we increase t. As in the Aizenman-Sims-Starr scheme, the computation of the derivative can be reduced to tracing the impact of local changes. Hence, once more we bring Proposition 10 to bear, this time in combination with the convexity assumption **POS**, to prove the following.

▶ **Proposition 13.** *We have* $\mathbb{E}[\log Z(\boldsymbol{G}^*)] \geq n \sup_{\pi \in \mathfrak{P}_*(\Omega)} \mathcal{B}(\pi) + o(n)$.

Finally, combining Proposition 7–13, we obtain Theorem 6.

3.6 Discussion

There has been a great deal of interest in inference problems on random factor graphs recently. The substantial literature on the stochastic block model alone, much of it devoted to corroborating the predictions from [12], is surveyed in [1, 26]. The literature on applications to modern coding theory until about 2008 is surveyed in [31]; important newer contributions include [20, 21]. Further recent applications include compressed sensing [14, 15], group testing [2, 10], code-division multiple access [17, 30] and the patient zero problem [3]. Apart and beyond this rigorous literature, there is a vast body of work based on either physics techniques such as the cavity method or computer experiments.

The great variety of concrete problems studied individually underscores the potential of generic proof techniques or, even better, general theorems that rigorise these predictions wholesale. A first contribution has been made by Coja-Oghlan, Krzalaka, Perkins and Zdeborová [11], who studied the teacher-student model on binomial random factor graph models. While the general proof strategy that we pursue here is guided by that paper, the present factor graph models are more general by allowing prescribed degree sequences for both the variable and factor nodes. From an application viewpoint this generality is highly desirable because, for example, the quality of an error correcting code or a group testing scheme can be boosted by optimising the degree distribution [31]. However, from a technical viewpoint this generality comes at the cost of losing (conditional) independence among the factor nodes. This issue is well known in random graph theory, where random graphs with given degrees require far more intricate proofs than, e.g., the Erdős–Rényi model [18]. Here, these difficulties are exacerbated by the fact that we study not just the plain random graph, which serves as a our null model, but the reweighted random graph distribution induced by the teacher-student scheme. In effect, many of the steps that were straightforwards in [11] become rather delicate due to stochastic dependencies. The key tool that allows us to cope with these dependencies is Proposition 10. Thus, while we follow the strategy from [11] of combining the Aizenman-Sims-Starr scheme with the interpolation method and although we adopt some of the technical ingredients from that work such as the "pinning lemma", the greater generality of the model leads us to crystallise and improve over the previous approach.

What are alternatives to the present strategy of combining the Aizenman-Sims-Starr scheme with the interpolation method? A classical approach to inference problems on random graphs is the second moment method [5]. Unfortunately, this approach does not generally allow for tight information-theoretic results. The reason is that the precise formula for the mutual information or the information-theoretic threshold in, e.g., the stochastic block model comes in terms of the optimiser of the Bethe free entropy functional. The distribution π where the maximum is obtained mirrors the outcome of a complicated message passing process. Intuitively, π is an idealised version of the empirical distribution of Belief Propagation messages that whiz around the factor graph upon convergence when launched from either a uniform initialisation or from the completely polarised initialisation corresponding to the ground truth. In some examples this fixed point can be characterised precisely and, unsurprisingly, turns out to be anything but trivial [6]. But we cannot expect the expressiveness required for such a complicated object from a plain second moment computation. A second conceptually elementary approach is to actually compute the message passing fixed point by hand, e.g., via the contraction method. But due to the intricacy of the calculations this method has been pushed through in only a few special cases [27].

Further powerful techniques include spatial coupling [16] and the adaptive interpolation method [7]. Both potentially allow for precise results. The basic idea behind spatial coupling is to convert the given model into a factor graph model with a superimposed geometric structure. A plus of spatial coupling is that it sometimes allows for better inference algorithms. A disadvantage is that the construction has to be carried out case-by-case. By comparison, the adaptive interpolation method has the advantage of being technically relatively clean. However, at least on sparse models its combinatorial nuts and bolts appear to be roughly equivalent to the combination of Aizenman-Sims-Starr and the interpolation argument used here. Furthermore, the latter approach has the merit of being closer in spirit to the physicists' cavity calculation. In addition, at this time the adaptive interpolation method has not been extended to models with given general degree sequences.

Further, there has been quite some work on dense random factor graph models where each variable appears in a constant fraction of factor nodes. Examples are spiked matrix/tensor models [8] or models of neural networks such as the Hopfield model [4, 23]. These methods are closer in nature to the classical Sherrington-Kirkpatrick model [28]. It seems fair to say that more is known about dense models than sparse ones because certain central limit theorem-like simplifications arise. In some cases, the Bethe variational principle reduces to a finite-dimensional or even scalar optimisation problem [13, 22].

To conclude we note that the study of inference problems typically comes in two instalments: an information-theoretic view that asks for thresholds beyond which in principle sufficient information is available to form a non-trivial estimate of the ground truth and an algorithmic view interested in polynomial-time algorithms. While the two perspectives might appear disparate at first glance, information-theoretic results on inference problems like in this paper in combination with tools such as spatial coupling have in the past led to efficient algorithms capable of attaining the information-theoretic thresholds [10, 15]. We view this as an exciting avenue for future research.

References

1 E. Abbe and A. Montanari. Conditional random fields, planted constraint satisfaction and entropy concentration. *Theory of Computing*, 11:413–443, 2015.

2 M. Aldridge, O. Johnson, and J. Scarlett. Group testing: an information theory perspective. *Foundations and Trends in Communications and Information Theory*, 2019.

3 F. Altarelli, A. Braunstein, L. Dall'Asta, A. Lage-Castellanos, and R. Zecchina. Bayesian inference of epidemics on networks via belief propagation. *Physical review letters*, 112:118701, 2014.

4 D. Amit, H. Gutfreund, and H. Sompolinsky. Storing infinite numbers of patterns in a spin-glass model of neural networks. *Physical Review Letters*, 55:1530, 1985.

5 J. Banks, C. Moore, J. Neeman, and P. Netrapalli. Information-theoretic thresholds for community detection in sparse networks. *Proc. 29th COLT*, pages 383–416, 2016.

6 V. Bapst, A. Coja-Oghlan, S. Hetterich, F. Rassmann, and D. Vilenchik. The condensation phase transition in random graph coloring. *Communications in Mathematical Physics*, 341:543–606, 2016.

7 J. Barbier, C. Chan, and N. Macris. Mutual information for the stochastic block model by the adaptive interpolation method. *Proc. IEEE International Symposium on Information Theory*, pages 405–409, 2019.

8 J. Barbier and N. Macris. The adaptive interpolation method for proving replica formulas. applications to the Curie–Weiss and Wigner spike models. *Journal of Physics A: Mathematical and Theoretical*, 52:294002, 2019.

9 A. Coja-Oghlan, A. Ergür, P. Gao, S. Hetterich, and M. Rolvien. The rank of sparse random matrices. *Proc. 31st SODA*, pages 579–591, 2020.

10 A. Coja-Oghlan, O. Gebhard, M. Hahn-Klimroth, and P. Loick. Optimal group testing. *Proceedings of Machine Learning Research (COLT)*, 2020.

11 A. Coja-Oghlan, F. Krzakala, W. Perkins, and L. Zdeborová. Information-theoretic thresholds from the cavity method. *Advances in Mathematics*, 333:694–795, 2018.

12 A. Decelle, F. Krzakala, C. Moore, and L. Zdeborová. Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications. *Phys. Rev. E*, 84:066106, 2011.

13 M. Dia, N. Macris, F. Krzakala, T. Lesieur, and L. Zdeborová. Mutual information for symmetric rank-one matrix estimation: A proof of the replica formula. *Advances in Neural Information Processing Systems*, pages 424–432, 2016.

14 D. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52:1289–1306, 2006.

15 D. Donoho, A. Javanmard, and A. Montanari. Information-theoretically optimal compressed sensing via spatial coupling and approximate message passing. *IEEE Transactions on Information Theory*, 59:7434–7464, 2013.

16 A. Giurgiu, N. Macris, and R. Urbanke. Spatial coupling as a proof technique and three applications. *IEEE Transactions on Information Theory*, 62:5281–5295, 2016.

17 D. Guo and C. Wang. Multiuser detection of sparsely spread cdma. *IEEE journal on selected areas in communications*, 26:421–431, 2008.

18 S. Janson, T. Łuczak, and A. Rucinski. Random graphs. *John Wiley & Sons*, 45, 2011.

19 F. Krzakala, A. Montanari, F. Ricci-Tersenghi, G. Semerjian, and L. Zdeborova. Gibbs states and the set of solutions of random constraint satisfaction problems. *Proc. National Academy of Sciences*, 104:10318–10323, 2007.

20 S. Kudekar, T. Richardson, and R. Urbanke. Spatially coupled ensembles universally achieve capacity under belief propagation. *IEEE Transactions on Information Theory*, 59:7761–7813, 2013.

21 S. Kumar, A. Young, N. Macris, and H. Pfister. Threshold saturation for spatially coupled ldpc and ldgm codes on bms channels. *IEEE Transactions on Information Theory*, 60:7389–7415, 2014.

22 M. Lelarge and L. Miolane. Fundamental limits of symmetric low-rank matrix estimation. *Conference on Learning Theory (COLT)*, pages 1297–1301, 2017.

23 M. Mézard. Mean-field message-passing equations in the Hopfield model and its generalizations. *Physical Review E*, 95:022117, 2017.

24 M. Mézard and A. Montanari. Information, physics and computation. *Oxford University Press*, 2009.

25 A. Montanari. Tight bounds for ldpc and ldgm codes under map decoding. *IEEE Transactions on Information Theory*, 51:3221–3246, 2005.

26 C. Moore. The computer science and physics of community detection: landscapes, phase transitions, and hardness. *Bull. EATCS*, 121, 2017.

27 E. Mossel, J. Neeman, and A. Sly. Reconstruction and estimation in the planted partition model. *Probability Theory and Related Fields*, 162:431–461, 2015.

28 D. Panchenko. The Sherrington-Kirkpatrick model. *Springer*, 2013.

29 J. Pearl. Probabilistic reasoning in intelligent systems: networks of plausible inference. *Elsevier*, 2014.

30 J. Raymond and D. Saad. Sparsely spread cdma – a statistical mechanics-based analysis. *Journal of physics A: mathematical and theoretical*, 40:12315, 2007.

31 T. Richardson and R. Urbanke. Modern coding theory. *Cambridge University Press*, 2012.

32 J. van den Brand and N. Jaafari. The mutual information of ldgm codes. *arXiv*, 2017. arXiv:1707.04413.

33 L. Zdeborová and F. Krzakala. Phase transition in the coloring of random graphs. *Phys. Rev. E*, 76:031131, 2007.

34 L. Zdeborová and F. Krzakala. Statistical physics of inference: thresholds and algorithms. *Advances in Physics*, 65:453–552, 2016.

The Edit Distance to k-Subsequence Universality

Joel D. Day
Loughborough University, UK

Pamela Fleischmann
Computer Science Department, Universität Kiel, Germany

Maria Kosche
Computer Science Department, Universität Göttingen, Germany

Tore Koß
Computer Science Department, Universität Göttingen, Germany

Florin Manea
Computer Science Department, Universität Göttingen, Germany
Campus-Institut Data Science, Göttingen, Germany

Stefan Siemer
Computer Science Department, Universität Göttingen, Germany

Abstract

A word u is a subsequence of another word w if u can be obtained from w by deleting some of its letters. In the early 1970s, Imre Simon defined the relation $\sim_k$ (called now Simon-Congruence) as follows: two words having exactly the same set of subsequences of length at most k are $\sim_k$-congruent. This relation was central in defining and analysing piecewise testable languages, but has found many applications in areas such as algorithmic learning theory, databases theory, or computational linguistics. Recently, it was shown that testing whether two words are $\sim_k$-congruent can be done in optimal linear time. Thus, it is a natural next step to ask, for two words w and u which are not $\sim_k$-equivalent, what is the minimal number of edit operations that we need to perform on w in order to obtain a word which is $\sim_k$-equivalent to u.

In this paper, we consider this problem in a setting which seems interesting: when u is a k-subsequence universal word. A word u with $\mathrm{alph}(u) = \Sigma$ is called k-subsequence universal if the set of subsequences of length k of u contains all possible words of length k over Σ. As such, our results are a series of efficient algorithms computing the edit distance from w to the language of k-subsequence universal words.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Design and analysis of algorithms

Keywords and phrases Subsequence, Scattered factor, Subword, Universality, k-subsequence universality, Edit distance, Efficient algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.25

Related Version *Full Version*: `https://arxiv.org/abs/2007.09192`

Funding The work of the four authors from Göttingen was supported by the DFG-grant 389613931.

1 Introduction

A word v is a subsequence (also called scattered factor or subword) of a word w if there exist (possibly empty) words $x_1, \ldots, x_{\ell+1}$ and $v_1, \ldots, v_\ell$ such that $v = v_1 \ldots v_\ell$ and $w = x_1 v_1 \ldots x_\ell v_\ell x_{\ell+1}$. That is, v is obtained from w by removing some of its letters.

The study of the relationship between words and their subsequences is a central topic in combinatorics on words and string algorithms, as well as in language and automata theory (see, e.g., the chapter *Subwords* by J. Sakarovitch and I. Simon in [55, Chapter 6] for an overview

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 25; pp. 25:1–25:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of the fundamental aspects of this topic). The concept of subsequences and its generalisations play an important role in various areas of theoretical computer science. For instance, in logic of automata theory, subsequences are used in the context of piecewise testability [60, 61], in particular to the height of piecewise testable languages [39, 40, 41], subword order [31, 44, 43], or downward closures [66]. In combinatorics on words, many concepts were developed around the idea of counting the occurrences of particular subsequences of a word, such as the k-binomial equivalence [54, 25, 46, 45], subword histories [59], and Parikh matrices [49, 56]. In the area of algorithms, subsequences appear, e.g., in classical problems such as the longest common subsequence [5, 10, 12], the shortest common supersequence [47], or the string-to-string correction [65]. From a practical point of view, subsequences are useful in scenarios related to bioinformatics, as well as in other areas where they model corrupted or lossy representations of an original string, see [57].

A major area of research related to subsequences is the study of the set of all subsequences of bounded length of a word, initiated by Simon in his PhD thesis [60]. In particular, Simon defined and studied (see [61, 55]) the relation $\sim_k$ (now called the Simon-Congruence) between words having exactly the same set of subsequences of length at most k, and used it in the study of piecewise testable languages, a class of regular languages with applications in learning theory, databases theory, or linguistics (see, e.g., [41] and the references therein). The surveys [51, 52] overview some of the extensions of Simon's seminal work from 1972 in various areas related to automata theory. Moreover, $\sim_k$ is a well-studied relation in the area of string algorithms, too. The problems of deciding whether two given words are $\sim_k$-equivalent, for a given k, and to find the largest k such that two given words are $\sim_k$-equivalent (and their applications) were heavily investigated in the literature, see, e.g., [34, 27, 62, 63, 18, 24] and the references therein. This year, optimal solutions were given for both these problems [6, 28]. In [6] it was shown how to compute the shortlex normal form of a given word in linear time, i.e., the minimum representative of a $\sim_k$-equivalence class w.r.t. shortlex ordering. This can be directly applied to test whether two words are $\sim_k$-equivalent: they need to have the same shortlex normal form. In [28], a data structure, called the Simon-Tree, was used to represent the equivalence classes induced by $\sim_k$ on the set of suffixes of a word, for all possible values of k, and then, given two words, a correspondence between their Simon-Trees was constructed to compute in linear time the largest k for which they are $\sim_k$-equivalent.

Motivation. As described above, asymptotically optimal algorithms are known for deciding whether two words w and u are $\sim_k$-equivalent. Thus, similarly to the case of other relations on strings (e.g., [4, 9]), it is natural to ask, for two words w and u, which are not $\sim_k$-equivalent, what is the minimal number of edit operations (edits, for short) that we need to perform on them in order to obtain two $\sim_k$-equivalent words. The edits we consider are the usual letter-insertion, -deletion, -substitution, and the scenario we assume is the following: we edit one word only (say w) and attempt to reach, with a minimal number of edits, an intermediate word which has the same subsequences of length k as the second input word (namely u).

This formulation is essentially an instance of a word-to-language edit distance problem, in which we wish to compute the distance between w and the language $L_{u,k}$ of words which are $\sim_k$-equivalent to u. It is well documented that word-to-language edit distance problems, alongside the classical word-to-word and also the language-to-language variants, are well motivated and have consequently been well studied (see, e.g., [64, 50, 33, 11, 38, 15, 16]). For instance, the authors of [13] write: *"the edit distance provides a quantitative measure of "how far apart" are two words, or a word from a language, or two languages, and it forms the basis for quantitatively comparing sequences, a problem that arises in many different* *areas, such as error-correcting codes, natural language processing, and computational biology; similarly, the edit distance between languages forms the foundations of a quantitative approach to verification".*

In our case, the languages $L_{u,k}$ are regular. In particular, for a given subsequence v of length k of u, we can easily construct a DFA recognising the language of all words containing v as a subsequence. Consequently, a finite automaton accepting $L_{u,k}$ can be obtained as a boolean combination of these DFAs. In fact, for a positive integer k, the set of all languages which can be written as the union of several languages $L_{u,k}$, where u are words of a finite set, is the class of k-piecewise testable languages [60, 61, 55], an important class of regular languages, with deep connections to logic and semigroup theory. Therefore, if we take as input the word w and the language $L_{u,k}$ given as an automaton $A_{u,k}$ with q states, we can solve our distance problem in time $O(|w|q^2)$ [64, 3]. However, this is not necessarily efficient, since even when $A_{u,k}$ is a minimal NFAs accepting $L_{u,k}$, the number q of states can be exponential in the size of the alphabet (and hence in the length of u; see [19]). Consequently, if we consider the input to be (w, u, k) rather than $(w, A_{u,k})$, the exact complexity remains unclear. We can, however, guarantee inclusion in NP as we can trivially rewrite w into the shortest word $u \in L_{u,k}$ using at most $|w| + |u|$ edits.

It is also worth pointing out that the order of w and u in the input matters: the number of edits necessarily applied to w order to reach a word w' such that $w' \sim_k u$ holds, is not generally equal to the number of edits needed to apply on u in order to reach a word u' such that $u' \sim_k w$. Consider, for example, the words $w = aba$, $u = aaabbbaaa$, and $k = 2$. We need one insertion to transform w into $abab$, which is $\sim_2$-equivalent to u, but we need two deletions to transform u into $aaabaaa$, which is $\sim_2$-equivalent to w. An intuitive explanation for this is that w is closer w.r.t. the edit distance to the set $L_{u,k}$ of words which are $\sim_k$-equivalent to u than u is to the set $L_{w,k}$ of words which are $\sim_k$-equivalent to w, and we only need to edit each of our words until it reaches the word which is closest to them from the respective sets.

Essentially, we are considering the word-to-language edit distance problem for regular languages (in fact, piecewise testable languages) which admit a particularly succinct representation: a single word u. One way to generalise this is to consider the edit distance from a word to the closure of a given language under $\sim_k$. The problem remains decidable when considering the closures of regular or context-free languages (the regular case can be solved in nondeterministic polynomial time when k is a constant, see [19]). On the other hand, we have already mentioned how taking the closure under $\sim_k$ can result in an exponential blow-up in the size of the representation of the language. Going in the other direction, one of the most natural restrictions is to consider only words u over an alphabet Σ for which all length-k subsequences over Σ occur, called *k-subsequence universal words* (called, for short, k-universal words) w.r.t. the alphabet Σ. This case is also among the ones for which the corresponding automata for $L_{u,k}$ may be exponentially large (see [19]), remaining thus non-trivial. This restriction forms the focus of our paper.

The focus of our paper. In some cases, the edit distance problem we introduced above admits an input-specification where the target language is defined in a way which is both easier-to-use and more succinct. One of these cases is the already mentioned language of k-subsequence universal words w.r.t. an alphabet $\Sigma = \{1, \ldots, \sigma\}$. While this language can be defined by a word $(1 \cdot 2 \cdots \sigma)^k$ of length $k\sigma$ or by an NFA with $\Theta(2^\sigma)$ states, it can also be simply specified by the number k and the alphabet Σ (or even only the size of this alphabet).

The main contribution of our paper, described below, is the study of the following problem: given a word w and a number k, compute the minimum number of edits we need to apply to w in order to obtain a k-universal word w.r.t. $\mathrm{alph}(w)$ (see [19] for a discussion on why the alphabet Σ used in the definition of universality is chosen here to be the set $\text{alph}(w)$ of letters occurring in the input word w). As such, we are interested in the edit distance from the input word w to the set of k-universal words w.r.t. $\text{alph}(w)$. We give a series of efficient algorithms showing how to solve this problem.

This investigation seems interesting to us as, on the one hand, the language of k-universal words plays an interesting role in the picture described in the **Motivation** section above. On the other hand, the class of languages of k-universal words occurs prominently in the study of the combinatorial and language theoretic properties of subsequences and piecewise testable languages. Indeed, in [39, 40, 41] the authors define and use the notion of k-rich words in relation to the study of the height of piecewise testable languages. The class of k-rich words coincides with that of k-subsequence universal words, which were further investigated, from a combinatorial point of view, in [20, 6]. Moreover, the idea of universality is quite important in formal languages and automata theory. The classical universality problem (see, e.g., [36]) is whether a given language L (over an alphabet Σ, specified by an automaton or grammar) is equal to Σ^*. The works [53, 42, 29] and the references therein discuss many variants of and results on the universality problem for various language generating and accepting formalisms. The universality problem was considered for words [48, 21] and partial words [14, 30] w.r.t. their factors. More precisely, one is interested in finding, for a given ℓ, a word w over an alphabet Σ, such that each word of length ℓ over Σ occurs exactly once as a contiguous factor of w. De Bruijn sequences [21] fulfil this property and have many applications in computer science or combinatorics, see [14, 30] and the references therein. It is worth noting that in the case of factor-universality it makes sense to ask for words where each factor occurs exactly once, but in the case of subsequence universality this is a trivial restriction, as in each long-enough word there will be subsequences occurring more than once [6].

As such, investigating k-subsequence universality from an algorithmic perspective is motivated by, fits in, and even enriches this well-developed and classical line of research.

Our results. The maximum k for which a word w is k-universal is called *the universality index* of w, and denoted $\iota(w)$. Firstly, we note that when we want to increase the universality index of a word by edits, it is enough to use only insertions. Similarly, when we want to decrease the universality index of a word, it is enough to consider deletions. So, to measure the edit distance to the class of k-subsequence universal words, for a given k, it is enough to consider either insertions or deletions. However, changing the universality of a word by substitutions (both increasing and decreasing it) is interesting in itself as one can see the minimal number of substitutions needed to transform a word w into a k-universal word as the *Hamming distance* [32] between w and the set of k-universal words. Thus, we consider all these operations independently and propose efficient algorithms computing the minimal number of insertions, deletions, and substitutions, respectively, needed to apply to a given word w in order to reach the class of k-universal words (w.r.t. the alphabet of w), for a given k. The time needed to compute these numbers is $O(nk)$ in the case of deletions and substitutions, as well as in the case of insertions when $k \leq n$ (for larger values of k it is just the time complexity of computing $k\sigma - n$, which is the value of the distance in that case). These algorithms are presented in the Section 4, and work in optimal linear time for constant k.

These algorithms are based, like most edit distance algorithms, on a dynamic programming approach. However, implementing such an approach within the time complexities stated above does not seem to follow directly from the known results on the word-to-word or word-to-language edit distance. In particular, we do not explicitly construct any k-universal word nor any representation (e.g., automaton or grammar) of the set of k-universal words, when computing the distance from the input word w to this set. Rather, we obtain the k-universal word which is closest w.r.t. edit distance to w as a byproduct of our algorithms. In our approach, we first develop (Section 3) several efficient data structures (most notably Lemma 3.5). Then (Section 4), for each of the considered operations, we make several combinatorial observations, allowing us to restrict the search space of our algorithms, and creating a framework where our data structures can be used efficiently.

Finally (in Section 5), we give algorithms running in $(n \log^{O(1)} \sigma)$-time computing the minimum number of insertions (respectively, substitutions) we need to apply to w in order to obtain a k-universal word, with $k > \iota(w)$. These algorithms rely heavily on the fact that computing the edit distance to k-universality can be reformulated, in this case, as computing the path of length k of minimum weight in a weighted DAG with the Monge property. In particular, these algorithms provide optimal linear-time solutions for our problem in the case of increasing the universality-index of words over constant-size alphabets.

For space reasons, some proofs, examples, and pseudocode for the algorithms are given in the full version of this paper [19]. A discussion on lower bounds is also given in [19].

2 Preliminaries

Let $\mathbb{N}$ be the set of natural numbers and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. Define for $i, j \in \mathbb{N}_0$ with $i < j$ the interval $[i : j]$ as $\{i, i+1, \ldots, j-1, j\}$. An alphabet Σ is a nonempty finite set of symbols called *letters*. A *word* is a finite sequence of letters from Σ, thus an element of the free monoid Σ^*. Let $\Sigma^+ = \Sigma^* \backslash \{\varepsilon\}$, where ε is the empty word. The *length* of a word $w \in \Sigma^*$ is denoted by $|w|$. Let Σ^k be the set of all words from Σ^* of length exactly k. A word $u \in \Sigma^*$ is a *factor* of $w \in \Sigma^*$ if $w = xuy$ for some $x, y \in \Sigma^*$. If $x = \varepsilon$ (resp. $y = \varepsilon$), u is called a *prefix* (resp. *suffix*) of w. The i^{th} letter of $w \in \Sigma^*$ is denoted by $w[i]$ for $i \in [1 : |w|]$. Set $w[i : j] = w[i]w[i+1] \cdots w[j]$ for $1 \leq i \leq j \leq |w|$, $|w|_{\mathtt{a}} = |\{i \in [1 : |w|] \,|\, w[i] = \mathtt{a}\}|$, and $\mathrm{alph}(w) = \{\mathtt{a} \in \Sigma \,||w|_{\mathtt{a}} > 0\}$ for $w \in \Sigma^*$. We can now introduce the notion of subsequence.

▶ **Definition 2.1.** *A word $v = v_1 \cdots v_\ell \in \Sigma^*$ is a* subsequence *of $w \in \Sigma^*$ if there exist $x_1, \ldots, x_{\ell+1} \in \Sigma^*$ with $w = x_1 v_1 \cdots x_\ell v_\ell x_{\ell+1}$. Let* $\mathrm{Subseq}(w)$ *be the set of all subsequences of w and define* $\mathrm{Subseq}_k(w) = \mathrm{Subseq}(w) \cap \Sigma^k$*, the set of subsequences of w of length $k \in \mathbb{N}$.*

For $k \in \mathbb{N}_0$, $\mathrm{Subseq}_k(w)$ is called the k-spectrum of w. Simon [61] defined the congruence $\sim_k$ in which $u, v \in \Sigma^*$ are congruent if they have the same k-spectrum. As introduced in [6] the notion of k-universality of a word over Σ denotes its property of having Σ^k as k-spectrum.

▶ **Definition 2.2.** *A word $w \in \Sigma^*$ is called* k-subsequence universal *(w.r.t. Σ, for short* k-universal*), for $k \in \mathbb{N}$, if* $\mathrm{Subseq}_k(w) = \Sigma^k$*. We abbreviate 1-universal by* universal*. The* universality-index $\iota(w)$ *of $w \in \Sigma^*$ is the largest k such that w is k-universal.*

If $\iota(w) = k$ then w is ℓ-universal for all $\ell \leq k$. Notice that k-universality is always w.r.t. a given alphabet Σ: the word $\mathtt{abcba}$ is universal for $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ but it is not universal for $\Sigma \cup \{\mathtt{d}\}$. In each algorithm presented in this paper, whenever we discuss about the universality index of some word (factor of the input word, or obtained from the input word via edit operations), we compute it with respect to the alphabet of the input word w.

The notion of ℓ-universality coincides to that of ℓ-richness introduced in [40, 41]. We use the name *ℓ-universality* rather than *ℓ-richness*, as richness of words is also used with other meanings, see, e.g., [23, 22]. We recall the arch factorisation, introduced by Hebrard [34].

▶ **Definition 2.3** ([34]). *For $w \in \Sigma^*$ the* arch factorisation *of w is $w = \mathrm{ar}_w(1) \cdots \mathrm{ar}_w(k) r(w)$ for some $k \in \mathbb{N}_0$ where $\mathrm{ar}_w(i)$ is universal, the last letter of $\mathrm{ar}_w(i)$, namely $\mathrm{ar}_w(i)[|\mathrm{ar}_w(i)|]$, does not occur in $\mathrm{ar}_w(i)[1 : |\mathrm{ar}_w(i)| - 1]$ for all $i \in [1 : k]$, and alph$(r(w)) \subset \Sigma$. The words $\mathrm{ar}_w(i)$ are called* arches *of w, $r(w)$ is called the* rest.

If the arch factorisation of w contains $k \in \mathbb{N}_0$ arches, then $\iota(w) = k$. The following immediate theorem based on the work of Simon [61] completely characterises the set of k-subsequence universal words, based on Hebrard's arch factorisation.

▶ **Theorem 2.4.** *The word $w \in \Sigma^*$ is k-universal if and only if there exist the words v_i, with $i \in [1 : k]$, such that $v_1 \cdots v_k = w$ and alph$(v_i) = \Sigma$ for all $i \in [1 : k]$.*

General algorithmic framework. The further preliminaries regard algorithms. The computational model we use is the standard unit-cost RAM with logarithmic word size: for an input of size n, each memory-word can hold $\log n$ bits. In all the problems, we assume that we are given a word w, with $|w| = n$, over an alphabet $\Sigma = \{1, 2, \ldots, \sigma\}$, with $|\Sigma| = \sigma \leq n$. This is a common assumption in string algorithms: the input alphabet is said to be *an integer alphabet*. For a more detailed general discussion on this model see, e.g., [17] or the full version of our paper [19]. We also assume that our input words contain at least two distinct letters, otherwise all the problems we consider become trivial.

The following theorem was proven in [6] and shows that the universality index and the arches can be obtained in linear time w.r.t. the word length.

▶ **Theorem 2.5.** *Let w be a word, with $|w| = n$, alph$(w) = \Sigma$, and $\Sigma = \{1, 2, \ldots, \sigma\}$. We can compute in linear time $O(n)$ the arch factorisation of w, and, as such, $\iota(w)$.*

More precisely, one can compute greedily, in linear time, the following decomposition of w into arches $w = u_1 \cdots u_k$ as follows:

- u_1 is the shortest prefix of w with $\mathrm{alph}(u_1) = \Sigma$, or $u_1 = w$ if there is no such prefix;
- if $u_1 \cdots u_i = w[1 : t]$, for some $i \in [1 : k]$ and $t \in [1 : n]$, we compute u_{i+1} as the shortest prefix of $w[t + 1 : n]$ with $\mathrm{alph}(u_{i+1}) = \Sigma$, or $u_{i+1} = w[t + 1 : n]$ if there is no such prefix.

In our results, we will use two well-known efficient data structures.

First, the *interval union-find* data structure [26, 37].

▶ **Definition 2.6** (Interval union-find). *Let $V = [1 : n]$ and S a set with $S \subseteq V$. The elements of $S = \{s_1, \ldots, s_p\}$ are called borders and are ordered $0 = s_0 < s_1 < \ldots < s_p < s_{p+1} = n + 1$ where s_0 and s_{p+1} are generic borders. For each border s_i, we define $V(s_i) = [s_{i-1} + 1 : s_i]$ as an induced interval. Now, $P(S) := \{V(s_i) \mid s_i \in S\}$ gives an ordered partition of the set V. The* interval union-find structure *maintains the partition $P(S)$ under the operations:*

- *For $u \in V$,* `find`(u) *returns $s_i \in S \cup \{n + 1\}$ such that $u \in V(s_i)$.*
- *For $u \in S$,* `union`(u) *updates the partition $P(S)$ to $P(S \setminus \{u\})$. That is, if $u = s_i$, then we replace the intervals $V(s_i)$ and $V(s_{i+1})$ by the single interval $[s_{i-1} + 1 : s_{i+1}]$ and update the partition so that further* `find` *and* `union` *operations can be performed.*

When using the data structure from Definition 2.6, we employ a less technical language: we describe the intervals stored initially in the structure, and then the unions are made between adjacent intervals. We can enhance the data structures so that the `find` operation returns both borders of the interval containing the searched value, as well as some other satellite data we decide to associate to that interval. The following lemma was shown in [26, 37].

▶ **Lemma 2.7.** *One can implement the interval union-find data structure, such that, the initialisation of the structures followed by a sequence of $m \in O(n)$ union and find operations can be executed in $\mathcal{O}(n)$ time and space.*

Finally, we recall the *Range Minimum Query* problem, and the main result on it [8].

▶ **Definition 2.8** (RMQ). *Let A be an array with n elements from a well-ordered set. We define* range minimum queries RMQ_A *for the array of A:* $\mathrm{RMQ}_A(i,j) = \arg\min\{A[t] \mid t \in [i:j]\}$, *for $i,j \in [1:n]$. That is, $\mathrm{RMQ}_A(i,j)$ is the position of the smallest element in the subarray $A[i:j]$; if there are multiple positions containing this smallest element, $\mathrm{RMQ}_A(i,j)$ is the leftmost of them. (When it is clear from the context, we drop the subscript A).*

▶ **Lemma 2.9.** *Let A be an array with n integer elements. One can preprocess A in $O(n)$ time and produce data structures allowing to answer in constant time* range minimum queries $\mathrm{RMQ}_A(i,j)$, *for any $i,j \in [1:n]$.*

3 Toolbox

In this section we present data structures which will be decisive in obtaining efficient solutions for the approached problems. Our running example will be the word $w = \texttt{bananaban}$, on which we illustrate some of the notions we define here. Full details are given in [19].

For a word w over an alphabet Σ, a position j of w, and a letter $a \in \Sigma$ which occurs in $w[1:j]$, let $\mathrm{last}_j[a] = \max\{i \leq j \mid w[i] = a\}$, the last position where a occurs before j; if a does not occur in $w[1:j]$ or for $j = 0$, then, by convention, $\mathrm{last}_j[a] = |w| + 1$. Let $S_j = \{\mathrm{last}_j[a] \mid a \in \mathrm{alph}(w[1:j])\}$. If i,j are two positions of w, let $\Delta(i,j)$ be the number of distinct letters occurring in $w[i:j]$, i.e., $\Delta(i,j) = |\mathrm{alph}(w[i:j])|$; if $i > j$, then $\Delta(i,j) = 0$. For a position i of w, and a letter $a \in \Sigma$, let $d_i[a] = \Delta(\mathrm{last}_i[a], i)$.

▶ **Lemma 3.1.** *Let w be a word, with $|w| = n$, $alph(w) = \Sigma$, and $\Sigma = \{1, 2, \ldots, \sigma\}$. We can compute in $O(n)$ the values $\Delta(1,\ell)$, for all $\ell \in [1:n]$.*

▶ **Lemma 3.2.** *Let w be a word, with $|w| = n$, $alph(w) = \Sigma$, and $\Sigma = \{1, 2, \ldots, \sigma\}$. We can compute in $O(n)$ the values $\Delta(i - \sigma + 1, i)$, for all $i \in [\sigma:n]$.*

▶ **Lemma 3.3.** *Let w be a word, with $|w| = n$, $alph(w) = \Sigma$, and $\Sigma = \{1, 2, \ldots, \sigma\}$. We can compute in $O(n)$ time $\mathrm{last}_{j\sigma+1}[a]$ and $d_{j\sigma+1}[a]$, for all $a \in \Sigma$ and all integers $1 \leq j \leq (n-1)/\sigma$.*

For $w = \texttt{bananaban}$, we have $|w| = 9$ and $\sigma = 3$. In Lemma 3.1 we compute $\Delta(1,1) = 1$, $\Delta(1,2) = 2$, and $\Delta(1,\ell) = 3$ for $\ell \in [3:9]$. In Lemma 3.2 we compute $\Delta(1,3) = 3$, $\Delta(2,4) = \Delta(3,5) = \Delta(4,6) = 2$, $\Delta(5,7) = 3$, $\Delta(6,8) = 2$, and $\Delta(7,9) = 3$. In Lemma 3.3 we compute the arrays $\mathrm{last}_4[\cdot]$ and $\mathrm{last}_7[\cdot]$. We get: $\mathrm{last}_4[\texttt{a}] = 4$, $\mathrm{last}_4[\texttt{b}] = 1$, $\mathrm{last}_4[\texttt{n}] = 3$, and $\mathrm{last}_7[\texttt{a}] = 6$, $\mathrm{last}_7[\texttt{b}] = 7$, $\mathrm{last}_7[\texttt{n}] = 5$. Therefore, $S_4 = \{1,3,4\}$, $S_7 = \{5,6,7\}$, and $d_4[\texttt{a}] = 1$, $d_4[\texttt{b}] = 3$, $d_4[\texttt{n}] = 2$, $d_7[\texttt{a}] = 2$, $d_7[\texttt{b}] = 1$, $d_7[\texttt{n}] = 3$.

For a word w and a position i of w, let $\mathrm{univ}[i] = \max\{j \mid w[j:i]$ is universal$\}$. That is, for the position i we compute the shortest universal word ending on that position. If there is no universal word ending on position i we set $\mathrm{univ}[i] = 0$.

Further, if $n = |w|$, let $V_w = \{\mathrm{univ}[i] \mid 1 \leq i \leq n\}$. In V_w we collect the starting positions of the shortest universal words ending at each position of the word w. Now, for $j \in V_w$, let $L_j = \{i \mid \mathrm{univ}[i] = j\}$; in other words, we group together the positions i of w for which the shortest universal word ending on i starts on some position j. Note that $L_0 = \{i \mid w[1:i]$ is not universal$\}$, i.e., the positions of w where no universal word ends.

Several observations are immediate: for $i \in L_j$, $i' \in L_{j'}$, we have $i \leq i'$ if and only if $j \leq j'$. As each position i of w belongs to a set L_j, for some $j \in V_w$, we get that $\{L_j \mid j \in V_w\}$ is a partition of $[1:n]$ into intervals. Furthermore, $w[i] \neq w[j]$ for all $i \in L_j$ and $j \neq 0$: if $w[i]$ would be the same as $w[j]$ then $w[j+1:i]$ would also be a universal word, so i would not be in L_j. Also, if $i = \max(L_j)$ for some $j > 0$ then $w[i+1] = w[j]$. Indeed, there exists $j' \in [j+1:i]$ such that $w[j':i+1]$ is universal. But $w[j]$ does not occur in $w[j':i]$, so $w[j] = w[i+1]$ must hold.

Further, we define for all positions i of w the value $\mathrm{freq}[i] = |w[1:i]|_{w[i]}$, the number of occurrences of $w[i]$ in $w[1:i]$. Also, let $T[i] = \min\{|w[i+1:n]|_a \mid a \in \Sigma\}$, for $i \in [0, n-1]$, be the least number of occurrences of a letter in $w[i+1:n]$; set $T[n] = 0$.

▶ **Lemma 3.4.** *Let w be a word, with $|w| = n$, $alph(w) = \Sigma$, and $\Sigma = \{1, 2, \ldots, \sigma\}$. We can compute in $O(n)$ time the following data structures:* 1. *the array* $\mathrm{univ}[\cdot]$*;* 2. *the set V_w and the lists L_j, for all $j \in V_w \setminus \{0\}$;* 3. *the array* $\mathrm{freq}[\cdot]$*;* 4. *the array* $T[\cdot]$*;* 5. *the values* $\mathrm{last}_{j-1}[w[i]]$*, for all $j \in V_w$ and all $i \in L_j$;* 6. *the values* $\mathrm{last}_{i-1}[w[i]]$*, for all $i \in [2:n]$.*

Consider again $w = \texttt{bananaban}$. In Lemma 3.4 we compute the following values. Firstly, $\mathrm{univ}[1] = \mathrm{univ}[2] = 0$, $\mathrm{univ}[\ell] = 1$ for $\ell \in [3:6]$, $\mathrm{univ}[7] = \mathrm{univ}[8] = 5$, $\mathrm{univ}[9] = 7$. Thus, $V_w = \{0, 1, 5, 7\}$ and $L_0 = [1:2]$, $L_1 = [3:6]$, $L_5 = [7:8]$, $L_7 = [9:9]$. Secondly, $\mathrm{freq}[1] = 1$, $\mathrm{freq}[2] = \mathrm{freq}[3] = 1$, $\mathrm{freq}[4] = \mathrm{freq}[5] = 2$, $\mathrm{freq}[6] = 3$, $\mathrm{freq}[7] = 2$, $\mathrm{freq}[8] = 4$, $\mathrm{freq}[9] = 3$. Moreover, $T[0] = 2$, $T[\ell] = 1$ for $\ell \in [1:6]$, and $T[\ell] = 0$ for $\ell \in [7:9]$. Then, for $j = 1$, we have $\mathrm{last}_0[a] = 0$, for $a \in \{\texttt{a}, \texttt{b}, \texttt{n}\}$; for $j = 5$, we have $\mathrm{last}_4[\texttt{b}] = 1$ and $\mathrm{last}_4[\texttt{a}] = 4$; for $j = 7$, we have $\mathrm{last}_6[\texttt{n}] = 5$. Finally, $\mathrm{last}_0[\texttt{b}] = 10$, $\mathrm{last}_1[\texttt{a}] = 10$, $\mathrm{last}_2[\texttt{n}] = 10$, $\mathrm{last}_3[\texttt{a}] = 2$, $\mathrm{last}_4[\texttt{n}] = 3$, $\mathrm{last}_5[\texttt{a}] = 4$, $\mathrm{last}_6[\texttt{b}] = 1$, $\mathrm{last}_7[\texttt{a}] = 6$, $\mathrm{last}_8[\texttt{n}] = 5$.

The main idea behind proving Lemmas 3.1, 3.3, and 3.4 is to traverse the word w left to right (or, respectively, right to left) and maintain the number of occurrences, as well as the last occurrence, of each letter in the prefix (respectively, suffix) of w that we have visited so far. For Lemma 3.2, we only consider a sliding window of size σ which traverses the word left-to-right, while maintaining similar data as before, but only for the content of the window. In all cases, this requires linear time and enables us to construct the desired data structures.

Together with the string-processing data structures we defined above, we need the following general technical data structures lemma. This lemma (combined with some combinatorial observations) will be used to speed up some of our dynamic programming algorithms.

In this lemma we process a list A which initially has σ elements, and in which we insert, in successive steps, σ new elements, by appending them always at the same end. For simplicity, we can assume that the list A is a sequence with 2σ elements (denoted $A[i]$, with $i \in [1:2\sigma]$), out of which the last σ are initially undefined. The i^{th} insertion would, consequently, mean setting $A[\sigma+i]$ to the actual value that we want to insert in the list A. In our lemma we will also repeatedly perform an operation which decrements the values of some elements of the list A. However, we will not require to be able to explicitly access, after every operation, all the elements of the list (so we will not need to retrieve the values $A[i]$). Consequently, we will not maintain explicitly the value of all the elements of A (that is, we will not update the elements affected by decrements). We are only interested in being able to retrieve (by value and position), at each moment, the smallest element and the last element of A. Thus, throughout the computation, we only maintain a subset of important elements of A, including the aforementioned two. We can now state our result, whose proof is based on Lemma 2.7.

▶ **Lemma 3.5.** *Let A be a list with σ elements (natural numbers) and let $m = \sigma$. We can execute (in order) the sequence of σ operations $o_1, \ldots, o_\sigma$ on A in overall $O(\sigma)$ time, where o_i consists of the following three steps, for $i \in [1:m]$:*

1. *Return $e = \arg\min\{A[i] \mid i \in [1:m]\}$ and $A[e]$.*
2. *For some $j_i \in [1:m]$, decrement all elements $A[j_i], A[j_i+1], \ldots, A[m]$ by 1.*
3. *For some natural number x_i, append the element x_i to A (i.e., set $A[m+1]$ to x_i), and increment m by 1 (i.e., set m to $m+1$).*

Proof. Firstly, we will *run a preprocessing* of A.

We begin by defining recursively a finite sequence of positions as follows:

- a_1 is the rightmost position of A on which $\min\{A[i] \mid i \in [1:\sigma]\}$ occurs;
- for $i \geq 2$, if $a_{i-1} < \sigma$, then a_i is the rightmost position on which $\min\{A[i] \mid i \in [a_{i-1}+1 : \sigma]\}$ occurs;
- for $i \geq 2$, if $a_{i-1} = \sigma$, then we can stop, our sequence will have $i-1$ elements.

Let p be the number of elements in the sequence defined above, i.e., our sequence is $a_1, \ldots, a_p$. For convenience, let $a_0 = 0$. Then the sequence $a_1, \ldots, a_p$ fulfils the following properties:

- $a_p = \sigma$ and $a_i > a_{i-1}$, for all $i \in [1:p]$;
- $A[a_i] > A[a_{i-1}]$ for all $i \in [2:p]$;
- for all $i \in [1:p]$, we have $A[a_i] < A[t]$, for all $t \in [a_i+1:\sigma]$;
- for all $i \in [1:p]$, we have $A[a_i] \leq A[t]$, for all $t \in [a_{i-1}+1:a_i]$.

By definition, for $i \in [1:p]$ we have $A[a_i] = \min\{A[t] \mid t \in [a_{i-1}+1:\sigma]\}$, $A[a_i] < \min\{A[t] \mid t \in [a_i+1:\sigma]\}$, and $a_1 = \min\{A[i] \mid i \in [1:\sigma]\}$. Clearly, we have $a_p = \sigma$.

The positions $a_1, \ldots, a_p$ can be computed in linear time $O(\sigma)$, in reversed order. As we do not know from the beginning the value of p, we will compute a sequence $b_1, b_2, \ldots$ of positions as follows. We start with $b_1 = \sigma$, $t = \sigma - 1$, and $i = 2$. Then, while $t \geq 1$ we do the following case analysis. If $A[t] < b_{i-1}$, then set $b_i = t$, increment i by 1, and decrement t by 1. Otherwise, if $A[t] \geq b_{i-1}$, just decrement t by 1. It is straightforward that this process takes $O(\sigma)$ time, and, when we have finished it, the number i is exactly the number p, and $a_i = b_{p-i+1}$.

Another observation is that, for $a_0 = 0$, the intervals $[a_{i-1}+1, a_i]$, for $i \in [1,p]$, define a partition of the interval $[1:\sigma]$ into p intervals. Therefore, we can define a partition of the interval $[1:2\sigma]$ into the intervals $[a_{i-1}+1:a_i]$, for $i \in [1,p]$, and $[t:t]$, for $t \in [\sigma+1:2\sigma]$. Thus, we construct in linear time, according to Lemma 2.6, an interval union-find data structure for the interval $[1:2\sigma]$, as induced by the intervals $[1:a_1]$, $[a_1+1:a_2], \ldots [a_{p-1}, a_p]$, $[\sigma+1:\sigma+1]$, $[\sigma+2:\sigma+2], \ldots [2\sigma:2\sigma]$.

Let us now take $m = \sigma$ (and assume the convention $A[0] = 0$). We associate as satellite data to each interval $[x:y]$ with $y \leq m$ from our interval union-find data structure the value $A[y] - A[x-1]$.

This entire preprocessing takes clearly $O(\sigma)$ time.

In order to explain how the operations are implemented, we assume as invariant that the following properties are fulfilled before o_i is executed, for $i \in [1:\sigma]$:

- A contains m elements;
- all intervals $[x:y]$ with $y > m$ from our interval union-find data structure are singletons (i.e., $x = y$);
- for each interval $[x:y]$ with $y \leq m$, we have the associated satellite data $A[y] - A[x-1]$;
- for each interval $[x:y]$ with $y \leq m$, we have that $A[y] \leq A[t]$ for $t \in [x:m]$ and $A[y] < A[t]$ for $t \in [y+1:m]$;
- we have stored in a variable ℓ the value $A[m]$.

This clearly holds after the preprocessing step, so before executing o_1.

Let us now explain *how the operation o_i is executed.*

The first step of o_i is to return $e = \min\{A[i] \mid i \in [1 : m]\}$ and i_e the rightmost position of the list A such that $A[i_e] = e$. We execute `find`(1) to return the first interval $[1 : i_e]$ stored in our interval union-find data structure; $A[i_e]$ is the satellite data associated to this interval (by convention, $A[i_e] - A[1-1] = A[i_e] - A[0] = A[i_e]$). The fact that the invariant property holds shows that i_e is correctly computed.

The second step of o_i is to decrement all elements $A[j_i], A[j_i+1], \ldots, A[m]$ by 1, for some $j_i \in [1 : m]$. We will make no actual change to the elements of the list A, as this would be too inefficient, but we might have to change the state of the union-find data structure, as well as the satellite data associated to some intervals of this structure.

So, let $[x : y]$ be the interval containing j_i, returned by `find`(j_i), and also assume first that $x \neq 1$.

According to the invariant, $A[j_i] \geq A[y]$ and $A[y] > A[x-1]$. After decrementing the elements $A[j_i], A[j_i+1], \ldots, A[m]$ by 1, the difference $A[t] - A[t']$ is exactly the same as before, for all $t, t' \in [j_i : m]$. In consequence, the relative order between the elements of the suffix $A[j_i : m]$ of the list A is preserved. Also, for all $t \in [x : j_i - 1]$, we have now $A[t] > A[y]$ (before decrementing $A[y]$ we had only $A[t] \geq A[y]$). However, the difference $A[y] - A[x-1]$ is now decreased by 1. If it stays strictly positive, we just update the satellite data of the respective interval (by decrementing it accordingly by 1). If $A[y] - A[x-1] = 0$, then we make the `union` of the interval $[z : x-1]$ (returned by `find`$(x-1)$) and $[x : y]$ to obtain the new interval $[z : y]$. Its satellite data is $A[y] - A[z-1] = A[x-1] - A[z-1]$, so the same as the satellite data that was before associated to $[z : x-1]$. The invariant is clearly preserved, as, even after decrementing it, $A[y]$ (which is now equal to $A[x-1]$) is strictly greater than $A[z-1]$, strictly smaller than $A[t]$, for $t \in [y+1 : m]$, and smaller than or equal to $A[t]$, for $t \in [z : y]$.

If the interval containing j_i is $[1 : y]$, then we just update the satellite data of the respective interval by decrementing it by 1.

The third step of o_i is to append the element x_i to A (i.e., set $A[m+1] = x_i$), for some natural number x_i, and increment m by 1.

We implement this as follows. Let $t = m$ and $q = A[m]$ (this value is stored and maintained using the variable ℓ). While $t \geq 1$ do the following. Let $[z, t]$ be the interval returned by `find`(t); we have $q = A[t]$. If $q \geq x_i$, make the union of $[z : t]$ and $[t+1 : m+1]$; update $q = q - (A[t] - A[z-1]) = A[z-1]$ (using the satellite data $A[t] - A[z-1]$ associated to $[z, t]$), update $t = z - 1$, and reiterate the loop. If $q < x_i$, exit the loop. After this, we set m to $m+1$ and $\ell = x_i$.

It is not hard to see that after running this third step, so before executing operation o_{i+1}, the invariant is preserved.

Performing operation o_i takes an amount of time proportional to the sum of the number of `union` and the number of `find` operations executed during its three steps. By Lemma 2.7, this means that executing all operations $o_1, \ldots, o_\sigma$ takes in total at most $O(\sigma)$ time. ◀

4 Edit Distance

We are interested in computing the minimal number of edits we need to apply to a word w, with $|w| = n$, $\mathrm{alph}(w) = \Sigma$, with universality index $\iota(w)$, so that it is transformed into a word with universality index k, w.r.t. the same alphabet Σ. The edits considered are insertion, deletion, substitution, and the number we want to compute can be seen as the *edit distance* between w and the set of k-universal words over Σ.

However, if we want to obtain a k-universal word with $k > \iota(w)$, then it is enough to consider only insertions. Indeed, deleting a letter of a word can only restrict the set of subsequences of the respective word, while in this case we are interested in enriching it. Substituting a letter might make sense, but it can be simulated by an insertion: assume one wants to substitute the letter $\mathtt{a}$ on position i of a word w by a $\mathtt{b}$. It is enough to insert a $\mathtt{b}$ next to position i, and the set of subsequences of w is enriched with all the words that could have appeared as subsequences of the word where $\mathtt{a}$ was actually replaced by $\mathtt{b}$. We might have some extra words in the set of subsequences, which would have been eliminated through the substitution, but it does not affect our goal of reaching k-universality.

If we want to obtain a word with universality index k, for $k < \iota(w)$, then it is enough to consider only deletions. Assume that we have a sequence of edits that transforms the word w into a word w' with universality index k. Now, remove all the insertions of letters from that sequence. The word w'' we obtain by executing this new sequence of operations clearly fulfils $\iota(w'') \leq \iota(w')$. Further, in the new sequence, replace all substitutions with deletions. We obtain a word w''' with a set of subsequences strictly included in the one of w'', so with $\iota(w''') \leq \iota(w'')$. As each deletion changes the universality index by at most 1, it is clear that (a prefix of) this new sequence of deletions witnesses a shorter sequence of edits which transforms w into a word of universality index k.

So, to increase the universality index of a word it is enough to use insertions and to decrease the universality index of a word it is enough to use deletions. Nevertheless, one might be interested in what happens if we only use substitutions. In this way, we can both decrease and increase the universality index of a word. Moreover, one can see the minimal number of substitutions needed to transform w into a k-universal word as the Hamming distance between w and the set of k-universal words. We will discuss each of these cases separately.

4.1 Insertions

▶ **Theorem 4.1.** *Let w be a word, with $|w| = n$, $alph(w) = \Sigma$, and $\Sigma = \{1, 2, \ldots, \sigma\}$. Let $k \geq \iota(w)$ be an integer. We can compute the minimal number of insertions needed to apply to w in order to obtain a k-universal word (w.r.t. Σ) in $O(nk)$ time if $k \leq n$ and $O(T(n, \sigma, k))$ time otherwise, where $T(n, \sigma, k)$ is the time needed to compute $k\sigma - n$.*

Proof.

Case 1. Let us assume first that $k \leq n$. We structured our proof in such a way that the idea of the solution, as well as the actual computation steps, and the arguments supporting their correctness are clearly marked. Pseudocode for this algorithm is given in the full version of the paper [19].

§ General approach. We want to transform the word w into a k-universal word with a minimal number of insertions. Assume that the word we obtain this way is w', and $|w'| = m$. Thus, w' has a prefix $w'[1 : m']$ which is k-universal, but $w'[1 : m' - 1]$ is not k-universal. Moreover, $w'[1 : m']$ is obtained from a prefix $w[1 : \ell]$ of w, and $w'[m' + 1 : m] = w[\ell + 1 : n]$. Indeed, any insertion done to obtain $w'[m' + 1 : m]$ can be simply omitted and still obtain a k-universal word from w, with a lower number of insertions.

Consequently, it is natural to compute the minimal number of insertions needed to transform $w[1 : \ell]$ into a t-universal word, for all $\ell \leq n$ and $t \leq k$. Let $M[\ell][t]$ denote this number. By the same reasoning as above, transforming (with insertions) $w[1 : \ell]$ into a t-universal word means that there exists a prefix $w[1 : \ell']$ of $w[1 : \ell]$ which is transformed into a $(t - 1)$-universal word and $w[\ell' + 1 : \ell]$ is transformed into a 1-universal word. Clearly, the number of insertions needed to transform $w[\ell' + 1 : \ell]$ into a 1-universal word is $\sigma - \Delta(\ell' + 1, \ell)$, i.e., the number of distinct letters not occurring in $w[\ell'+1:\ell]$. As we are interested in the minimal number of insertions needed to transform $w[1:\ell]$ into a t-universal word, we need to find a position ℓ' such that the total number of insertions needed to transform $w[1:\ell']$ into a $(t-1)$-universal word and $w[\ell'+1:\ell]$ into a 1-universal word is minimal.

§ Algorithm – initial idea. So, for $\ell \in [1:n]$ and $t \in [1:k]$, $M[\ell][t]$ is the minimal number of insertions needed to make $w[1:\ell]$ t-universal. By the explanations above, we get the following recurrence $M[\ell][t] = \min\{M[\ell'][t-1] + (\sigma - \Delta(\ell'+1,\ell)) \mid \ell' \leq \ell\}$. Clearly, $M[\ell][1] = \sigma - \Delta(1,\ell)$. Also, it is immediate to note that $M[\ell][t] \geq M[\ell''][t]$ for all $\ell \leq \ell''$. Indeed, transforming a word into a t-universal word can always be done with at most as many insertions as those used in transforming any of its prefixes into a t-universal word.

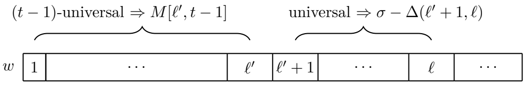

$(t-1)$-universal ⇒ $M[\ell', t-1]$
universal ⇒ $\sigma - \Delta(\ell'+1, \ell)$
w
1
···
ℓ'
$\ell'+1$
···
ℓ
···

Figure 1 Illustration of the formula developed for the computation of $M[\ell][t]$.

We now want to compute the elements of matrix M. Before this, we produce the data structures of Lemma 3.3 (and we use the notations from its framework). That is, we compute in $O(n)$ time $\text{last}_{j\sigma+1}[a]$ and $d_{j\sigma+1}[a] = \Delta(\text{last}_{j\sigma+1}[a], j\sigma+1)$, for all $a \in \Sigma$ and all $j \leq \frac{(n-1)}{\sigma}$.

By Lemma 3.1, we can compute the values $M[\ell][1]$, for all $\ell \in [1:n]$ in $O(n)$ time. However, a direct computation of the values $M[\ell][t]$, for $t > 1$, according to the recurrence above is not efficient. Implemented directly, it requires $O(n^2k)$ time; using an efficient structure (e.g., interval trees) for computing the various minima leads to an $O(nk\log n)$-time solution; exploiting the algebraic properties of M (related to the Monge property [1]) leads to an $O(nk\log\sigma/\log\log\sigma)$-time solution. We will describe a more efficient solution.

§ A useful observation. Assume that to transform $w[1:\ell]$ into a t-universal word we transform $w[1:\ell']$ into a $(t-1)$-universal word and $w[\ell'+1:\ell]$ into a 1-universal word. The number of insertions needed to do this is $M[\ell'][t-1] + (\sigma - \Delta(\ell'+1,\ell))$. If $w[\ell'+1]$ occurs twice in $w[\ell'+1:\ell]$, then $M[\ell'][t-1] + (\sigma - \Delta(\ell'+1,\ell)) \geq M[\ell'+1][t-1] + (\sigma - \Delta(\ell'+2,\ell))$. Thus, we can rewrite our recurrence in the following way, using the framework of Lemma 3.3: $M[\ell][t] = \min\{M[\ell'][t-1] + (\sigma - \Delta(\ell'+1,\ell)) \mid \ell'+1 \in S_\ell \cup \{\ell+1\}\}$ (recall the definition of $S_\ell = \{\text{last}_\ell[a] \mid a \in \text{alph}(w[1:\ell])\}$ from Section 3).

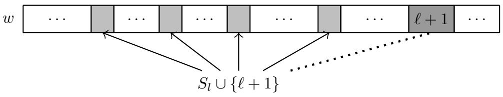


Figure 2 Only the positions $\ell'+1 \in \{\text{last}_\ell[a] \mid a \in \text{alph}(w[1:\ell])\} \cup \{\ell+1\} = S_\ell \cup \{\ell+1\}$ are needed to compute $M[\ell][t]$ by dynamic programming. These positions are depicted here in grey.

Once more, a brief analysis can be done. Using directly this observation leads to an $O(nk\sigma)$-time algorithm for our problem; an implementation based on, e.g., interval trees runs in $O(nk\log\sigma)$-time. In the following we see that a faster solution exists.

In fact, in the efficient version of our algorithm we will use a slightly weaker formula, where the minimum is computed for all elements $\ell'+1$ from a set $S'_\ell \cup \{\ell+1\}$, instead of the set $S_\ell \cup \{\ell+1\}$, where S'_ℓ is a superset of size at most 2σ of S_ℓ defined as follows. If $\ell = j\sigma + i$, for some $j \leq (n-1)/\sigma$ and $i \in [1:\sigma]$, then $S'_\ell = \begin{cases} S_\ell & \text{if } i = 1, \\ S_{j\sigma+1} \cup \{j\sigma+2, \ldots, j\sigma+i\} & \text{if } i \in [2:\sigma]. \end{cases}$

§ Algorithm – the efficient variant. Using the observation above, together with Lemma 3.5, we can compute the elements of the matrix M efficiently using dynamic programming.

So, let us consider a value $t \geq 2$. Assume that we have computed the values $M[\ell][t-1]$, for all $\ell \in [1:n]$. We now want to compute the values $M[\ell][t]$, for all $\ell \in [1:n]$. The main idea in doing this efficiently is to split the computation of the elements on column $M[\cdot][t]$ of the matrix M in phases. In phase j we compute the values $M[j\sigma+1][t]$, $M[j\sigma+2][t], \ldots$, $M[(j+1)\sigma][t]$, for $j \leq (n-1)/\sigma$.

We now consider some j, with $0 \leq j \leq (n-1)/\sigma$. We want to apply Lemma 3.5, so we need to define the list A of size σ. This is done as follows.

We will maintain an auxiliary array $\text{pos}[\cdot]$ with σ elements. Moreover, the element $A[i]$, for each i, is accompanied by two satellite information: a position of w and the letter found on that position. For a from 1 to σ, if $d_{j\sigma+1}[a] = \sigma - i$ for some $i < \sigma$ then we set $A[i+1] = M[\text{last}_{j\sigma+1}[a]-1][t-1]+i$ and $\text{pos}[a] = i+1$; the satellite data of $A[i+1]$ is the pair $(\text{last}_{j\sigma+1}[a], a)$. If, for some letter a, $\text{last}_{j\sigma+1}[a] = n+1$ and $d_{j\sigma+1}[a] = 0$ (i.e., a does not occur in $w[1:j\sigma+1]$) we set $\text{pos}[a] = 0$.

Intuitively, one can see the elements of A as triples: $(A[e], \text{last}_{j\sigma+1}[a], a)$ where $A[e] = M[\text{last}_{j\sigma+1}[a]-1][t-1]+e-1$, with $e \in [1:\sigma]$ and $a \in \Sigma$. More precisely, let $a_d, a_{d-1}, \ldots, a_1$ be the letters of Σ that occur in $w[1:j\sigma+1]$, ordered such that $\text{last}_{j\sigma+1}[a_e] < \text{last}_{j\sigma+1}[a_f]$ if and only if $e > f$. At this point, we have defined only the last d elements of A and, for $i \in [1:d]$, the element on position $\sigma-i+1$ is $A[\sigma-i+1] = M[\text{last}_{j\sigma+1}[a_i]-1][t-1]+(\sigma-i)$ and has the satellite data $(\text{last}_{j\sigma+1}[a_i], a_i)$. Also, $\text{pos}[a_i] = \sigma-i+1$. The first $\sigma-d$ elements of A are set to ∞; as convention, applying arithmetic operations to ∞ leaves it unchanged.

We set m to σ and define (and apply) a sequence of operations $o_1, \ldots, o_\sigma$ as in Lemma 3.5.

An invariant: We want to ensure that the list A fulfils the following invariant properties before the execution of each operation o_i.

- For $e \in [1:d]$, the triple on position $\sigma-e+1$ of A is: $(M[\text{last}_{j\sigma+1}[a_e]-1][t-1]+(\sigma-\Delta(\text{last}_{j\sigma+1}[a_e], j\sigma+i)), \text{last}_{j\sigma+1}[a_e], a_e)$. That is, $A[\sigma-e+1] = M[\text{last}_{j\sigma+1}[a_e]-1][t-1]+(\sigma-\Delta(\text{last}_{j\sigma+1}[a_e], j\sigma+i))$.
- For $g \in [1:i-1]$, the triple on position $\sigma+g$ of A is: $(M[j\sigma+g][t-1]+(\sigma-\Delta(j\sigma+g+1, j\sigma+i)), j\sigma+g+1, w[j\sigma+g+1])$. That is $A[\sigma+g] = M[j\sigma+g][t-1]+(\sigma-\Delta(j\sigma+g+1, j\sigma+i))$.
- $\text{pos}[a]$ is the position of the rightmost position i storing a triple $(A[i], \ell, a)$.

That is, the list A contains all the values $M[\ell][t-1]+(\sigma-\Delta(\ell+1, j\sigma+i))$, for $\ell+1 \in S_{j\sigma+1} \cup \{j\sigma+2, \ldots, j\sigma+i\}$, and $\text{pos}[a]$ indicates the rightmost position of the list A where we store a value $M[\ell][t-1]+(\sigma-\Delta(\ell+1, j\sigma+i))$ with $w[\ell+1] = a$. A consequence of this is that $A[\text{pos}[a]] = M[\text{last}_{j\sigma+i}[a]-1][t-1]+(\sigma-\Delta(\text{last}_{j\sigma+i}[a], j\sigma+i))$.

The invariant clearly holds for $i = 1$.

§ Algorithm – application of Lemma 3.5. In o_i, we extract the minimum q of A. Then set $M[j\sigma+i][t] = \min\{q, M[j\sigma+i][t-1]+\sigma\}$. We decrement by 1 all elements of A on the positions $\text{pos}[a]+1, \text{pos}[a]+2, \ldots, m$, where $a = w[j\sigma+i+1]$. Then, we append to A the element $M[j\sigma+i][t-1]+(\sigma-1)$, with the satellite data $(j\sigma+i+1, a)$, which implicitly increments m by 1, and set $\text{pos}[a] = m$.

▷ Claim 4.2. The invariant holds after operation o_i.

Proof of Claim 4.2. We now need to show that the invariant is preserved after this step. If $a = w[j\sigma+i+1]$ then the number of distinct letters occurring after each position $g > \text{last}_{j\sigma+i}[a]$ in $w[1:j\sigma+i]$ is exactly one smaller than the number of distinct letters occurring after g in $w[1:j\sigma+i+1]$. This means that $M[g-1][t-1]+(\sigma-\Delta(g, j\sigma+i+1))$ is one smaller than

$M[g-1][t-1]+(\sigma-\Delta(g,j\sigma+i))$. Consequently, all values occurring on positions greater than $\mathrm{pos}[a]$ in the list A, which stored some values $M[g-1][t-1]+(\sigma-\Delta(g,j\sigma+i+1))$ with $g>\mathrm{last}_{j\sigma+i}[a]$, should be decremented by 1. Also, the number of distinct letters occurring after each position $g\leq \mathrm{last}_{j\sigma+i}[a]$ in $w[1:j\sigma+i]$ is exactly the same as number of distinct letters occurring after g in $w[1:j\sigma+i+1]$. Thus, all values occurring on positions smaller or equal to $\mathrm{pos}[a]$ in the list A, which stored some values $M[g-1][t-1]+(\sigma-\Delta(g,j\sigma+i+1))$ with $g\leq \mathrm{last}_{j\sigma+i}[a]$, should stay the same. So, the invariant holds for the first $\sigma+i-1$ positions of A. After appending $M[j\sigma+i][t-1]+(\sigma-1)$ to A and incrementing m, then the invariant holds for the position $\sigma+i$ (which is also the last position) of A too, so the invariant still holds for all positions of A.

Furthermore, the only position of the pos array that needs to be updated after operation o_i is $\mathrm{pos}[a]$, and it needs to be set to the new value of m. This is exactly what we do. ◁

▷ Claim 4.3. $M[j\sigma+i][t]$ is correctly computed, for all $i\in[1:\sigma]$.

Proof of Claim 4.3. According to the invariant, before executing operation o_i, A contains the values $M[\ell][t-1]+(\sigma-\Delta(\ell+1,j\sigma+i))$, for $\ell+1\in S_{j\sigma+1}$, and $M[j\sigma+g][t-1]+(\sigma-\Delta(j\sigma+g+1,j\sigma+i))$, for $g\in[1:i-1]$. As $S'_{j\sigma+i}=S_{j\sigma+1}\cup\{j\sigma+g+1\mid g\in[1:i-1]\}$ is a superset of size at most 2σ of $S_{j\sigma+i}$, we obtain that $M[j\sigma+i][t]$ is correctly computed as the minimum between the smallest value in A and $M[j\sigma+i][t-1]+\sigma$. ◁

§ Algorithm – the result of applying Lemma 3.5. After executing the σ operations $o_1,\ldots,o_\sigma$, we have computed the values $M[j\sigma+1][t]$, $M[j\sigma+2][t],\ldots$, $M[(j+1)\sigma][t]$ correctly. We can move on to phase $j+1$ and repeat this process.

§ The result and complexity. The minimal number of insertions needed to make w k-universal is, according to the observations we made, correctly computed as $M[n][k]$.

By Lemma 3.5, computing $M[j\sigma+1][t]$, $M[j\sigma+2][t],\ldots$, $M[(j+1)\sigma][t]$ takes $O(\sigma)$ for each j. Overall, computing the entire column $M[\cdot][t]$ takes $O(n)$ time. We do this for all $t\leq k$, so we use $O(nk)$ time in total to compute all elements of M. This concludes Case 1.

Case 2. If $k>n$, we return $k\sigma-n$. We need, in all cases, $k\sigma-n$ insertions to obtain a word of length $k\sigma$ from w. This is also sufficient: we first use $n(\sigma-1)$ insertions to transform w into $(1\cdots\sigma)^n$; then, by $(k-n)\sigma$ insertions, we further transform it into $(1\cdots\sigma)^k$. So the time needed to solve our problem, in this case, is the time needed to compute $k\sigma-n$. ◀

Note that, if k is in $O(c^n)$ for constant c, then $T(n,\sigma,k)\in O(n\log\sigma)$. Hence, in that case, our algorithm runs in $O(n\log\sigma)$ time. If $k\in O(1)$ our algorithm runs in optimal $O(n)$ time.

4.2 Deletions

▶ **Theorem 4.4.** *Let w be a word, with $|w|=n$, $alph(w)=\Sigma$, and $\Sigma=\{1,2,\ldots,\sigma\}$. Let k be an integer with $k\leq\iota(w)\leq n/\sigma$. We can compute in $O(nk)$ time the minimal number of deletions needed to obtain a word of universality index k (w.r.t. Σ) from w.*

The idea of this proof is the following. Assume that w' is a word of universality index k obtained via the sequence of deletions of minimal length from w. Clearly, w' is a subsequence of w, and, by the decomposition defined in Theorem 2.5, there exist $w'_1,\ldots,w'_k$, all of universality index exactly 1, and w'_{k+1}, of universality index 0, such that $w'=w'_1\cdots w'_k w'_{k+1}$. It follows that each of the words w'_i is a subsequence of w too. So we will try to identify each subsequence $w'_1\cdots w'_p$ for $p\leq k$ and the shortest factor $w[1:i]$ from which it is obtained. To this end, we define the matrix N, where $N[i][p]$ is the minimal number of deletions we need to apply to $w[1:i]$, without deleting $w[i]$, to obtain a word v from it, with $\iota(v)=p$

and $\iota(v[1:|v|-1]) = p-1$ (for $i \in [1:n]$ and $p \in [1:k]$). If $\iota(w[1:i]) \geq 1$, then $N[i][1] = |w[1:i]|_{w[i]} - 1$, as we have to delete all occurrences of $w[i]$ from $w[1:i]$, except the one on position i. Then, $N[i][p] = \min\{N[j][p-1] + |w[j+1:i]|_{w[i]} - 1 \mid j < i$ such that $\iota(w[j+1:i]) \geq 1\}$. This gives a dynamic programming algorithm for computing N. Using additional data structures extending the standard Range Minimum Queries structures (see Lemma 2.9), we can compute the elements of N in $O(nk)$ time. To show the statement, we return $\min\{N[i,k] + T[i] \mid 1 \leq i \leq n\}$, using the array T of Lemma 3.4.

4.3 Substitutions

▶ **Theorem 4.5.** *Let w be a word, with $|w| = n$, alph$(w) = \Sigma$, and $\Sigma = \{1, 2, \ldots, \sigma\}$. Let k be an integer $0 \leq k \leq \lfloor \frac{n}{\sigma} \rfloor$. We can compute the minimal number of substitutions needed to apply to w in order to obtain a k-universal word (w.r.t. Σ) in $O(nk)$ time.*

The case $k > \iota(w)$ is treated similarly to the case of changing the universality of a word by insertions, described in Theorem 4.1. We define a matrix M, with $M[\ell][t]$ being the minimal number of substitutions one needs to apply to $w[1:\ell]$ in order to make it t-universal, for all $\ell \in [1:n]$ and all $t \in [1:k]$. Then, we derive the following recurrence: $M[\ell][t] = \min\{M[\ell'][t-1] + (\sigma - \Delta(\ell'+1, \ell)) \mid \ell'+1 \in \mathfrak{S}_\ell\}$, where $\mathfrak{S}_\ell = (S_\ell \cap [(t-1)\sigma : \ell-\sigma]) \cup \{\ell-\sigma+1\}$ (S_ℓ is defined in Section 3). The fact that at most σ elements of $M[\cdot][t-1]$ are used to compute each of the elements $M[\ell][t]$ allows us to apply Lemma 3.5 in almost the same way as we did in the algorithm of Theorem 4.1, and compute all the elements of M in $O(nk)$ time. The number we want to compute is $M[n][k]$.

For the case $k < \iota(w)$, we show that when decreasing the universality index of a word, it makes no difference whether we use substitutions or deletions. So, it is enough to use the algorithm of Theorem 4.4 and return the computed result as the answer to our current problem.

Note that, while substitutions and deletions can be used similarly to decrease the universality index of a word, we always need at least as many substitutions as insertions to increase it. To see that this inequality can also be strict, note that one insertion is enough to make `aabb` 2-universal, but we need two substitutions to achieve the same result.

5 Extensions

In this paper, we presented a series of algorithms computing the minimal number of edits one needs to apply to a word w in order to reach k-subsequence universality. In fact (see the proofs in the full version of this paper [19]), one can extend our algorithms and, using additional $O(k|\text{alph}(w)|)$ time, we can effectively construct a k-universal word which is closest to w, with respect to the edit distance. All our algorithms can be implemented in linear space (see [19]) using a technique called *Hirschberg's trick* [35].

The algorithms we presented work in a general setting: the processed words are over an integer alphabet. It seems natural to ask whether faster solutions for inputs over an alphabet of constant size (e.g., binary alphabets) exist. To this end, we state the following result.

▶ **Theorem 5.1.** *Let w be a word, with $|w| = n$, alph$(w) = \Sigma$, and $\Sigma = \{1, 2, \ldots, \sigma\}$. Let k be an integer $\iota(w) < k$. We can compute the minimal number of insertions (resp., substitutions) needed to apply to w in order to obtain a k-universal word (w.r.t. Σ) in $(n \log^{O(1)} \sigma)$-time.*

We describe here the idea used in the case of insertions (as it works with small modifications for substitutions, too). Full details are given in [19]. We define a weighted directed acyclic graph G with the nodes $0, 1, \ldots, n$ and directed edges (i, j) with $i < j$. Let $\omega(i, j) = \sigma - \Delta(i+1, j)$ (i.e., the number of letter of Σ which do not appear in $w[i+1:j]$) be the weight of the edge (i, j). It is not hard to show that the number of edits needed to transform w into a k-universal word equals the weight of a minimum weight k-link path in G (see [7, 2, 58]). The graph G will not be explicitly constructed, but we can construct in $(n \log^{O(1)} \sigma)$-time an oracle data structure allowing us to retrieve in $O(\log \sigma / \log \log \sigma)$ the weight of any edge (i, j) of the graph. Further, as G fulfills the concave Monge property (i.e., $\omega(i, j) + \omega(i + 1, j + 1) < \omega(i + 1, j) + \omega(i, j + 1)$ holds for all $0 < i + 1 < j < n$), then the minimum weight k-link path in G can be computed in $(n \log^{O(1)} \sigma)$-time, using the algorithms of [7, 2]. If $\sigma \in O(1)$, then the algorithms of Theorem 5.1 run in optimal linear time $O(n)$.

It is open whether a similar result holds for the case of decreasing the universality index of a word. However, the main open questions remaining from this work go back to our initial motivation. Namely, we would be very interested in settling the complexity of the following problem: Given a word w and a k-piecewise testable language L, succinctly specified, compute efficiently the edit distance from w to L. In particular, the case when L is defined as the language of words $\sim_k$-equivalent to a target-word u seems very interesting to us.

References

1 Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987.

2 Alok Aggarwal, Baruch Schieber, and Takeshi Tokuyama. Finding a minimum-weight k-link path graphs with the concae monge property and applications. *Discret. Comput. Geom.*, 12:263–280, 1994.

3 Cyril Allauzen and Mehryar Mohri. Linear-space computation of the edit-distance between a string and a finite automaton. *CoRR*, abs/0904.4686, 2009. arXiv:0904.4686.

4 Lorraine A. K. Ayad, Carl Barton, and Solon P. Pissis. A faster and more accurate heuristic for cyclic edit distance computation. *Pattern Recognit. Lett.*, 88:81–87, 2017.

5 Ricardo A. Baeza-Yates. Searching subsequences. *Theor. Comput. Sci.*, 78(2):363–376, 1991.

6 Laura Barker, Pamela Fleischmann, Katharina Harwardt, Florin Manea, and Dirk Nowotka. Scattered factor-universality of words. In Natasa Jonoska and Dmytro Savchuk, editors, *Proc. DLT 2020*, volume 12086 of *Lecture Notes in Computer Science*, pages 14–28, 2020.

7 Wolfgang W. Bein, Lawrence L. Larmore, and James K. Park. The d-edge shortest-path problem for a monge graph. *Technical Report*, SAND–92-1724C:1–10, 1992.

8 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *Proc. LATIN 2000*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94, 2000.

9 Giulia Bernardini, Huiping Chen, Grigorios Loukides, Nadia Pisanti, Solon P. Pissis, Leen Stougie, and Michelle Sweering. String sanitization under edit distance. In Inge Li Gørtz and Oren Weimann, editors, *Proc. CPM 2020*, volume 161 of *LIPIcs*, pages 7:1–7:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

10 Karl Bringmann and Bhaskar Ray Chaudhury. Sketching, streaming, and fine-grained complexity of (weighted) LCS. In *Proc. FSTTCS 2018*, volume 122 of *LIPIcs*, pages 40:1–40:16, 2018.

11 Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. Truly sub-cubic algorithms for language edit distance and RNA-folding via fast bounded-difference min-plus product. In *Proc. FOCS 2016*, pages 375–384, 2016.

12 Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proc. SODA 2018*, pages 1216–1235, 2018.

13 Krishnendu Chatterjee, Thomas A. Henzinger, Rasmus Ibsen-Jensen, and Jan Otop. Edit distance for pushdown automata. In *Proc. ICALP 2015*, volume 9135 of *Lecture Notes in Computer Science*, pages 121–133. Springer, 2015.

14 Herman Z. Q. Chen, Sergey Kitaev, Torsten Mütze, and Brian Y. Sun. On universal partial words. *Electronic Notes in Discrete Mathematics*, 61:231–237, 2017.

15 Hyunjoon Cheon and Yo-Sub Han. Computing the shortest string and the edit-distance for parsing expression languages. In *Proc. DLT 2020*, volume 12086 of *Lecture Notes in Computer Science*, pages 43–54, 2020.

16 Hyunjoon Cheon, Yo-Sub Han, Sang-Ki Ko, and Kai Salomaa. The relative edit-distance between two input-driven languages. In *Proc. DLT 2019*, volume 11647 of *Lecture Notes in Computer Science*, pages 127–139, 2019.

17 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.

18 Maxime Crochemore, Borivoj Melichar, and Zdenek Tronícek. Directed acyclic subsequence graph - overview. *J. Discrete Algorithms*, 1(3-4):255–280, 2003.

19 Joel D. Day, Pamela Fleischmann, Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. The edit distance to k-subsequence universality. *CoRR*, abs/2007.09192, 2020. arXiv:2007.09192.

20 Joel D. Day, Pamela Fleischmann, Florin Manea, and Dirk Nowotka. k-spectra of weakly-c-balanced words. In *Proc. DLT 2019*, volume 11647 of *Lecture Notes in Computer Science*, pages 265–277, 2019.

21 Nicolaas G. de Bruijn. A combinatorial problem. *Koninklijke Nederlandse Akademie v. Wetenschappen*, 49:758–764, 1946.

22 Aldo de Luca, Amy Glen, and Luca Q. Zamboni. Rich, sturmian, and trapezoidal words. *Theor. Comput. Sci.*, 407(1-3):569–573, 2008.

23 Xavier Droubay, Jacques Justin, and Giuseppe Pirillo. Episturmian words and some constructions of de Luca and Rauzy. *Theor. Comput. Sci.*, 255(1-2):539–553, 2001.

24 Lukas Fleischer and Manfred Kufleitner. Testing Simon's congruence. In *Proc. MFCS 2018*, volume 117 of *LIPIcs*, pages 62:1–62:13, 2018.

25 Dominik D. Freydenberger, Pawel Gawrychowski, Juhani Karhumäki, Florin Manea, and Wojciech Rytter. Testing k-binomial equivalence. In Multidisciplinary Creativity*, a collection of papers dedicated to G. Păun 65th birthday*, pages 239–248, 2015. available in CoRR abs/1509.00622.

26 Harold N. Gabow and Robert Endre Tarjan. A linear-time algorithm for a special case of disjoint set union. In *Proc. 15th STOC*, pages 246–251, 1983.

27 Emmanuelle Garel. Minimal separators of two words. In *Proc. CPM 1993*, volume 684 of *Lecture Notes in Computer Science*, pages 35–53, 1993.

28 Pawel Gawrychowski, Maria Kosche, Tore Koss, Florin Manea, and Stefan Siemer. Efficiently testing Simon's congruence. *CoRR*, abs/2005.01112, 2020. arXiv:2005.01112.

29 Pawel Gawrychowski, Martin Lange, Narad Rampersad, Jeffrey O. Shallit, and Marek Szykula. Existential length universality. In *Proc. STACS 2020*, volume 154 of *LIPIcs*, pages 16:1–16:14, 2020.

30 Bennet Goeckner, Corbin Groothuis, Cyrus Hettle, Brian Kell, Pamela Kirkpatrick, Rachel Kirsch, and Ryan W. Solava. Universal partial words over non-binary alphabets. *Theor. Comput. Sci*, 713:56–65, 2018.

31 Simon Halfon, Philippe Schnoebelen, and Georg Zetzsche. Decidability, complexity, and expressiveness of first-order logic over the subword ordering. In *Proc. LICS 2017*, pages 1–12, 2017.

32 R. W. Hamming. Error detecting and error correcting codes. *Bell Syst. Tech. J.*, 29(2):147–160, 1950.

33 Yo-Sub Han, Sang-Ki Ko, and Kai Salomaa. The edit-distance between a regular language and a context-free language. *Int. J. Found. Comput. Sci.*, 24(7):1067–1082, 2013.

34 Jean-Jacques Hebrard. An algorithm for distinguishing efficiently bit-strings by their subsequences. *Theor. Comput. Sci.*, 82(1):35–49, 22 May 1991.

35 Daniel S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18(6):341–343, 1975.

36 Markus Holzer and Martin Kutrib. Descriptional and computational complexity of finite automata - A survey. *Inf. Comput.*, 209(3):456–470, 2011.

37 Hiroshi Imai and Takao Asano. Dynamic segment intersection search with applications. In *Proc. FOCS 1984*, pages 393–402, 1984.

38 Rajesh Jayaram and Barna Saha. Approximating language edit distance beyond fast matrix multiplication: Ultralinear grammars are where parsing becomes hard! In *Proc. ICALP 2017*, volume 80 of *LIPIcs*, pages 19:1–19:15, 2017.

39 Prateek Karandikar, Manfred Kufleitner, and Philippe Schnoebelen. On the index of Simon's congruence for piecewise testability. *Inf. Process. Lett.*, 115(4):515–519, 2015.

40 Prateek Karandikar and Philippe Schnoebelen. The height of piecewise-testable languages with applications in logical complexity. In *Proc. CSL 2016*, volume 62 of *LIPIcs*, pages 37:1–37:22, 2016.

41 Prateek Karandikar and Philippe Schnoebelen. The height of piecewise-testable languages and the complexity of the logic of subwords. *Log. Methods Comput. Sci.*, 15(2), 2019.

42 Markus Krötzsch, Tomás Masopust, and Michaël Thomazo. Complexity of universality and related problems for partially ordered NFAs. *Inf. Comput.*, 255:177–192, 2017.

43 Dietrich Kuske. The subtrace order and counting first-order logic. In *Proc. CSR 2020*, volume 12159 of *Lecture Notes in Computer Science*, pages 289–302, 2020.

44 Dietrich Kuske and Georg Zetzsche. Languages ordered by the subword order. In *Proc. FOSSACS 2019*, volume 11425 of *Lecture Notes in Computer Science*, pages 348–364, 2019.

45 Marie Lejeune, Julien Leroy, and Michel Rigo. Computing the k-binomial complexity of the Thue-Morse word. In *Proc. DLT 2019*, volume 11647 of *Lecture Notes in Computer Science*, pages 278–291, 2019.

46 Julien Leroy, Michel Rigo, and Manon Stipulanti. Generalized Pascal triangle for binomial coefficients of words. *Electron. J. Combin.*, 24(1.44):36 pp., 2017.

47 David Maier. The complexity of some problems on subsequences and supersequences. *J. ACM*, 25(2):322–336, April 1978.

48 Monroe H. Martin. A problem in arrangements. *Bull. Amer. Math. Soc.*, 40(12):859–864, December 1934.

49 Alexandru Mateescu, Arto Salomaa, and Sheng Yu. Subword histories and Parikh matrices. *J. Comput. Syst. Sci.*, 68(1):1–21, 2004.

50 Giovanni Pighizzini. How hard is computing the edit distance? *Inf. Comput.*, 165(1):1–13, 2001.

51 Jean-Eric Pin. The consequences of Imre Simon's work in the theory of automata, languages, and semigroups. In *Proc. LATIN 2004*, volume 2976 of *Lecture Notes in Computer Science*, page 5, 2004.

52 Jean-Eric Pin. The influence of Imre Simon's work in the theory of automata, languages and semigroups. *Semigroup Forum*, 98:1–8, 2019.

53 Narad Rampersad, Jeffrey Shallit, and Zhi Xu. The computational complexity of universality problems for prefixes, suffixes, factors, and subwords of regular languages. *Fundam. Inf.*, 116(1-4):223–236, January 2012.

54 Michel Rigo and Pavel Salimov. Another generalization of abelian equivalence: Binomial complexity of infinite words. *Theor. Comput. Sci.*, 601:47–57, 2015.

55 Jacques Sakarovitch and Imre Simon. Subwords. In M. Lothaire, editor, *Combinatorics on Words*, chapter 6, pages 105–142. Cambridge University Press, 1997.

56 Arto Salomaa. Connections between subwords and certain matrix mappings. *Theoret. Comput. Sci.*, 340(2):188–203, 2005.

57 David Sankoff and Joseph Kruskal. *Time Warps, String Edits, and Macromolecules The Theory and Practice of Sequence Comparison.* Cambridge University Press, 2000 (reprinted). originally published in 1983.

58 Baruch Schieber. Computing a minimum-weight k-link path in graphs with the concave monge property. In *Proc. SODA 1995*, pages 405–411. ACM/SIAM, 1995.

59 Shinnosuke Seki. Absoluteness of subword inequality is undecidable. *Theor. Comput. Sci.*, 418:116–120, 2012. doi:10.1016/j.tcs.2011.10.017.

60 Imre Simon. *Hierarchies of events with dot-depth one - Ph.D. thesis*. University of Waterloo, 1972.

61 Imre Simon. Piecewise testable events. In *Autom. Theor. Form. Lang., 2nd GI Conf.*, volume 33 of *LNCS*, pages 214–222, 1975.

62 Imre Simon. Words distinguished by their subwords (extended abstract). In *Proc. WORDS 2003*, volume 27 of *TUCS General Publication*, pages 6–13, 2003.

63 Zdenek Tronícek. Common subsequence automaton. In *Proc. CIAA 2002 (Revised Papers)*, volume 2608 of *Lecture Notes in Computer Science*, pages 270–275, 2002.

64 Robert A. Wagner. Order-n correction for regular languages. *Commun. ACM*, 17(5):265–268, 1974.

65 Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, January 1974.

66 Georg Zetzsche. The complexity of downward closure comparisons. In *Proc. ICALP 2016*, volume 55 of *LIPIcs*, pages 123:1–123:14, 2016.

Barrington Plays Cards: The Complexity of Card-Based Protocols

Pavel Dvořák ✉
Charles University, Prague, Czech Republic

Michal Koucký ✉
Charles University, Prague, Czech Republic

Abstract

In this paper we study the computational complexity of functions that have efficient card-based protocols. A study of card-based protocols was initiated by den Boer [6] as a means for secure two-party computation. Our contribution is two-fold: We classify a large class of protocols with respect to the computational complexity of functions they compute, and we propose other encodings of inputs which require fewer cards than the usual 2-card representation.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases Efficient card-based protocol, Branching program, Turing machine

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.26

Related Version *Previous Version*: `https://arxiv.org/abs/2010.08445` [7]

Funding The authors were supported by Czech Science Foundation GAČR grant #19-27871X.

Acknowledgements We thank Václav Blažej for suggesting how to use face-up cards to emulate red and blue card backs.

1 Introduction

A study of card-based protocols as a means for secure two-party computation was initiated by den Boer [6]. In this scenario, we have two players – Alice and Bob – who hold inputs x and y respectively. Their goal is to securely compute a given function f on those inputs. By secure computation, we mean that the players learn nothing from observing the computation except for what is implied by the output $f(x, y)$. Den Boer introduced a model where the inputs x and y are encoded by a sequence of playing cards and the players operate on the cards to compute the function. They can use additional cards for computation. In particular, den Boer showed how to securely compute AND of two bits using five cards in total.

Crépeau and Kilian [5] improved this results. They represent each input bit by two face-down cards: 1 is represented as ♡♣, and 0 as ♣♡. They provided a secure protocol for AND which takes two bits b_1 and b_2 represented by two face-down cards and outputs $b_1 \wedge b_2$ represented again by two face-down cards. Since NOT can be obtained by swapping the two cards representing a given bit this allows to use their technique to compute any function. This allow us to evaluate any Boolean circuit on the inputs by a protocol of length proportional to the size of the circuit and using a number of auxiliary cards that corresponds to the *width* of the circuit.

Nishida et al. [19] reduced the number of auxiliary cards to 6 for any Boolean function. For most functions, the protocol will be of exponential length as it essentially evaluates the DNF of f. Several other works studied the number of cards necessary for computing various elementary functions such as AND and XOR [8, 12, 10, 9, 19, 21, 17, 14, 16, 12, 1, 10].

Motivated by the question what can be efficiently computed by such protocols and how many cards one needs to compute various functions, in this work, we investigate secure *efficient* protocols that are protocols of polynomial length. Our contribution is two-fold: We classify a large class of protocols with respect to the computational complexity of functions they compute, and we propose other encodings of inputs which require fewer cards than the 2-card representation. We summarize our results next:

1. We show that *oblivious* protocols of polynomial length that do not modify their input (they are *read-only*) and use only a constant number of auxiliary cards compute precisely the functions in NC^1, the class of functions computed by Boolean circuits of logarithmic depth. (Alternatively, NC^1 is the class of functions computed by Boolean formulas of polynomial size.) By oblivious protocol we mean a protocol whose actions depend only on the current visible state.

2. Oblivious read-only protocols of polynomial length with a logarithmic number of auxiliary cards correspond to the class of functions computable by polynomial-size branching programs. (This class is also known as $\mathsf{L}/poly$, the non-uniform version of deterministic log-space.)

3. We also investigate protocols that use a constant number of auxiliary cards but are allowed to use the cards representing the input for their computation provided that they guarantee that by the end of the computation the input will be restored to its original value. We show that such protocols can compute functions that are believed to be outside of NC^1. For example, they can compute languages that are complete for NL, the non-deterministic log-space. Hence, read-only protocols are presumably weaker than protocols that may modify their input.

4. We study alternative encodings of inputs that are more efficient that the 2-card encoding. We look at 1-card encoding where 1 is represented by ♡ and 0 by ♣. In this encoding, Alice and Bob need only one card per bit to commit the bit. We show similar complexity results for this encoding as for the 2-card encoding: read-only protocols with a constant number of auxiliary cards are NC^1, with a logarithmic number of cards it is the non-uniform log-space, and if we allow using the input cards for computation we get potentially more powerful protocols.

   A disadvantage of the 1-card protocol is that it still needs a supply of n cards ♡ and n cards ♣ to represent any n-bit input. Although, if one restricted his attention to inputs that contain the same number of 1's and 0's, it would suffice to have $n/2$ cards ♡ and $n/2$ cards ♣. Such inputs form a substantial fraction of all n-bit inputs, they are $\Theta(2^n/\sqrt{n})$ many.

5. We propose a new 1/2-card encoding which requires only $n/2$ cards ♡ and $n/2$ cards ♣ to represent any n-bit input. The 1/2-card encoding is obtained from the 2-card encoding by removing from each pair of cards one card, in total one half of the ♡-cards and one half of the ♣-cards. There is an empty space left instead of each removed card. There is a way for each player to encode his input so that the other player learns no information about the opponent's input. We show that using this encoding we can simulate any read-only protocol that uses 2-card encoding. Hence, any NC^1 function on n bits can be securely computed using only $n + O(1)$ cards, counting also the input cards. We do not know how to securely perform protocols for 1/2-card encoding that would modify their input.

1.1 Previous Work

As mentioned above, a study of card-based protocols was initiated by den Boer [6] who introduced a secure 5-card protocol for computing AND. However, this protocol does not produce output in a face-down 2-card format, thus it can not be used for designing protocols for arbitrary function. Since then a lot of work was done in improving AND protocols and other primitive functions. Crépeau and Kilian [5] provided a 1-party card-based protocol where the player can pick a random permutation π with no fixed point and the player has no information about π. They introduced an AND protocol, which takes two bits b_1, b_2 represented in the 2-card format as input and outputs two cards which represent $b_1 \wedge b_2$ in face-down 2-card format. They also introduced a protocol for copying a bit in the 2-card format, which is used during simulation of circuits. Their protocols with a protocol for NOT (which is trivial) can be used for computing any Boolean function f and the number of used cards is at most linear in the size of a circuit (using AND and NOT gates) computing f. However, during the computation they use helping cards of other suits then ♣ and ♡. Niemi and Renvall [18] improved it and they designed a protocol computing arbitrary function f which uses only cards of suits ♣ and ♡. They also introduced a protocol to copy a single card with almost perfect security – the card suit is revealed only with a small probability. Such protocol cannot exist with perfect security as was proved by Mizuki and Shizuya [15]. The copying and AND protocols were further improved and simplified in [21, 17, 14, 16, 12, 1, 10].

Nishida et al. [19] proved that any Boolean function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ can be computed with $4n$ cards encoding the input and 6 additional helping cards. Mizuki [13] proved that at most $2n + 2$ is needed to compute AND of n bits. Francis et al. [8] provided protocols and tight lower bounds on the number of cards needed for computing any two-bit input two-bit output function. Other lower bounds for AND of 2 bits and of n bits in various regimes were provided by Koch et al. [12, 10] and by Kastner et al. [9].

The security of card-based protocols is provided by shuffling the cards – one player shuffles the cards (applies some random permutation to them) in a way so that the other player has no information about the new order of shuffled cards. Koch et al. [12] provided a 4-card AND protocol. However, they used a non-uniform distribution for picking a random permutation, which is difficult to perform by humans. Nishimura et al. [20] suggested an "easy-for-human" procedure how to apply a shuffling permutation picked from a non-uniform distribution using envelopes.

Koch and Walzer [10] studied private function evaluation. In this setting, Alice has an encoding of some program P which computes a function f. They studied various models, thus the program P can be a Turing machine, RAM, a circuit, or a branching program. Bob has an encoding of a string x and their goal is to compute securely $f(x)$. (Again, after the computation they know only $f(x)$ and they have no other information about each other input, i.e., Alice has no other information about x and Bob has no other information about f.) They proved that such universal computation exists for those models. They did not study the complexity of such simulations. Whereas, we study complexity of computing a function f when the input to f is divided between Alice and Bob.

One can distinguish two types of attack.

1. Passive: honest-but-curious player – she follows the protocol but she wants to retrieve as much information as possible about the other player input.
2. Active: malicious player – she can deviate from the protocol.

Koch and Walzer [11] proved that if a passive-secure protocol Π uses only uniform closed shuffles (each shuffling permutation is picked uniformly from some permutation group) then the protocol Π can be transformed into an active-secure protocol.

2 Preliminaries

2.1 Card-based Protocols

In this section we define *card-based protocols* which securely compute some Boolean function on a joint input of Alice and Bob. Alice gets an input $x \in \{0,1\}^n$ and Bob gets an input $y \in \{0,1\}^n$, and their goal is to compute $f(x,y)$ for some function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, while not revealing anything about their input to the other player. The protocol proceeds first by Alice and Bob committing their input into a sequence of cards, and then operating on the cards together with some auxiliary cards. At the end of the protocol, the players learn the output $f(x,y)$.

In this section we consider the usual *2-card encoding* of the input, where each input is represented as a sequence of cards, two cards per bit: value 1 is represented by $\heartsuit\clubsuit$ and value 0 is represented by $\clubsuit\heartsuit$ where the cards are put face-down on the table. Hence each player needs $2n$ cards to commit his input. All the cards have the same back, say blue. In the beginning, face-down cards representing the player inputs are in front of the players. Between them, there is a *deck* of s prescribed auxiliary cards of $\heartsuit$ and $\clubsuit$. There is available some empty space on the table to operate with the cards. We assume that the cards are placed on the table in some specific positions (locations), numbered $1,\ldots,m$, where:

- $1,\ldots,2n$ are positions of Alice's input cards,
- $2n+1,\ldots,4n$ are positions of Bob's cards,
- $4n+1,\ldots,4n+s$ are the initial positions of the helping cards in the deck,
- $4n+s+1,\ldots,m$ are initially empty positions.

We call the positions $1,\ldots,4n$ as the input positions and the remaining positions as the *work space*. We say a position is *occupied* if there is a card on it, otherwise, it is *empty*. We denote an empty position by $\times$. Let $q = m - 4n$ denote the amount of the work space. We assume $q = O(s)$. Thus, there are $4n+s$ cards on the table and $4n+q = m$ positions.

The players can move their input cards and cards from the deck to the work space and back. Formally, the basic *actions* which can be executed by the players are:

Move(p,i,j): The player p moves a card from the position i to position j.

Shuffle(p,T,Γ): The player p applies a random permutation from Γ to the cards on the table on positions $T \subseteq \{4n+1,\ldots,m\}$.

Turn(p,i): The player p turns the i-th card on the table face-up if it is face-down, and vice versa.

The protocol specifies which action to take next based on the sequences of visible states seen on the table so far. The current visible state of the table is what an external viewer could observe, that is which positions are currently occupied and what is the top of each card laying on the table. If there are c distinct cards then there are at most $(c+2)^m$ distinct visible states. Hence, based on the sequence of visible states from the beginning of the game the protocol specifies which action to take next or whether to end. In the end, the protocol specifies which cards represent the output of the run of the protocol. (They might be face-down.) We say the protocol Π *computes* a function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ if for all inputs $x,y \in \{0,1\}^n$, on the inputs x and y the protocol outputs $f(x,y)$. The length of the protocol is the maximum number of actions executed by the protocol over all inputs $(x,y) \in \{0,1\}^n \times \{0,1\}^n$ and all possible outcomes of shuffling. We say that a protocol is *oblivious* if the action executed next depends only on the current visible state and the number of actions taken so far.

The shuffling operation provides randomness for the execution of the protocol. Hence, the sequence of visible states the protocol passes through is a random variable. We will say that a protocol is *secure* if for any pair of inputs (x,y) and (x',y') to Alice and Bob, where

$f(x, y) = f(x', y')$, the distribution of the sequence of visible states of the protocol on inputs (x, y) and (x', y') is the same. Notice, that this implies that neither of the players learns anything about the input of the other player except for what is implied by $f(x, y)$.

Often we will be interested in protocols that provide their output encoded in face-down cards. In such a scenario we will require for the security of the protocol that the distributions of visible states during the protocol will be identical for all input pairs (x, y).

We say the protocol is *robust* if a cheating player, that is a player who deviates from the protocol, is either caught by reaching an invalid visible state (where cards have unexpected values or positions) or the distribution of visible states does not leak any information about the other player input except for what would be leaked by honest players. In particular, if say Bob is cheating and Alice is honest, for a robust protocol we require that for any two inputs x, x' of Alice and any input y of Bob, where $f(x, y) = f(x', y)$, the distribution of the sequence of visible states during the game on inputs (x, y) and (x', y) is the same. We will be designing only robust oblivious protocols.

We say the protocol is *read-only* if the value of cards placed on the input positions $1, \dots, 4n$ is always the same whenever a position is occupied.

Let s-SP be the class of function families $\{f_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}\}_{n\geq 0}$ for which we have a sequence of secure read-only oblivious protocols, one for each n, which are of length polynomial in n, with deck size s and work space size $2s$. (At the beginning the first s work space positions are occupied by the deck of cards, and the remaining s positions are empty). We might allow s to be a function of n. We define a class of secure polynomial-length protocols $\mathsf{SP} = \bigcup_{s\geq 1} s\text{-}\mathsf{SP}$. That is a function belongs to SP if it has polynomial length protocols which use a constant number of auxiliary cards and constant size work space.

2.2 Branching Programs

A *branching program* B is a directed acyclic graph G such that each vertex has out-degree either 2 or 0. The set of edges E of the graph G is split into two sets, zero-edges E_0 and one-edges E_1, in such a way that every vertex v of out-degree 2 is incident to exactly one outgoing zero-edge and exactly one outgoing one-edge. Each vertex of out-degree 2 is labeled by an index $\ell \in [n]$, by $[n]$ we mean a set $\{1, \dots, n\}$.

A branching program B is *layered* if the vertices are partitioned into layers $L_1, \dots, L_d$. The edges go only from a layer L_i to a layer L_{i+1} (for all $i < d$). Vertices of out-degree 0 are exactly vertices in the layer L_d. The number of layers d is the *length* of B and the *width* w of B is the maximum size of its layers, i.e., $w = \max_i |L_i|$.

A layered branching program is *oblivious* if vertices in the same layer have the same label. A branching program is a *permutation branching program* if each layer has exactly w vertices and for every two consecutive layers L_j and L_{j+1} zero-edges and one-edges form matching M_j^0 and M_j^1, respectively. We can view the matchings M_j^0 and M_j^1 as two permutations $\pi_j^0, \pi_j^1 : [w] \to [w]$. Note that we can rearrange all layers such that all permutations π_j^0 are identities.

One vertex of in-degree 0 is an *initial* vertex $\bar{v}$. Some vertices of out-degree 0 are denoted as *accepting* vertices. The computation of a branching program B on an input string $x \in \{0,1\}^n$ proceeds as follows. It starts in the initial vertex $\bar{v}$ which is the first *active* vertex. Suppose v is an active vertex and $\ell \in [n]$ is the label of v. If the out-degree of v is 2, then the next active vertex is determined by the zero- or one-edge according to the value of x_ℓ. More formally, let $e = \{v, v'\}$ be the edge in E_{x_ℓ}. Then, the vertex v' is the new active vertex. We repeat this procedure until a vertex u of out-degree 0 is reached. An input $x \in \{0,1\}^n$ is accepted if and only if u is an accepting vertex. The branching program B computes a function $f : \{0,1\}^n \to \{0,1\}$ if it accepts exactly those $x \in \{0,1\}^n$ such that $f(x) = 1$.

The class of functions PB contains all the functions computable by layered branching programs of constant width and polynomial length. A permutation branching program is *restricted* if it has exactly one accepting vertex v_{acc} and exactly one rejecting vertex v_{rej} in the last layer L_d. The computation of a restricted permutation branching program ends always in the vertices v_{acc} or v_{rej} and it accepts an input if it ends in the accepting vertex v_{acc}. A class w-PBP contains Boolean functions which are computable by restricted permutation branching programs of width w and polynomial length. We use the famous Barrington's theorem [2], which says that constant-width (permutation) branching programs are as powerful as NC^1-circuits.

▶ **Theorem 1** (Barrington [2]). $\mathsf{PB} \subseteq \mathsf{NC}^1 \subseteq 5\text{-}\mathsf{PBP}$.

3 Simulating Branching Programs

In this section, we prove one of our main theorems that read-only oblivious protocols of polynomial length that use constant work space compute the same functions as polynomial-size constant-width branching programs.

▶ **Theorem 2.** $\mathsf{SP} = \mathsf{NC}^1$.

To simulate a branching program by SP-protocol we need an oblivious implementation of copying a bit in the committed 2-card format. We use a procedure by Crépeau and Kilian [5]. It is straightforward to implement the procedure to be oblivious. We state the proof of the following theorem for the sake of completeness.

▶ **Theorem 3.** *There is a secure oblivious protocol that takes a bit b in 2-card representation placed in the work space and produces two 2-card copies of the bit in the work space. The protocol needs an auxiliary deck with three cards $\heartsuit$ and three $\clubsuit$ with the same back as the input bit.*

Proof.

1. Alice arranges the cards from the auxiliary deck face-up to create the following configuration.

$$\underbrace{?\ ?}_{b}\ \heartsuit\clubsuit\heartsuit\clubsuit\heartsuit\clubsuit$$

2. She turns the last six cards. Both, Alice and Bob, apply a random cyclic shift (denoted by $\langle , \rangle$) to them.

$$\underbrace{?\ ?}_{b}\langle\underbrace{?\ ?}_{b'}\underbrace{?\ ?}_{b'}\underbrace{?\ ?}_{b'}\rangle$$

3. They apply a random cyclic shift to the first four cards.

$$\langle ?\ ?\ ?\ ?\rangle\underbrace{?\ ?}_{b'}\underbrace{?\ ?}_{b'}$$

4. She turns the first four cards face-up.
   a. If the sequence is alternating (i.e., $\heartsuit\clubsuit\heartsuit\clubsuit$ or its shift) then $b = b'$. Thus, the last 4 cards represent two copies of b.

$$\heartsuit\clubsuit\heartsuit\clubsuit\underbrace{?\ ?}_{b}\underbrace{?\ ?}_{b}$$

b. Otherwise (i.e., $\heartsuit\clubsuit\clubsuit\heartsuit$ or its shift) then $b = 1 - b'$. Thus, the last 4 cards represent two copies of negation of b. In that case, she switches the fifth with the sixth card and the seventh with the eighth card to represent two copies of b as well.

$$\heartsuit\clubsuit\clubsuit\heartsuit\underbrace{?\ ?}_{1-b}\underbrace{?\ ?}_{1-b}$$

Alice and Bob might want to turn over and shuffle the first four left-over auxiliary cards after step 4. The last four cards represent two copies of b face-down in the 2-card format. To make the protocol oblivious we must implement both 4.a) and 4.b) by the same number of actions. To do so we include additional actions in 4.a) which have no effect such as shuffling a single card.

It is clear the described protocol is secure. The only step where they can gain some information about b is Step 4, when Alice turns some cards. However, the cyclic shifts in Steps 2 and 3 were done by both players. Thus, Alice reveals the alternating sequence ($\heartsuit\clubsuit\heartsuit\clubsuit$) in Step 4 with the probability exactly $\frac{1}{2}$ (independently on the value of b) even if one of the players would be cheating. Thus, the protocol is secure. ◀

To prove $\mathsf{NC}^1 \subseteq \mathsf{SP}$ we use as the first step Barrington's theorem [2]. By Barrington's theorem, each function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ from NC^1 can be computed by a polynomial length width-5 restricted permutation branching program. We will build a protocol that simulates the actions of the branching program layer by layer. We will keep track of the image of the initial vertex of the branching program. For that, we will use five cards $\heartsuit\clubsuit\clubsuit\clubsuit\clubsuit$, where the position of $\heartsuit$ corresponds to the image of the initial vertex (the active vertex), and we will apply the permutations prescribed by the branching program on those five cards. If the input variable assigned to a particular level of the branching program is set to 1 we are expected to perform the permutation otherwise we are supposed to do nothing, i.e., apply the identity permutation. Any permutation can be decomposed into a sequence of simple transpositions (swaps) so we will use swaps conditioned by the input variable to either permute the five cards or leave them the way they are. We will implement the following primitive: Alice and Bob want to conditionally swap two cards α, β according to the value of bit b represented in the face-down 2-card form in the work space without revealing the value of b. They also want to make sure that if $b = 1$ the swap occurs and if $b = 0$ the swap does not occur.

▶ **Theorem 4.** *Let $\tilde{\alpha}, \tilde{\beta}$ be two sequences of face-down cards of the same length in the work space, and let $\gamma\,\delta$ be a face-down 2-card representation of b in the work space. There is a secure oblivious protocol such that during the protocol players swap the sequences α and β if and only if $b = 1$. The protocol uses two auxiliary cards $\clubsuit$.*

Proof. The swapping protocol works as follows.

1. Alice rearranges the input cards together with two auxiliary face-up cards $\clubsuit$ as follows:

   $$\clubsuit\gamma\ \tilde{\alpha}\ \clubsuit\delta\ \tilde{\beta}$$

   Thus, if $b = 0$ we have $\clubsuit\clubsuit\tilde{\alpha}\clubsuit\heartsuit\tilde{\beta}$ and if $b = 1$ we have $\clubsuit\heartsuit\tilde{\alpha}\clubsuit\clubsuit\tilde{\beta}$. The players do not know which situation are they in.
2. Both, Alice and Bob, apply a random cyclic shift to the cards, e.g.:

   $$\tilde{\alpha}\ \clubsuit\delta\ \tilde{\beta}\ \clubsuit\gamma$$

3. Alice turns the cards γ and δ representing b face-up. She knows what cards to turn, as the cards γ and δ are preceded by ♣ face-up. At the end she reorders the sequence (keeping the cyclic order) so that ♣♣ are the first cards, e.g.:

   $$\tilde{\alpha}\clubsuit\heartsuit\tilde{\beta}\clubsuit\clubsuit \to \clubsuit\clubsuit\tilde{\alpha}\clubsuit\heartsuit\tilde{\beta}$$

   If $b = 0$ then $\gamma\delta = \clubsuit\heartsuit$ and the sequences $\tilde{\alpha}\tilde{\beta}$ are not swapped. On the other hand if $b = 1$ then $\gamma\delta = \heartsuit\clubsuit$ and the sequences are swapped.

Note that the cards in $\tilde{\alpha}$ and $\tilde{\beta}$ are face-down during the whole protocol. It is also clear that this is a secure and robust protocol, and it can be implemented obliviously. In Step 3 the cards δ and γ representing the bit b are revealed. However, because of random cyclic shifts in Step 2 (again done by both players), these cards are in the order ♡♣ with probability $\frac{1}{2}$, independently of the value of b. Thus the swapping protocol is secure. ◀

▶ Remark 5. We point out that there is a more efficient swap protocol that does not use any auxiliary card. In step 2 Alice and Bob could perform random bisection cut introduced by Mizuki and Sone [17]. During the random bisection cut, they split the deck into two halves of the same number of cards and then they swap them or not (at random). Thus, in step 3 of the swap protocol, they would know which cards they should turn, even without the face-up ♣ cards. Mizuki and Sone [17] also introduced a more more efficient copy protocol (it uses 6 cards instead of 8) but it also uses the random bisection cut. We prefer the standard random cut as it is easier to cheat during the random bisection cut.

Now we prove the first inclusion of Theorem 2.

▶ **Theorem 6.** 5-PBP $\subseteq$ SP.

Proof. Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be a function in 5-PBP. Then, the function f can be computed by a branching program P with the following properties:

1. Each layer has exactly 5 vertices. The input vertex is the first vertex in the first layer. The computation ends either in the accepting vertex v_{acc} or in the rejecting vertex v_{rej}.
2. The permutation from each layer i to the layer $i+1$ corresponding to 0 is the identity.

Alice and Bob represent the first layer as ♡♣♣♣♣, each card represents one vertex in the layer. The card ♡ represents the active vertex in the layer (the initial vertex in the first layer). We call these 5 cards the *program cards.* Alice and Bob put the program cards at the work space and turn them face-down. Alice and Bob simulate the program P layer by layer. They apply permutations determined by P to the program cards according to the player's input bits. Suppose we have a representation of the active vertex in the i-th layer and we want to calculate the active vertex at the $(i+1)$-th layer. Without loss of generality the label of the i-th layer is Alice's bit x_ℓ (otherwise the roles of Alice and Bob are reversed). Thus, we want to apply some permutation $\rho_i \in S_5$ to the program cards if $x_\ell = 1$ and keep the order of the program cards if $x_\ell = 0$.

We decompose the permutation ρ_i into transpositions $\tau_1 \circ \cdots \circ \tau_r$. For $j = 1, \ldots, r$, Alice will apply the transposition τ_j to the program cards. She runs the protocol Γ_1 of Theorem 3 to get cards γ, δ representing her bit x_ℓ in the work space. More formally, after the execution of Γ_1, there are two pairs of cards such that each pair represent the bit x_ℓ in the 2-card format. She puts one pair back to the input positions, i.e., the protocol Π is indeed read-only. We denote the cards of the second pair as γ and δ. Alice will use them for a conditional swap. She runs the protocol Γ_2 given by Theorem 4 (applied to the cards γ, δ and to the two program cards which should be affected by the transposition τ_j). That is, Alice swaps the two cards that τ_j is acting on if and only if $x_\ell = 1$. After applying this procedure for all transpositions $\tau_1, \ldots, \tau_r$, the permutation ρ_i got applied to the program cards if and only if $x_\ell = 1$ (otherwise the order of the program cards does not change).

Alice and Bob repeat this procedure for each layer of the branching program. Let α be the card representing the accepting vertex v_{acc} and β be the card representing the rejecting vertex v_{rej} at the end of the simulation. The cards α, β represent the output of the program. If the input is accepted, then the accepting vertex v_{acc} is active at the end of the simulation and thus the card α has suit $\heartsuit$ and the card β has suit $\clubsuit$. Thus, the cards α, β represent 1. On the other hand, if the input is rejected, then the rejecting vertex v_{rej} is active. Thus, the cards α, β have suits $\clubsuit$ and $\heartsuit$, respectively, and they represent 0.

The protocol Π is clearly SP-protocol as the players only sequentially apply the copying protocol Γ_1 and the swapping protocol Γ_2 to the program cards. We claim the simulation protocol Π is secure. Both protocols Γ_1 and Γ_2 are secure. The other helping cards which are not used only during a single run of Γ_1 or Γ_2 are exactly the program cards. They are placed face-up from the deck and then turned face-down for the rest of the protocol. Thus, the program cards could only reveal the output $f(x, y)$ at the end of the protocol (if they are turned face-up) and no other information. Therefore, we conclude that protocol Π is secure. ◀

Now we prove the opposite inclusion of Theorem 2.

▶ **Theorem 7.** $\mathsf{SP} \subseteq \mathsf{PB}$.

Proof. Consider a family of functions $\{f_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}\}_{n \geq 0}$ for which we have a sequence of secure read-only oblivious protocols, one for each n, which are of length polynomial in n, with deck size s and work space size $2s$. Let c be the number of different cards used by the protocol. At any moment, the work space can be in at most $(2c+1)^{2s}$ different states which we call the internal states of the protocol. For any $n \geq 1$ we will build a width-$(2c+1)^{2s}$ branching program of the same length T_n as the protocol for f_n. Each layer of the branching program consists of vertices where each vertex corresponds to one internal state of the protocol. We need to define edges between the layers of the branching program.

Let v be a vertex at layer $t \in \{0, \ldots, T_n - 1\}$. It corresponds to some internal state which in turn determines a visible state that together with t determines the action taken by the protocol at such a state. We define the edges based on the type of that action. If the action is a move of a card from some input position into the work space then node v queries the value of the corresponding input variable and the outgoing edges lead to nodes corresponding to internal states that reflect a move of the card into the work space. If the action is a shuffle operation then node v queries variable x_1 and irrespective of its value both outgoing edges go to the node in the next layer corresponding to the internal state obtained by applying one of the allowed permutations. (The particular choice of the permutation does not matter.) Similarly, for a turn of a card or a move of a card within the work space or out from the work space, the edges will go into a node that reflects the internal state after the move. Note that during an execution of a secure read-only protocol, the players never turn any card at an input position. Thus, the suit of a turned card is always determined by the current internal state. Therefore, we can add edges for a turn action without querying any specific input bit. In the last layer, we designate vertices that correspond to accepting states of the protocol as accepting all other nodes will be rejecting.

It should be clear from the construction that the resulting branching program computes f_n and has the required properties. ◀

Theorem 2 is a corollary of Theorems 6 and 7 and Barrington's theorem (Theorem 1).

4 Simulating Turing Machines

In this section, we will look at computation that obliviously and securely computes on committed inputs in 2-card representation. The exact split of the input between Alice and Bob is irrelevant in this section so we assume that the total length of the input is n bits. The protocols are expected to preserve the committed inputs: Although they may be allowed to modify the committed input during the computation, by the end of the computation the committed input must be restored to its original form. The protocols do not leak any information about the committed inputs except for what can be derived from the output cards if they are inspected. The protocols can be carried out by either player. To guarantee robustness and security shuffle operations should be always done by both players. (We use only uniformly random shuffle and random cyclic shift so performing them twice does not change their output distribution.)

▶ **Theorem 8.** *Let $s(n) \geq \log n$ be a non-decreasing function. Let f be a function computable by a Turing machine in space $s(n)$. Then f is in $O(s(n))$-*SP.

Let $SEL : \{0,1\}^3 \rightarrow \{0,1\}$ be the selection function that $SEL(c,b,a) = a$ if $c = 0$ and $SEL(c,b,a) = b$ otherwise.

Proof. We describe the algorithm for the protocol in high-level form and leave details of the construction to the interested reader. Let f be computable by a Turing machine M. Without loss of generality, we assume M uses a binary alphabet, on inputs of length n it uses work space exactly $s(n)$ bits and computes for $t(n)$ steps. The output of M is determined by the first bit of its work tape. We will simulate the computation of M step by step.

The protocol will use $8s(n) + 4\log n + O(1)$ auxiliary cards. Two blocks w and w' will represent $2s(n)$ bits, each, and two blocks p and j will represent $\log n$ bits each. In addition to that there is a block q of $O(1)$ bits, and some additional auxiliary bits. We need a constant number of positions to be empty. All the bits are encoded in 2-card representation. The block w represents the content of the work tape of M, 2 bits per tape cell, where the second bit indicates the presence of the work tape head on that particular tape cell. The block p encodes in binary the current position of the input head of M. The block q encodes the internal state of M.

The protocol simulates one step of M as follows: first, it determines the value b of the bit scanned by the input head, then it calculates into w' the content of the work tape of M after this step. Then it updates the internal state, the input head position p, and switches w' and w.

To determine b the protocol looks at each input bit x_i one by one and records the one that has an index corresponding to p. Set b to 0. For $i = 1, \ldots, n$, the protocol copies x_i into some work space b', it sets j to represent i, obliviously compares p and j while recording the result into c. (Comparing bit strings can be done by an NC^1 circuit so there is an oblivious protocol for that of poly-logarithmic length.) From c, b and b' we can calculate the new value of b by evaluating $SEL(c, b, b')$. This can be done obliviously. After processing all the input bits, b has the value of the currently scanned input bit.

Now, we can determine w', the content of the work tape of M after this step of the computation. We compute w' cell by cell. The value of each cell is a function of the input bit b, M's state q, and the previous content of the cell in w together with the content of adjacent cells. Hence, the value of each bit of w' is a function of constantly many bits and can be computed obliviously.

After computing w', we can also calculate d, the direction in which the input head of M should move, and the new state q' of M. This can be done by scanning w for the work tape position, and recording the relevant information for q' and d when we pass over the current work cell similarly to determining the value of the input bit b.

From p and d, we obliviously calculate the next position of the input head into j. (Each bit of j can be computed by an NC^1 circuit from p and d.) Finally, we switch the contents of w and w', p and j, and q and q'.

We repeat this procedure $t(n)$ times. In the end, the first bit of w indicates the output of M.

As each step of the computation can be implemented securely, obliviously, and robustly, we obtain a secure, oblivious, and robust protocol for f that uses $O(s(n))$ work space and $O(s(n))$ auxiliary cards. ◀

As the card-based protocols allow for non-uniformity by protocols using $O(\log n)$ work space we can simulate not only log-space Turing machines but also polynomial-size branching programs (the *non-uniform log-space*). The above proof can be extended to Turing machines taking advice: the protocol can provide the advice bit by bit during the phase when the input is scanned bit by bit to determine b. (We assume that the advice is provided to the Turing machine on bit positions with index $> n$. For those positions instead of copying the non-existent input bits, the protocol hardwires the appropriate bit into b'. As the advice is the same for each input, this can be done publicly.)

By essentially the same proof as Theorem 7 we can obtain a simulation of oblivious, read-only secure protocols that use a logarithmic amount of work space by branching programs of polynomial size. Let $O(\log n)\text{-}\mathsf{SP} = \bigcup_k (k + k\log n)\text{-}\mathsf{SP}$. We get:

▶ **Theorem 9.** *The class of functions computable by polynomial-size branching programs equals to $O(\log n)$-SP.*

4.1 Read-write Protocols

So far we have looked only at read-only protocols. If we remove the condition to be read-only we get a potentially larger class of functions computable by such protocols. When the protocol is not read-only, we still require the protocol to restore its input into the original state by the end of the computation. We also require the protocol to be secure so not to leak any information about the input except for what is implied by the protocol output cards.

We give examples of functions that can be computed by protocols modifying their input which we conjecture are outside of the read-only protocol class with similar bound on the work space. Proving this conjecture would amount to separating NC^1 from log-space, a major open problem in complexity theory.

Let $s(n)$ be a non-decreasing function such that $\log n \leq s(n) \leq n/2\log^* n$. Let $g : \{0,1\}^n \to \{0,1\}$ be in NC^1, and $h : \{0,1\}^{n-s(n)\log^* n} \to \{0,1\}$ be a function computable by a Turing machine in space $O(s(n))$ and polynomial time. Define $f : \{0,1\}^n \to \{0,1\}$ as follows:

$$f(x) = \begin{cases} g(x) & \text{if } x_{n+1-s(n)\log^* n} \cdots x_n \neq 0 \cdots 0, \\ h(x_1 \cdots x_{n-s(n)\log^* n}) & \text{otherwise.} \end{cases}$$

▶ **Theorem 10.** *The function f defined above is computable by secure robust oblivious protocols of polynomial length that use a constant amount of work space.*

Proof. The protocol for f proceeds as follows. It first computes the OR of the input bits $x_{n+1-s(n)\log^* n} \cdots x_n$ using a protocol for NC^1 functions where the output c of the protocol is encoded in 2-card representation in its work space. Then it computes the value g of $g(x)$

encoded in 2-card representation in the work space. Finally, it uses the cards representing input bits $x_{n+1-s(n)\log^* n} \cdots x_n$ to simulate the computation of a Turing machine for h as in the proof of Theorem 8. The simulation is done so that if $c = 1$ then nothing is done to the input (the simulation is vacuous) and if $c = 0$ the simulation is really happening. The simulation uses the input bits $x_{n+1-s(n)\log^* n} \cdots x_n$ to store w, w', p and j (from the simulation), everything else is done in the actual work space of constant size. Whenever the simulation wants to write some value a into an input position used for the simulation, it copies the value into the work space, it copies there the current value d of the destination position, computes $\mathit{SEL}(c, a, d)$ and replaces cards in the destination by the output of $\mathit{SEL}(c, a, d)$. (Hence, if $c = 0$ nothing has happened.) Reading a value can be done by copying the particular bit into the work space and then working with the copied cards. This way the input is undisturbed if $c = 1$ and it is overwritten if $c = 0$. At the end of simulation, the protocol copies the output bit h, which is the first bit of w into the actual work space, and writes value 0 to all input bits $x_{n+1-s(n)\log^* n} \cdots x_n$ conditionally on $c = 0$. (Bit values read from the storage, taking part in the vacuous computation, will be the same throughout the computation. So the simulations of the NC^1 circuits implementing various steps of the computation will be secure.)

Finally, the protocol computes $\mathit{SEL}(c, h, g)$ which is the output of the protocol. All parts of the protocol can be done securely and obliviously. (This is true also when $c = 0$ and the simulation of the Turing machine is bogus.) The protocol restores its committed input by the end of the computation. ◀

One can also use the technique of catalytic computation to construct protocols for functions not know to be in NC^1. Buhrman et al. [3, 4] show how to use memory which contains some information for computation while restoring the memory to its original content by the end of the computation. For example, they can solve the connectivity on directed graphs this way, the problem $\mathit{CONN}(G)$: Given an $n \times n$ adjacency matrix of a directed graph G decide whether there is a path from vertex 1 to vertex n. They present a polynomial-size program for $\mathit{CONN}(G)$, which uses $3n^2 + 1$ work registers and n^2 input registers, each holding one bit of information. The program consists of instruction of the form

$$r_i \leftarrow r_i \oplus u \cdot v,$$

where u and v are arbitrary registers different from r_i or constants 0 and 1. The program is oblivious, so it is a straight line program consisting of such instructions. The program has the property that all registers are guaranteed to have the initial value by the end of the computation except for one specified work register which contains the output value. It is straightforward to implement each such an instruction by secure and robust protocol since the instructions are computable in NC^1.

This allows to design an oblivious, secure protocol of polynomial length with constant work space for a function $f' : \{0,1\}^{4n^2} \to \{0,1\}$ that is defined as: $f'(G_1, G_2, G_3, G_4) = 1$ if and only if from the vertex 1 we can reach the vertex n in each of the graphs represented by adjacency matrices G_1, G_2, G_3 and G_4. Such a function is unlikely to be contained in NC^1, as $\mathit{CONN}(G)$ is known to be complete for *nondeterministic* log-space computation. Hence, it is unlikely that protocols that are allowed to modify their input could be simulated by read-only protocols using similar resources.

5 More Efficient Input Encodings

5.1 1-Card Encoding

In this section, we consider other ways how Alice and Bob can commit their input which use fewer cards. The first natural encoding is to represent each bit 1 by face-down card ♡ and bit 0 by ♣. These cards would be stored in front of the players in input positions $1, \dots, 2n$. Whenever the players want to operate with the committed bit they need to extend it to 2-card representation.

There are two ways we know how to do it. Niemi and Renvall [18] gave a protocol that is able to extend the bit without knowing its value. However, there is a small probability of leaking the value of the bit being extended. The probability is inversely proportional to the number of cards used for the protocol. Hence, one would need a large number of helping cards in order to make sure that the probability of leaking information is negligible. That would erase any savings from the 1-card representation.

The other way which we use here is to allow the player who owns the particular input bit to extend it using a designated deck of two face-down cards ♣ and ♡. Once the bit is extended it can be copied by the protocol from Theorem 3, the first card of the first copy can be put back in the input position, the second card can be put back into the auxiliary deck, and the second copy can be used for further computation. The auxiliary deck containing the same cards as earlier should be shuffled by both players at the end of this procedure. The protocol is robust since a player cheating by extending the input bit by a wrong card will be caught in Step 4 of the copying protocol.

For this procedure, we need to augment our set of actions by the action of extending a bit by a complementary card from a designated deck. This action can be performed by shuffling the auxiliary deck, then peeking at the value of the card we are extending, and selecting the complementary card by peeking at each card in the deck.

With this operation in mind, we need to extend the definition of protocol security. We say a protocol is *secure from Alice* if for any pair of inputs (x, y) and (x, y') to Alice and Bob, the distribution of the sequence of visible states of the protocol together with the sequence of cards seen by Alice while peeking at them during the extension action on inputs (x, y) and (x, y') is the same. Similarly, the protocol is *secure from Bob* if for any pair of inputs (x, y) and (x', y) to Alice and Bob, the distribution of visible states and cards peeked at by Bob will be the same on both inputs (x, y) and (x', y). The protocol is secure if it is secure from both Alice and Bob.

Using the extension action we can perform all read-only protocols that used the 2-card bit commitment of inputs even for inputs committed in 1-card representation. They will be secure as long as the player performing each extension is the owner of the input bit as seeing his/her input bits does not affect the security definition. Hence, the power of the model stays essentially the same with this modification.

Security becomes more of an issue for protocols that are allowed to modify their inputs. Yet, we can prove a result similar to Theorem 10 for slightly modified function f'. Let $g : \{0,1\}^n \to \{0,1\}$ be in NC^1, and $h : \{0,1\}^{n-s(n)\log^* n} \to \{0,1\}$ be a function computable by a Turing machine in space $O(s(n))$ and polynomial time, where $\log n \le s(n) \le n/2\log^* n$.. Define $f' : \{0,1\}^n \to \{0,1\}$ as follows:

$$f'(x) = \begin{cases} g(x) & \text{if } x_{n+1-s(n)\log^* n} \cdots x_n \neq 0101\cdots01, \\ h(x_1 \cdots x_{n-s(n)\log^* n}) & \text{otherwise.} \end{cases}$$

We assume $s(n)\log^* n$ is even, and the first half of $x_{n+1-s(n)\log^* n} \cdots x_n$ is held by Alice and the other half by Bob. The other bits can be split between the players arbitrarily.

▶ **Theorem 11.** *The function f' defined above is computable by secure robust oblivious protocols of polynomial length that use a constant amount of work space and 1-card encoding of input bits.*

Proof. The protocol for f' proceeds similarly to the one in Theorem 10. It first verifies whether the input bits $x_{n+1-s(n)\log^* n} \cdots x_n$ differ from $0101 \cdots 01$ (assuming their number is even) using protocol for NC^1 functions. The output c of the verification is encoded in 2-card representation in the work space. Then the protocol computes the value g of $g(x)$ encoded in 2-card representation in the work space. Up until this point, we use the protocol described above to extend input bits into 2-card representations by the player who owns the input bit.

Now we want to use the cards representing input bits $x_{n+1-s(n)\log^* n} \cdots x_n$ to simulate computation of a Turing machine M for computing $h = h(x_1, \dots, x_{n-s(n)\log^* n})$ as in the previous proofs. We will use these input cards for storage when $c = 0$ and when $c = 1$ we will keep them intact. In the former case, we will eventually reset the input bits/cards to the initial state. Let I be positions of cards which represent bits $x_1 \cdots x_{n-s(n)\log^* n}$ at the beginning of the protocol. Thus, the cards on the positions I represent (in the 1-card encoding) the input for M. These cards will be in a read-only regime during the whole computation. Let J be the positions of cards representing $x_{n+1-s(n)\log^* n} \cdots x_n$. The cards on J will represent in 2-card encoding the content of tapes of M during the computation. Thus, $|J| = s(n)\log^* n$ but the cards on J will represent $\frac{1}{2}s(n)\log^* n$ bits. As $x_{n+1-s(n)\log^* n} \cdots x_n = 0101 \cdots 01$ (if $c = 0$), the cards on J represent $\frac{1}{2}s(n)\log^* n$ zeros at the beginning of the protocol.

The simulation proceeds in a similar way as the simulation in proof of Theorem 10. For reading bits from I that encode the input bits to M we use the same 1-card extension protocol as above to copy them into work-space. Now, we need procedures that will read and write bits in 2-card representations from positions J. However, if $c = 1$ some two consecutive positions would not represent a bit correctly (the two cards on them would have the same suit). The read/write procedures need to work and be secure even in this case. The players cannot simply inspect the cards on positions J because they may represent some intermediate results of the computation.

First, we describe how to read a bit b encoded in J. We want to create a 2-card representation of bit b in work space if $c = 0$ or a valid 2-card representation of some bit if $c = 1$. The bit b is represented by 2 cards α, β on positions in J. Note that α and β can be of the same suit if $c = 1$. Suppose Alice owns the positions in J representing the bit b, the case of Bob's positions is symmetric. First, Alice will add complementary cards to α and β as follows. Alice prepares the sequence: $\clubsuit\clubsuit\alpha\clubsuit\heartsuit\beta$, then she turns the second and fifth card face-down to get a sequence

$$\clubsuit ? \alpha \clubsuit ? \beta.$$

Bob shuffles the six cards cyclically at random. Now, Alice extends the card preceding each $\clubsuit$ that is face-up (cards α and β) by a complementary card to the right taken from an auxiliary deck. Alice sees the suit of the cards α and β during this action but she does not know their actual order. Bob shuffles the cards cyclically again and turns face-up the cards following the two $\clubsuit$ that are face-up. Now, by a cyclic shift, they rearrange the cards so that they look like

$$\clubsuit\clubsuit\alpha\overline{\alpha}\clubsuit\heartsuit\beta\overline{\beta},$$

where $\overline{\alpha}$ and $\overline{\beta}$ are cards complementary to α and β, respectively. They can copy each of the card pairs $\alpha, \overline{\alpha}$ and $\beta, \overline{\beta}$ to verify that Alice used complementary cards and the pairs $\alpha, \overline{\alpha}$ and $\beta, \overline{\beta}$ indeed represent two bits in 2-card encodings.

By following this protocol, Alice learns whether the two cards α, β have the same suit or are distinct. If the suits are the same she also learns the suit. However, in that situation $c = 1$ and she already knew all this information. In the later case, she knows that the bits at those input positions are distinct, but she knew that already. She does not learn their relative order because of the shuffle by Bob. Thus, she does not know whether they were altered since the beginning of the protocol or not.

To finish the read procedure, Alice copies the pair $\alpha, \overline{\alpha}$ (by the protocol of Theorem 3) to get two copies represented by cards $\alpha', \overline{\alpha'}, \alpha'', \overline{\alpha''}$. She returns the cards α'' and β back to the positions in J from which she moved the cards α and β at the beginning. The cards α' and $\overline{\alpha'}$ are used further in the computation. Other cards $(\overline{\alpha''}, \overline{\beta})$ are moved back to the auxiliary deck. If the cards α and β have different suits then the cards $\alpha', \overline{\alpha'}$ represent the bit b, as the card α and α' have the same suit. If the cards α and β have the same suit then the cards $\alpha', \overline{\alpha'}$ are a valid representation of some bit b'. However, in that case $c = 1$ and the value of b' is irrelevant for the computation. Bit values read from the storage, taking part in the bogus computation, will be consistent throughout the computation. Thus, the simulations of the NC^1 circuits implementing various steps of the computation are secure.

Now, we describe how to store a bit in 2-card encodings on to some positions in J. Again, suppose we want to store a bit b on to positions owned by Alice and occupied by cards γ_1 and γ_2. Let α and β be cards representing b. We want a procedure that will do the following. If $c = 0$ then the cards γ_1 and γ_2 are replaced by cards of the same suits as α and β, respectively. Otherwise, if $c = 1$ then the new cards need to have the same suits as γ_1 and γ_2. First, Alice will add complementary cards to γ_1 and γ_2 to get a sequence $\gamma_1, \overline{\gamma_1}, \gamma_2, \overline{\gamma_2}$ (she proceeds in the same was as in the read procedure above). Let d_1 and d_2 be bits represented by $\gamma_1, \overline{\gamma_1}$ and $\gamma_2, \overline{\gamma_2}$, respectively. She creates two copies of b, negates the second one, and computes $a_1 = SEL(c, b, d_1)$ and $a_2 = SEL(c, 1 - b, d_2)$. Let $\delta_1, \overline{\delta_1}$ and $\delta_2, \overline{\delta_2}$ be cards representing a_1 and a_2 respectively. If $c = 0$ then the cards δ_1, δ_2 represent the bit b. If $c = 1$ then the cards δ_1 and δ_2 have the same suits as the cards γ_1 and γ_2, respectively. Thus, Alice moves the cards δ_1 and δ_2 into the positions of the cards γ_1 and γ_2 and moves the rest of the cards to the deck.

To avoid leakage of information from the way Alice picks the cards from the auxiliary deck when picking a card of a particular suit, she proceeds as follows. She knows how many cards of that suit are in the deck. Thus, she shuffles the cards at random and then proceeds left to right to pick one of the cards of that suit uniformly at random. To achieve that she picks each card of the desired suit with probability $1/(k + 1)$, where k is the number of unseen cards of the desired suit still in the deck. This process guarantees that Alice will pick a card from a completely random position.

In this way the protocol can use $x_{n+1-s(n)\log^* n} \cdots x_n$ to compute $h = h(x)$. After obtaining value h it outputs $SEL(c, h, g)$. ◀

Hence, also in the case of the 1-card representation of the input one can take advantage of the input cards to compute functions that seem unattainable with read-only protocols.

5.2 1/2-Card Encoding

In the 1/2-card encoding we represent value 1 by either ♡× or ×♣, and value 0 by either ♣× or ×♡. Here × represents an empty bit position. To commit her input Alice picks $n/2$ of her input bits, and for those input bits, she leaves the empty spot × in the position of ♡, for the remaining bits she leaves the empty spot in place of ♣ (in the 2-card encoding of the bit). This way, she uses exactly $n/2$ cards ♣ and ♡ to commit her input. It is easy to verify that for each bit there is exactly 1/2 probability that the missing card will be on the left. Hence, the positions of the missing cards do not leak any information about her input. Bob proceeds in the same way to commit his input.

After committing their inputs they can run any read-only protocol similar to the case of 1-card encoding. Whenever an input bit is needed it is copied into a 2-card representation by essentially the same protocol as in the case of 1-card encoding.

This means that we need only $n + O(1)$ cards to compute any NC^1 function on n-bit inputs.

We do not know how to implement protocols which could modify their inputs. Modifying an input bit would require either picking the empty spot in the representation at random (which could lead to using substantially more cards of each type) or reusing the cards that are there. In the latter case we do not know how to do it without leaking information.

References

1 Yuta Abe, Yu-ichi Hayashi, Takaaki Mizuki, and Hideaki Sone. Five-card and protocol in committed format using only practical shuffles. In *Proceedings of the 5th ACM on ASIA Public-Key Cryptography Workshop*, APKC '18, page 3–8, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3197507.3197510.

2 David Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc1. *Journal of Computer and System Sciences*, 38:150–164, February 1989. doi:10.1016/0022-0000(89)90037-8.

3 Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: Catalytic space. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, page 857–866, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2591796.2591874.

4 R. E. Cleve. *Methodologies for Designing Block Ciphers and Cryptographic Protocols*. PhD thesis, University of Toronto, CAN, 1989.

5 Claude Crépeau and Joe Kilian. Discreet solitary games. In Douglas R. Stinson, editor, *Advances in Cryptology — CRYPTO' 93*, pages 319–330, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

6 Bert den Boer. More efficient match-making and satisfiability the five card trick. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology — EUROCRYPT '89*, pages 208–217, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.

7 Pavel Dvořák and Michal Koucký. Barrington plays cards: The complexity of card-based protocols, 2020. arXiv:2010.08445.

8 Danny Francis, Syarifah Ruqayyah Aljunid, Takuya Nishida, Yu-ichi Hayashi, Takaaki Mizuki, and Hideaki Sone. Necessary and sufficient numbers of cards for securely computing two-bit output functions. In Raphaël C.-W. Phan and Moti Yung, editors, *Paradigms in Cryptology – Mycrypt 2016. Malicious and Exploratory Cryptology*, pages 193–211, Cham, 2017. Springer International Publishing.

9 Julia Kastner, Alexander Koch, Stefan Walzer, Daiki Miyahara, Yu-ichi Hayashi, Takaaki Mizuki, and Hideaki Sone. The minimum number of cards in practical card-based protocols. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 126–155, Cham, 2017. Springer International Publishing.

10 Alexander Koch. The landscape of optimal card-based protocols. Cryptology ePrint Archive, Report 2018/951, 2018. urlhttps://eprint.iacr.org/2018/951.

11 Alexander Koch and Stefan Walzer. Foundations for actively secure card-based cryptography. In Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara, editors, *10th International Conference on Fun with Algorithms, FUN 2021, May 30 to June 1, 2021, Favignana Island, Sicily, Italy*, volume 157 of *LIPIcs*, pages 17:1–17:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.FUN.2021.17.

12 Alexander Koch, Stefan Walzer, and Kevin Härtel. Card-based cryptographic protocols using a minimal number of cards. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015*, pages 783–807, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

13 Takaaki Mizuki. Card-based protocols for securely computing the conjunction of multiple variables. *Theor. Comput. Sci.*, 622(C):34–44, April 2016. doi:10.1016/j.tcs.2016.01.039.

14 Takaaki Mizuki, Michihito Kumamoto, and Hideaki Sone. The five-card trick can be done with four cards. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, pages 598–606, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

15 Takaaki Mizuki and Hiroki Shizuya. A formalization of card-based cryptographic protocols via abstract machine. *Int. J. Inf. Secur.*, 13(1):15–23, February 2014. doi:10.1007/s10207-013-0219-4.

16 Takaaki Mizuki and Hiroki Shizuya. Practical card-based cryptography. In Alfredo Ferro, Fabrizio Luccio, and Peter Widmayer, editors, *Fun with Algorithms*, pages 313–324, Cham, 2014. Springer International Publishing.

17 Takaaki Mizuki and Hideaki Sone. Six-card secure and and four-card secure xor. In Xiaotie Deng, John E. Hopcroft, and Jinyun Xue, editors, *Frontiers in Algorithmics*, pages 358–369, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

18 Valtteri Niemi and Ari Renvall. Secure multiparty computations without computers. Technical report, Turku Centre for Computer Science, 1997.

19 Takuya Nishida, Yu-ichi Hayashi, Takaaki Mizuki, and Hideaki Sone. Card-based protocols for any boolean function. In Rahul Jain, Sanjay Jain, and Frank Stephan, editors, *Theory and Applications of Models of Computation*, pages 110–121, Cham, 2015. Springer International Publishing.

20 Akihiro Nishimura, Yu-ichi Hayashi, Takaaki Mizuki, and Hideaki Sone. An implementation of non-uniform shuffle for secure multi-party computation. In *Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography*, AsiaPKC '16, page 49–55, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2898420.2898425.

21 Anton Stiglic. Computations with a deck of cards. *Theor. Comput. Sci.*, 259(1):671–678, May 2001.

Round-Competitive Algorithms for Uncertainty Problems with Parallel Queries

Thomas Erlebach
School of Informatics, University of Leicester, UK

Michael Hoffmann
School of Informatics, University of Leicester, UK

Murilo Santos de Lima[1]
School of Informatics, University of Leicester, UK

Abstract

The area of computing with uncertainty considers problems where some information about the input elements is uncertain, but can be obtained using queries. For example, instead of the weight of an element, we may be given an interval that is guaranteed to contain the weight, and a query can be performed to reveal the weight. While previous work has considered models where queries are asked either sequentially (adaptive model) or all at once (non-adaptive model), and the goal is to minimize the number of queries that are needed to solve the given problem, we propose and study a new model where k queries can be made in parallel in each round, and the goal is to minimize the number of query rounds. We use competitive analysis and present upper and lower bounds on the number of query rounds required by any algorithm in comparison with the optimal number of query rounds. Given a set of uncertain elements and a family of m subsets of that set, we present an algorithm for determining the value of the minimum of each of the subsets that requires at most $(2+\varepsilon) \cdot \mathrm{opt}_k + \mathrm{O}\left(\frac{1}{\varepsilon} \cdot \lg m\right)$ rounds for every $0 < \varepsilon < 1$, where opt_k is the optimal number of rounds, as well as nearly matching lower bounds. For the problem of determining the i-th smallest value and identifying all elements with that value in a set of uncertain elements, we give a 2-round-competitive algorithm. We also show that the problem of sorting a family of sets of uncertain elements admits a 2-round-competitive algorithm and this is the best possible.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Discrete mathematics; Theory of computation → Theory and algorithms for application domains

Keywords and phrases online algorithms, competitive analysis, explorable uncertainty, parallel algorithms, minimum problem, selection problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.27

Related Version *Full Version*: `https://arxiv.org/abs/2101.05032`

Funding This research was supported by EPSRC grant EP/S033483/1.

Acknowledgements We would like to thank Markus Jablonka for helpful discussions.

1 Introduction

Motivated by real-world applications where only rough information about the input data is initially available but precise information can be obtained at a cost, researchers have considered a range of **uncertainty problems with queries** [7, 13, 14, 15, 16, 19, 26]. This research area has also been referred to as **queryable uncertainty** [12] or **explorable uncertainty** [17]. For example, in the input to a sorting problem, we may be given for each

[1] Corresponding author

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 27; pp. 27:1–27:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

input element, instead of its precise value, only an interval containing that point. Querying an element reveals its precise value. The goal is to make as few queries as possible until enough information has been obtained to solve the sorting problem, i.e., to determine a linear order of the input elements that is consistent with the linear order of the precise values. Motivation for explorable uncertainty comes from many different areas (see [12] and the references given there for further examples): The uncertain input elements may, e.g., be locations of mobile nodes or approximate statistics derived from a distributed database cache [29]. Exact information can be obtained at a cost, e.g., by requesting GPS coordinates from a mobile node, by querying the master database or by a distributed consensus algorithm.

The main model that has been studied in the explorable uncertainty setting is the **adaptive query model**: The algorithm makes queries one by one, and the results of previous queries can be taken into account when determining the next query. The number of queries made by the algorithm is then compared with the best possible number of queries for the given input (i.e., the minimum number of queries sufficient to solve the problem) using competitive analysis [5]. An algorithm is **ρ-query-competitive** (or simply ρ-competitive) if it makes at most ρ times as many queries as an optimal query set. A very successful algorithm design paradigm in this area is based on the concept of **witness sets** [7, 14]. A witness set is a set of input elements for which it is guaranteed that every query set that solves the problem contains at least one query in that set. If a problem admits witness sets of size at most ρ, one obtains a ρ-query-competitive algorithm by repeatedly finding a witness set and querying all its elements.

Some work has also considered the **non-adaptive query model** (see, e.g., [15, 28, 29]), where all queries are made simultaneously and the set of queries must be chosen in such a way that they certainly reveal sufficient information to solve the problem. In the non-adaptive query model, one is interested in complexity results and approximation algorithms.

In settings where the execution of a query takes a non-negligible amount of time and there are sufficient resources to execute a bounded number of queries simultaneously, the query process can be completed faster if queries are not executed one at a time, but in **rounds** with k simultaneous queries. Such scenarios include e.g. IoT environments (such as drones measuring geographic data), or teams of interviewers doing market research. Apart from being well motivated from an application point of view, this variation of the model is also theoretically interesting because it poses new challenges in selecting a useful set of k queries to be made simultaneously. Somewhat surprisingly, however, this has not been studied yet. In this paper, we address this gap and analyze for the first time a model where the algorithm can make up to k queries per round, for a given value k. The query results from previous rounds can be taken into account when determining the queries to be made in the next round. Instead of minimizing the total number of queries, we are interested in minimizing the number of query rounds, and we say that an algorithm is **ρ-round-competitive** if, for any input, it requires at most ρ times as many rounds as the optimal query set.

A main challenge in the setting with k queries per round is that the witness set paradigm alone is no longer sufficient for obtaining a good algorithm. For example, if a problem admits witness sets with at most 2 elements, this immediately implies a 2-query-competitive algorithm for the adaptive model, but only a k-round-competitive algorithm for the model with k queries per round. (The algorithm is obtained by simply querying one witness set in each round, and not making use of the other $k - 2$ available queries.) The issue is that, even if one can find a witness set of size at most ρ, the identity of subsequent witness sets may depend on the outcome of the queries for the first witness set, and hence we may not know how to compute a number of different witness sets that can fill a query round if $k \gg \rho$.

Our contribution. Apart from introducing the model of explorable uncertainty with k queries per round, we study several problems in this model: Minimum, Selection and Sorting. For Minimum (or Sorting), we assume that the input can be a family $\mathcal{S}$ of subsets of a given ground set $\mathcal{I}$ of uncertain elements, and that we want to determine the value of the minimum of (or sort) all those subsets. For Selection, we are given a set $\mathcal{I}$ of n uncertain elements and an index $i \in \{1, \ldots, n\}$, and we want to determine the i-th smallest value of the n precise values, and all the elements of $\mathcal{I}$ whose value is equal to that value.

Our main contribution lies in our results for the Minimum problem. We present an algorithm that requires at most $(2+\varepsilon) \cdot \mathrm{opt}_k + \mathrm{O}\left(\frac{1}{\varepsilon} \cdot \lg m\right)$ rounds, for every $0 < \varepsilon < 1$, where opt_k is the optimal number of rounds and $m = |\mathcal{S}|$. (The execution of the algorithm does not depend on ε, so the upper bound holds in particular for the best choice of $0 < \varepsilon < 1$ for given opt_k and m.) Interestingly, our algorithm follows a non-obvious approach that is reminiscent of primal-dual algorithms, but no linear programming formulation features in the analysis. For the case that the sets in $\mathcal{S}$ are disjoint, we obtain some improved bounds using a more straightforward algorithm. We also give lower bounds that apply even to the case of disjoint sets, and show that our upper bounds are close to best possible. Note that the Minimum problem is equivalent to the problem of determining the maximum element of each of the sets in $\mathcal{S}$, e.g., by simply negating all the numbers involved. A motivation for studying the Minimum problem thus arises from the minimum spanning tree problem with uncertain edge weights [11, 14, 17, 26]: Determining the maximum-weight edge of each cycle of a given graph allows one to determine a minimum spanning tree. Therefore, there is a connection between the problem of determining the maximum of each set in a family of possibly overlapping sets (which could be the edge sets of the cycles of a given graph) and the minimum spanning tree problem. The minimum spanning tree problem with uncertain edge weights has not been studied yet for the model with k queries per round, and seems to be difficult for that setting. In particular, it is not clear in advance for which cycles of the graph a maximum-weight edge actually needs to be determined, and this makes it very difficult to determine a set of k queries that are useful to be asked in parallel. We hope that our results for Minimum provide a first step towards solving the minimum spanning tree problem.

Another motivation for solving multiple possibly overlapping sets comes from distributed database caches [29], where one wants to answer database queries using cached local data and a minimum number of queries to the master database. Values in the local database cache may be uncertain, and exact values can be obtained by communicating with the central master database. Different database queries might ask for the record with minimum value in the field with uncertain information among a set of database records satisfying certain criteria, or for a list of such database records sorted by the field with uncertain information. Answering such database queries while making a minimum number of queries for exact values to the master database corresponds to the Minimum and Sorting problems we consider.

For the Selection problem, we obtain a 2-round-competitive algorithm. For Sorting, we show that there is a 2-round-competitive algorithm, by adapting ideas from a recent algorithm for sorting in the standard adaptive model [21], and that this is best possible.

We also discuss the relationship between our model and another model of parallel queries proposed by Meißner [27], and we give general reductions between both settings.

Literature overview. The seminal paper on minimizing the number of queries to solve a problem on uncertainty intervals is by Kahan [22]. Given n elements in uncertainty intervals, he presented optimal deterministic adaptive algorithms for finding the maximum, the median, the closest pair, and for sorting. Olston and Widom [29] proposed a distributed database system which exploits uncertainty intervals to improve performance. They gave non-adaptive algorithms for finding the maximum, the sum, the average and for counting problems. They also considered the case in which errors are allowed within a given bound, so a trade-off between performance and accuracy can be achieved. Khanna and Tan [23] extended this previous work by investigating adaptive algorithms for the situation in which bounded errors are allowed. They also considered the case in which query costs may be non-uniform, and presented results for the selection, sum and average problems, and for compositions of such functions. Feder *et al.* [16] studied the generalized median/selection problem, presenting optimal adaptive and non-adaptive algorithms. They proved that those are the best possible adaptive and non-adaptive algorithms, respectively, instead of evaluating them from a competitive analysis perspective. They also investigated the **price of obliviousness**, which is the ratio between the non-adaptive and adaptive strategies.

After this initial foundation, many classic discrete problems were studied in this framework, including geometric problems [7, 9], shortest paths [15], network verification [4], minimum spanning tree [11, 14, 17, 26], cheapest set and minimum matroid base [13, 28], linear programming [25, 30], traveling salesman [32], knapsack [19], and scheduling [2, 3, 10]. The concept of witness sets was proposed by Bruce *et al.* [7], and identified as a pattern in many algorithms by Erlebach and Hoffmann [12]. Gupta *et al.* [20] extended this framework to the setting where a query may return a refined interval, instead of the exact value of the element.

The problem of sorting uncertainty data has received some attention recently. Halldórsson and de Lima [21] presented better query-competitive algorithms, by using randomization or assumptions on the underlying graph structure. Other related work on sorting has considered sorting with noisy information [1, 6] or preprocessing the uncertain intervals so that the actual numbers can be sorted efficiently once their precise value are revealed [31].

The idea of performing multiple queries in parallel was also investigated by Meißner [27]. Her model is different, however. Each round/batch can query an unlimited number of intervals, but at most a fixed number of rounds can be performed. The goal is to minimize the total number of queries. Meißner gave results for selection, sorting and minimum spanning tree problems. We discuss this model in Section 6. A similar model was also studied by Canonne and Gur for property testing [8].

Organization of the paper. We present some definitions and preliminary results in Section 2. Sections 3, 4 and 5 are devoted to the sorting, minimum and selection problems, respectively. In Section 6, we discuss the relationship between the model we study and the model of Meißner for parallel queries [27]. We conclude in Section 7.

2 Preliminaries and Definitions

For the problems we consider, the input consists of a set of n continuous uncertainty intervals $\mathcal{I} = \{I_1, \ldots, I_n\}$ in the real line. The precise value of each data item is $v_i \in I_i$, which can be learnt by performing a query; formally, a query on I_i replaces this interval with $\{v_i\}$. We wish to solve the given problem by performing the minimum number of queries (or query rounds). We say that a closed interval $I_i = [\ell_i, u_i]$ is **trivial** if $\ell_i = u_i$; clearly $I_i = \{v_i\}$, so trivial intervals never need to be queried. Some problems require that intervals are either open or trivial; we will discuss this in further detail when addressing each problem. For a given realization $v_1, \ldots, v_n$ of the precise values, a set $Q \subseteq \mathcal{I}$ of intervals is a **feasible query set** if querying Q is enough to solve the given problem (i.e., to output a solution that can be proved correct based only on the given intervals and the answers to the queries in Q), and an **optimal query set** is a feasible query set of minimum size. Since the precise values are initially unknown to the algorithm and can be defined adversarially, we have an online exploration problem [5]. We fix an optimal query set OPT_1, and we write $\mathrm{opt}_1 := |\mathrm{OPT}_1|$. An algorithm which performs up to $\rho \cdot \mathrm{opt}_1$ queries is said to be **ρ-query-competitive**. Throughout this paper, we only consider deterministic algorithms.

In previous work on the adaptive model, it is assumed that queries are made sequentially, and the algorithm can take the results of all previous queries into account when deciding the next query. We consider a model where queries are made in **rounds** and we can perform up to k queries in parallel in each round. The algorithm can take into account the results from all queries made in previous rounds when deciding which queries to make in the next round. The adaptive model with sequential queries is the special case of our model with $k = 1$. We denote by opt_k the optimal number of rounds to solve the given instance. Note that $\mathrm{opt}_k = \lceil \mathrm{opt}_1/k \rceil$ as OPT_1 only depends on the input intervals and their precise values and can be distributed into rounds of k queries arbitrarily. For an algorithm ALG we denote by ALG_1 the number of queries it makes, and by ALG_k the number of rounds it uses. An algorithm which solves the problem in up to $\rho \cdot \mathrm{opt}_k$ rounds is said to be **ρ-round-competitive**. A query performed by an algorithm that is not in OPT_1 is called a **wasted** query, and we say that the algorithm **wastes** that query; a query performed by an algorithm that is not wasted is **useful**.

▶ **Proposition 2.1.** *If an algorithm makes all queries in* OPT_1*, wastes w queries in total over all rounds excluding the final round, always makes k queries per round except possibly in the final round, and stops as soon as the queries made so far suffice to solve the problem, then its number of rounds will be* $\lceil (\mathrm{opt}_1 + w)/k \rceil \leq \mathrm{opt}_k + \lceil w/k \rceil$.

The problems we consider are Minimum, Sorting and Selection. For Minimum and Sorting, we assume that we are given a set $\mathcal{I}$ of n intervals and a family $\mathcal{S}$ of m subsets of $\mathcal{I}$. For Sorting, the task is to output, for each set $S \in \mathcal{S}$, an ordering of the elements in S that is consistent with the order of their precise values. For Minimum, the task is to output, for each $S \in \mathcal{S}$, an element whose precise value is the minimum of the precise values of all elements in S, along with the value of that element.[2] Regarding the family $\mathcal{S}$, we can distinguish the cases where $\mathcal{S}$ contains a single set, where all sets in $\mathcal{S}$ are pairwise disjoint, and the case where the sets in $\mathcal{S}$ may overlap, i.e., may have common elements. For Selection, we are given a set $\mathcal{I}$ of n intervals and an index $i \in \{1, \dots, n\}$. The task is to output the i-th smallest value v^* (i.e., the value in position i in a sorted list of the precise values of the n intervals), as well as the set of intervals whose precise value equals v^*. We also discuss briefly a variant of Minimum in which we seek all elements whose precise value is the minimum and a variant of Selection in which we only seek the value v^*.

For a better understanding of the problems, we give a simple example for Sorting with $k = 1$. We have a single set with two intersecting intervals. There are four different configurations of the realizations of the precise values, which are shown in Figure 1. In Figure 1a, it is enough to query I_1 to learn that $v_1 < v_2$; however, if an algorithm first queries I_2, it cannot decide the order, so it must query I_1 as well. In Figure 1b we have a symmetric situation. In Figure 1c, both intervals must be queried (i.e., the only feasible query set is $\{I_1, I_2\}$), otherwise it is not possible to decide the order. Finally, in Figure 1d it is

[2] In some of the literature, it is only required to identify the element with minimum value. Returning the precise minimum value, however, is also an important problem, as discussed in [26, Section 7] for the minimum spanning tree problem.

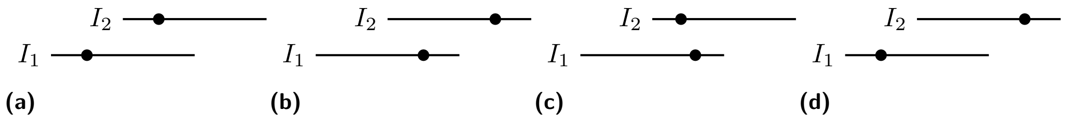


Figure 1 Example of Sorting for two intervals and the possible realizations of the precise values. We have that $\text{opt}_1 = 1$ in (a), (b) and (d), and $\text{opt}_1 = 2$ in (c).

enough to query either I_1 or I_2; hence, both $\{I_1\}$ and $\{I_2\}$ are feasible query sets. Since those realizations are initially identical to the algorithm, this example shows that no deterministic algorithm can be better than 2-query-competitive, and this example can be generalized by taking multiple copies of the given structure. For Minimum, however, an optimum solution can always be obtained by first querying I_1 (and then I_2 only if necessary): Since we need the precise value of the minimum element, in Figure 1b it is not enough to just query I_2.

3 Sorting

In this section we discuss the Sorting problem. We allow open, half-open, closed, and trivial intervals in the input, i.e., I_i can be of the form $[\ell_i, u_i]$ with $\ell_i \leq u_i$, or $(\ell_i, u_i]$, $[\ell_i, u_i)$ or (ℓ_i, u_i) with $\ell_i < u_i$.

First, we consider the case where $\mathcal{S}$ consists of a single set S, which we can assume to contain all n of the given intervals. We wish to find a permutation $\pi : [n] \to [n]$ such that $v_i \leq v_j$ if $\pi(i) < \pi(j)$, by performing the minimum number of queries possible. This problem was addressed for $k = 1$ in [21, 22, 27]; it admits 2-query-competitive deterministic algorithms and has a deterministic lower bound of 2.

For Sorting, if two intervals $I_i = [\ell_i, u_i]$ and $I_j = [\ell_j, u_j]$ are such that $I_i \cap I_j = \{u_i\} = \{\ell_j\}$, then we can put them in a valid order without any further queries, because clearly $v_i \leq v_j$. Therefore, we say that two intervals I_i and I_j **intersect** (or are **dependent**) if either their intersection contains more than one point, or if I_i is trivial and $v_i \in (\ell_j, u_j)$ (or *vice versa*). This is equivalent to saying that I_i and I_j are dependent if and only if $u_i > \ell_j$ and $u_j > \ell_i$. Two simple facts are important to notice, which are proven in [21]:

- For any pair of intersecting intervals, at least one of them must be queried in order to decide their relative order; i.e., any intersecting pair is a witness set.
- The **dependency graph** that represents this relation, with a vertex for each interval and an edge between intersecting intervals, is an interval graph [24].

We adapt the 2-query-competitive algorithm for Sorting by Halldórsson and de Lima [21] for $k = 1$ to the case of arbitrary k. Their algorithm first queries all non-trivial intervals in a minimum vertex cover in the dependency graph. By the duality between vertex covers and independent sets, the unqueried intervals form an independent set, so no query is necessary to decide the order between them. However, the algorithm still must query intervals in the independent set that intersect a trivial interval or the value of a queried interval. To adapt the algorithm to the case of arbitrary k, we first compute a minimum vertex cover and fill as many rounds as necessary with the given queries. After the answers to the queries are returned, we use as many rounds as necessary to query the intervals of the remaining independent set that contain a trivial point.

▶ **Theorem 3.1.** *The algorithm of Halldórsson and de Lima [21] yields a 2-round-competitive algorithm for Sorting that runs in polynomial time.*

Proof. Any feasible query set is a vertex cover in the dependency graph, due to the fact that at least one interval in each intersecting pair must be queried. Therefore a minimum vertex cover is at most the size of an optimal query set, so the first phase of the algorithm spends at most opt_k rounds. Since all intervals queried in the second phase are in any solution, again we spend at most another opt_k rounds. As the minimum vertex cover problem for interval graphs can be solved in polynomial time [18], the overall algorithm is polynomial as well. ◀

The problem has a lower bound of 2 on the round-competitive factor. This can be shown by having kc copies of a structure consisting of two dependent intervals, for some $c \geq 1$. OPT_1 may query only one interval in each pair, while we can force any deterministic algorithm to query both of them (cf. the configurations shown in Figures 1a and 1b). We have that $\mathrm{opt}_k = c$ while any deterministic algorithm will spend at least $2c$ rounds.

We remark that the 2-query-competitive algorithm for Sorting with $k = 1$ due to Meißner [27], when adapted to the setting with arbitrary k in the obvious way, only gives a bound of $2 \cdot \mathrm{opt}_k + 1$ rounds. Her algorithm first greedily computes a maximal matching in the dependency graph and queries all non-trivial matched vertices, and then all remaining intervals that contain a trivial point.

Now we study the case of solving a number of problems on different subsets of the same ground set of uncertain elements. In such a setting, it may be better to perform queries that can be reused by different problems, even if the optimum solution for one problem may not query that interval. We can reuse ideas from the algorithms for single problems that rely on the dependency graph. We define a new dependency relation (and dependency graph) in such a way that two intervals are dependent if and only if they intersect *and* belong to a common set. Note that the resulting graph may not be an interval graph, so some algorithms for single problems may not run in polynomial time for this generalization.

If we perform one query at a time ($k = 1$), then there are 2-competitive algorithms. One such is the algorithm by Meißner [27] described above; since a maximal matching can be computed greedily in polynomial time for arbitrary graphs, this algorithm runs in polynomial time for non-disjoint problems. If we can make $k \geq 2$ queries in parallel, then this algorithm performs at most $2 \cdot \mathrm{opt}_k + 1$ rounds, and the analysis is tight since we may have an incomplete round in between the two phases of the algorithm. If we relax the requirement that the algorithm runs in polynomial time, then we can obtain an algorithm that needs at most $2 \cdot \mathrm{opt}_k$ rounds, by first querying non-trivial intervals in a minimum vertex cover of the dependency graph (in as many rounds as necessary) and then the intervals that contain a trivial interval or the value of a queried interval (again, in as many rounds as necessary).

4 The Minimum Problem

For the Minimum problem, we assume without loss of generality that the intervals are sorted by non-decreasing left endpoints; intervals with the same left endpoint can be ordered arbitrarily. The **leftmost** interval among a subset of $\mathcal{I}$ is the one that comes earliest in this ordering. We also assume that all intervals are open or trivial; otherwise the problem has a trivial lower bound of n on the query-competitive ratio [20].

First, consider the case $\mathcal{S} = \{\mathcal{I}\}$, i.e., we have a single set. It is easy to see that the optimal query set consists of all intervals whose left endpoint is strictly smaller than the precise value of the minimum: If I_i with precise value v_i is a minimum element, then all other intervals with left endpoint strictly smaller than v_i must be queried to rule out that their value is smaller than v_i, and I_i must be queried (unless it is a trivial interval) to determine the value of the minimum. The optimal set of queries is hence a *prefix* of the sorted list of uncertain intervals (sorted by non-decreasing left endpoint). This shows that there is a 1-query-competitive algorithm when $k = 1$, and a 1-round-competitive algorithm for arbitrary k: In each round we simply query the next k uncertain intervals in the order of non-decreasing left endpoint, until the problem is solved. For $k = 1$, the same method yields a 1-query-competitive algorithm for the case with several sets: The algorithm can always query an interval with smallest left endpoint for any of the sets that have not yet been solved.[3]

In the remainder of this section, we consider the case of multiple sets and $k > 1$. We first present a more general result for potentially overlapping sets, then we give better upper bounds for disjoint sets. At the end of the section, we also present lower bounds.

Let $W(x) = x \lg x$; the inverse W^{-1} of W will show up in our analysis. Note that $W^{-1}(x) = \Theta(x/\lg x)$; this can be proved via implicit differentiation.

Throughout this section, we assume w.l.o.g. that the optimum must make at least one query in each set (or we consider only sets that require some query). We also assume that any algorithm always discards from each set all elements that are certainly not the minimum of that set, i.e., all elements for which it is already clear based on the available information that their value must be larger than the minimum value of the set (this is where the right endpoints of intervals also need to be considered). We adopt the following terminology. A set in $\mathcal{S}$ is **solved** if we can determine the value of its minimum element. A set is **active** at the start of a round if the queries made in previous rounds have not solved the set yet. An active set **survives** a round if it is still active at the start of the next round. An active set that does not survive the current round is said to be **solved in** the current round.

To illustrate these concepts, let us discuss a first simple strategy to build a query set Q for a round. Let $\mathcal{P}$ be the set of intervals queried in previous rounds. The **prefix length** of an active set S is the length of the maximum prefix of elements from Q in the list of non-trivial intervals in $S \setminus \mathcal{P}$ ordered by non-decreasing left endpoints. The algorithm proceeds by repeatedly adding to Q the leftmost non-trivial element not in $Q \cup \mathcal{P}$ from an arbitrary active set with minimum prefix length. We call this the **balanced** algorithm, and denote it by BAL. We give an example of its execution in Figure 2, with $m = 3$ disjoint sets and $k = 5$. The optimum solution queries the first three elements in S_1 and S_2, and all elements in S_3. Since the algorithm picks an arbitrary active set with minimum prefix length, it may give preference to S_1 and S_2 over S_3, thus wasting one query in S_1 and one in S_2 in round 2. All sets are active at the beginning of round 2; S_1 and S_2 are solved in round 2, while S_3 survives round 2. Since S_1 and S_2 are solved in round 2, they are no longer active in round 3, so the algorithm no longer queries any of their elements.

4.1 The Minimum Problem with Arbitrary Sets

We are given a set $\mathcal{I}$ of n intervals and a family $\mathcal{S}$ of m possibly overlapping subsets of $\mathcal{I}$, and a number $k \geq 2$ of queries that can be performed in each round.

Unfortunately, it is possible to construct an instance in which BAL uses as many as $k \cdot \mathrm{opt}_k$ rounds. In particular, it does not take into consideration that some elements are shared between different sets. The challenge is how to balance queries between sets in a better way.

[3] If we want to determine all elements whose value equals the minimum, it is not hard to see that the optimal set of queries for each set is again a prefix. As all our algorithms require only this property, we obtain corresponding results for that problem variant, even for inputs with arbitrary closed, open and half-open intervals.

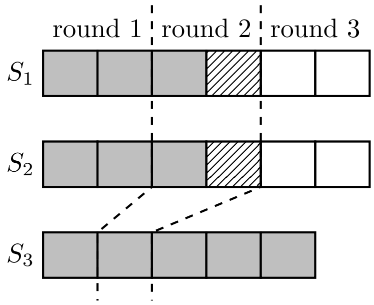


Figure 2 Possible execution of BAL for $m = 3$ disjoint sets and $k = 5$. Each interval is represented by a box, and the optimum solution is a prefix of each set. The solid boxes are useful queries, the two hatched boxes are wasted queries, and the white boxes are not queried by the algorithm.

Algorithm 1 Computing a query round for possibly non-disjoint sets.

```
Data: family $\mathcal{S} = \{S_1, \ldots, S_m\}$ of active subsets of the ground set $\mathcal{I}$
Result: set $Q \subseteq \mathcal{I}$ of at most $k$ queries to make
begin
    $Q \leftarrow$ set of leftmost unqueried elements of all sets in $\mathcal{S}$;
    if $|Q| \geq k$ then
        $Q \leftarrow$ arbitrary subset of $Q$ with size $k$;
    else
        $b_i \leftarrow 0$ for all $S_i \in \mathcal{S}$;
        while $|Q| < k$ and there are unqueried elements in $\mathcal{I} \setminus Q$ do
            foreach $e \in \mathcal{I} \setminus Q$ do
                $F_e \leftarrow \{i \mid e$ is the leftmost unqueried element from $\mathcal{I} \setminus Q$ in $S_i\}$;
            increase all $b_i$ simultaneously at the same rate until there is an unqueried
              element $e \in \mathcal{I} \setminus Q$ that satisfies $\sum_{i \in F_e} b_i = 1$;
            $Q \leftarrow Q \cup \{e\}$;
            $b_i \leftarrow 0$ for all $i \in F_e$;
    return $Q$;
```

We give an algorithm that requires at most $(2 + \varepsilon) \cdot \mathrm{opt}_k + \mathrm{O}\left(\frac{1}{\varepsilon} \cdot \lg m\right)$ rounds, for every $0 < \varepsilon < 1$. (The execution of the algorithm does not depend on ε, so the upper bound holds in particular for the best choice of $0 < \varepsilon < 1$ for given opt_k and m.) It is inspired by how some primal-dual algorithms work. The pseudocode for determining the queries to be made in a round is shown in Algorithm 1. First, we try to include the leftmost element of each set in the set of queries Q. If those are not enough to fill a round, then we maintain a variable b_i for each set S_i, which can be interpreted as a budget for each set. The variables are increased simultaneously at the same rate, until the sets that share a current leftmost unqueried element not in Q have enough budget to buy it. More precisely, at a given point of the execution, for each element $e \in \mathcal{I} \setminus Q$, let F_e contain the indices of the sets that have e as their leftmost unqueried element not in Q. We include e in Q when $\sum_{i \in F_e} b_i = 1$, and then we set b_i to zero for all $i \in F_e$. We repeat this process until $|Q| = k$ or there are no unqueried elements in $\mathcal{I} \setminus Q$.

When a query e is added to Q, we say that it is **charged** to the sets S_i with $i \in F_e$. The amount of charge for set S_i is equal to the value of b_i just before b_i is reset to 0 after adding e to Q. We also say that the set S_i **pays** this amount for e.

▶ **Definition 4.1.** *Let $\varepsilon > 0$. A round is* **ε-good** *if at least $k/2$ of the queries made by Algorithm 1 are also in OPT_1 (i.e., are useful queries), or if at least a/r active sets are solved in that round, where a is the number of active sets at the start of the round and $r = (2(1+\varepsilon) + \sqrt{2\varepsilon^2 + 4\varepsilon + 4})/\varepsilon$. A round that is not ε-good is called* **ε-bad***.*

Note that $r > 2$ for any $\varepsilon > 0$.

▶ **Lemma 4.2.** *If a round is ε-bad, then Algorithm 1 will make at least $2k/(2+\varepsilon)$ useful queries in the following round.*

Proof. Let a denote the number of active sets at the start of an ε-bad round. Let s be the number of sets that are solved in the current round; note that $s < a/r$ because the current round is ε-bad. Let T be the total amount by which each value b_i has increased during the execution of Algorithm 1. If the simultaneous increase of all b_i is interpreted as time passing, then T corresponds to the point in time when the computation of the set Q has been completed. For example, if some set S_i did not pay for any element during the whole execution, then T is equal to the value of b_i at the end of the execution of Algorithm 1.

Let Q be the set of queries that Algorithm 1 makes in the current round. We claim that every wasted query in Q is charged only to sets that are solved in this round. Consider a wasted query e that is in some set S_j not solved in this round. At the time e was selected, j cannot have been in F_e because otherwise e would be a useful query. Therefore, we do not charge e to S_j.

The total number of wasted queries is therefore bounded by Ts, as these queries are paid for by the s sets solved in this round. As the number of wasted queries in a bad round is larger than $k/2$, we therefore have $Ts > k/2$. As $s < a/r$, we get $k/2 < Ta/r$, so $T > (r/2) \cdot (k/a)$.

Call a surviving set S_i **rich** if $b_i > k/a$ when the computation of Q is completed. A set that is not rich is called **poor**. Note that a poor set must have spent at least an amount of $(r/2 - 1) \cdot (k/a) > 0$, as its total budget would be at least $T > (r/2) \cdot (k/a)$ if it had not paid for any queries. As the poor sets have paid for fewer than $k/2$ elements in total (as there are fewer than $k/2$ useful queries in the current round), the number of poor sets is bounded by $\frac{k/2}{(r/2-1)\cdot(k/a)} = a/(r-2) > 0$. As there are more than $(1 - 1/r) \cdot a$ surviving sets and at most $a/(r-2)$ of them are poor, there are at least $(1 - 1/r) \cdot a - a/(r-2) = ((r-2)(r-1) - r)/(r(r-2)) \cdot a = 2a/(2+\varepsilon) > 0$ surviving sets that are rich.

Let e be any element that is the leftmost unqueried element (at the end of the current round) of a rich surviving set. If e was the leftmost unqueried element of more than a/k rich surviving sets, those sets would have been able to pay for e (because their total remaining budget would be greater than $k/a \cdot a/k = 1$) before the end of the execution of Algorithm 1, a contradiction to e not being included in Q. Hence, the number of distinct leftmost unqueried elements of the at least $2a/(2+\varepsilon)$ rich surviving sets is at least $(2a/(2+\varepsilon))/(a/k) = 2k/(2+\varepsilon)$. So the following round will query at least $2k/(2+\varepsilon)$ elements that are the leftmost unqueried element of an active set, and all those are useful queries that are made in the next round. ◀

▶ **Theorem 4.3.** *Let opt_k denote the optimal number of rounds and A_k the number of rounds made if the queries are determined using Algorithm 1. Then, for every $0 < \varepsilon < 1$, $A_k \le (2+\varepsilon) \cdot \mathrm{opt}_k + \mathrm{O}\left(\frac{1}{\varepsilon} \cdot \lg m\right)$.*

Proof. In every round, one of the following must hold:

- The algorithm makes at least $k/2$ useful queries.
- The algorithm solves at least a fraction of $1/r$ of the active sets.
- If none of the above hold, the algorithm makes at least $2k/(2+\varepsilon)$ useful queries in the following round (by Lemma 4.2).

The number of rounds in which the algorithm solves at least a fraction of $1/r$ of the active sets is bounded by $\lceil \log_{r/(r-1)} m \rceil = \mathrm{O}\left(\frac{1}{\varepsilon} \cdot \lg m\right)$, since $1/\left(\lg \frac{r}{r-1}\right) < 5/\varepsilon$ for $0 < \varepsilon < 1$. In every round where the algorithm does not solve at least a fraction of $1/r$ of the active sets, the algorithm makes at least $k/(2+\varepsilon)$ useful queries on average (if in any such round it makes fewer than $k/2$ useful queries, it makes $2k/(2+\varepsilon)$ useful queries in the following round). The number of such rounds is therefore bounded by $(2+\varepsilon) \cdot \mathrm{opt}_k$. ◀

We do not know if this analysis is tight, so it would be worth investigating this question.

4.2 The Minimum Problem with Disjoint Sets

We now consider the case where $k \geq 2$ and the m sets in the given family $\mathcal{S}$ are pairwise disjoint. For this case, it turns out that the balanced algorithm achieves good upper bounds.

▶ **Theorem 4.4.** $\mathrm{BAL}_k \leq \mathrm{opt}_k + \mathrm{O}(\lg \min\{k, m\})$.

Proof. First we prove the bound for $m \leq k$. Index the sets in such a way that S_i is the i-th set that is solved by BAL, for $1 \leq i \leq m$. Sets that are solved in the same round are ordered by non-decreasing number of queries made in them in that round by BAL. In the round when S_i is solved, there are at least $m-(i-1)$ active sets, so the number of wasted queries in S_i is at most $\frac{k}{m-(i-1)}$. (BAL makes at most $\left\lceil \frac{k}{m-(i-1)} \right\rceil$ queries in S_i, and at least one of these is not wasted.) The total number of wasted queries is then at most $\sum_{i=1}^{m} \frac{k}{m-(i-1)} = \sum_{i=1}^{m} k/i = k \cdot H(m)$, where $H(m)$ denotes the m-th Harmonic number. By Proposition 2.1, $\mathrm{BAL}_k \leq \mathrm{opt}_k + \mathrm{O}(\lg m)$.

If $m > k$, observe that the algorithm does not waste any queries until the number of active sets is at most k. From that point on, it wastes at most $k \cdot H(k)$ queries following the arguments in the previous paragraph, so the number of rounds is bounded by $\mathrm{opt}_k + \mathrm{O}(\log k)$. ◀

We now give a more refined analysis that provides a better bound for $\mathrm{opt}_k = 1$, as well as a better multiplicative bound than what would follow from Theorem 4.4.

▶ **Lemma 4.5.** *If* $\mathrm{opt}_k = 1$, *then* $\mathrm{BAL}_k \leq \mathrm{O}(\lg m / \lg \lg m)$.

Proof. Consider an arbitrary instance of the problem with $\mathrm{opt}_k = 1$. Let $R+1$ be the number of rounds needed by the algorithm. For each of the first R rounds, we consider the fraction b_i of active sets that are not solved in that round. More formally, for the i-th round, for $1 \leq i \leq R$, if a_i denotes the number of active sets at the start of round i and a_{i+1} the number of active sets at the end of round i, then we define $b_i = a_{i+1}/a_i$.

Consider round i, $1 \leq i \leq R$. A set that is active at the start of round i and is still active at the start of the round $i+1$ is called a **surviving set**. A set that is active at the start of round i and gets solved by the queries made in round i is called a **solved set**. For each surviving set, all queries made in that set in round i are useful. For each solved set, at least one query made in that set is useful. We claim that this implies the algorithm makes at least kb_i useful queries in round i. To see this, observe that if the algorithm makes $\lfloor k/a_i \rfloor$ queries in a surviving set and $\lceil k/a_i \rceil$ queries in a solved set, we can conceptually move one useful query from the solved set to the surviving set. After this, the a_{i+1} surviving sets contain at least k/a_i useful queries on average, and hence $a_{i+1} \cdot k/a_i = b_i k$ useful queries in total.

As OPT_1 must make all useful queries and makes at most k queries in total, we have that $\sum_{i=1}^{R} kb_i \leq \mathrm{opt}_1 \leq k$, so $\sum_{i=1}^{R} b_i \leq 1$. Furthermore, as there are m active sets initially and there is still at least one active set after round R, we have that $\prod_{i=1}^{R} b_i = a_{R+1}/a_1 \geq 1/m$. To get an upper bound on R, we need to determine the largest possible value of R for which there exist values $b_i > 0$ for $1 \leq i \leq R$ satisfying $\sum_{i=1}^{R} b_i \leq 1$ and $\prod_{i=1}^{R} b_i \geq 1/m$. We gain nothing from choosing b_i with $\sum_{i=1}^{R} b_i < 1$, so we can assume $\sum_{i=1}^{R} b_i = 1$. In that case, the value of $\prod_{i=1}^{R} b_i$ is maximized if we set all b_i equal, namely $b_i = 1/R$. So we need to determine the largest value of R that satisfies $\prod_{i=1}^{R} 1/R \geq 1/m$, or equivalently $R^R \leq m$, or $R \lg R \leq \lg m$. This shows that $R \leq W^{-1}(\lg m) = \mathrm{O}(\lg m / \lg\lg m)$. ◀

▶ **Corollary 4.6.** *If* $\mathrm{opt}_k = 1$*, then* $\mathrm{BAL}_k \leq \mathrm{O}(\lg k / \lg\lg k)$*.*

Proof. If $k \geq m$, then the corollary follows from Lemma 4.5. If $k < m$, there can be at most k active sets, because the optimum performs at most k queries since $\mathrm{opt}_k = 1$. Hence, we only need to consider these k sets and can apply Lemma 4.5 with $m = k$. ◀

Now we wish to extend these bounds to arbitrary opt_k. It turns out that we can reduce the analysis for an instance with arbitrary opt_k to the analysis for an instance with $\mathrm{opt}_k = 1$, assuming that BAL is implemented in a round-robin fashion. A formal description of such an implementation is as follows: fix an arbitrary order of the m sets of the original problem instance as $S_1, S_2, \ldots, S_m$, and consider it as a cyclic order where the set after S_m is S_1. In each round, BAL distributes the k queries to the active sets as follows. Let i be the index of the set to which the last query was distributed in the previous round (or let $i = m$ if we are in the first round). Then initialize $Q = \emptyset$ and repeat the following step k times. Let j be the first index after i such that S_j is active and has unqueried non-trivial elements that are not in Q; pick the leftmost unqueried non-trivial element in $S_j \setminus Q$, insert it into Q, and set $i = j$. The resulting set Q is then queried. The proof of the following theorem is omitted.

▶ **Theorem 4.7.** BAL *is* $\mathrm{O}(\lg \min\{k, m\} / \lg\lg \min\{k, m\})$*-round-competitive.*

4.3 Lower Bounds

In this section we present lower bounds for Minimum that hold even for the more restricted case where the family $\mathcal{S}$ consists of disjoint sets.

▶ **Theorem 4.8.** *For arbitrarily large m and any deterministic algorithm* ALG*, there exists an instance with m sets and $k > m$ queries per round, such that* $\mathrm{opt}_k = 1$*,* $\mathrm{ALG}_k \geq W^{-1}(\lg m)$ *and* $\mathrm{ALG}_k = \Omega(W^{-1}(\lg k))$*. Hence, there is no* $\mathrm{o}(\lg \min\{k, m\} / \lg\lg \min\{k, m\})$*-round-competitive deterministic algorithm.*

Proof. Fix an arbitrarily large positive integer M. Consider an instance with $m = M^M$ sets, and let $k = M^{M+1}$. Each set contains Mk elements, with the i-th element having uncertainty interval $(1 + i\varepsilon, 100 + i\varepsilon)$ for $\varepsilon = 1/m$. The adversary will pick for each set an index j and set the j-th element to be the minimum, by letting it have value $1 + (j + 0.5)\varepsilon$, while the i-th element for $i \neq j$ is given value $100 + (i - 0.5)\varepsilon$. The optimal query set for the set is thus its first j elements. We assume that an algorithm queries the elements of each set in order of increasing lower interval endpoints. (Otherwise, the lower bound only becomes larger.)

Consider the start of a round when $a \leq m$ sets are still active; initially $a = m$. The adversary observes how the algorithm distributes its k queries among the active sets and repeatedly adds the active set with largest number of queries (from the current round) to a set $\mathcal{L}$, until the total number of queries from the current round in sets of $\mathcal{L}$ is at least $(M-1)k/M$. Let $\mathcal{S}'$ denote the remaining active sets. Note that $|\mathcal{S}'| \geq a/M$. For the sets in $\mathcal{L}$, the adversary chooses the minimum in such a way that a single query in the current round would have been sufficient to find it, while the sets in $\mathcal{S}'$ remain active (and so the optimum must make the same queries in them that the algorithm has made in the current round, and these are at most k/M queries). We continue for M rounds. In the M-th round, the adversary picks the minimum in all remaining sets in such way that a single query in that round would have been sufficient to solve the set. The optimal number of queries is then at most $(M-1)k/M + M^M = (M-1)k/M + k/M = k$, and hence $\mathrm{opt}_k = 1$. On the other hand, we have $\mathrm{ALG}_k = M$.

We can now express this lower bound in terms of k or m as follows: As $m = M^M$, we have $\lg m = M \lg M$ and hence $M = W^{-1}(\lg m)$. As $k = M^{M+1}$, we have $\lg k = (M+1)\lg M$ and hence $M = \Omega(W^{-1}(\lg k))$. Thus, the theorem follows. ◀

▶ **Theorem 4.9.** *No deterministic algorithm* ALG *attains* $\mathrm{ALG}_k \leq \mathrm{opt}_k + o(\lg \min\{k, m\})$.

Proof. Let $k = m$ be an arbitrarily large integer. The intervals of the m sets are chosen as in the proof of Theorem 4.8, for a sufficiently large value of M. Let a be the number of active sets at the start of a round; initially $a = m$. After each round, the adversary considers the set S_j in which the algorithm has made the largest number of queries, which must be at least k/a. The adversary picks the minimum element in S_j in such a way that a single query in the current round would have been enough to solve it, and keeps all other sets active. This continues for m rounds. The number of wasted queries is at least $k/m + k/(m-1) + \cdots + k/2 + k - m = k \cdot (H(m) - 1) = k \cdot \Omega(\lg k)$. As the algorithm must also make all queries in OPT_1, the theorem follows from Proposition 2.1. ◀

We conclude thus that the balanced algorithm attains matching upper bounds for disjoint sets. For non-disjoint sets, a small gap remains between our lower and upper bounds.

5 Selection

An instance of the Selection problem is given by a set $\mathcal{I}$ of n intervals and an integer i, $1 \leq i \leq n$. Throughout this section we denote the i-th smallest value in the set of n precise values by v^*.

If we only want to find the value v^*, then we can adapt the analysis in [20] to obtain an algorithm that performs at most $\mathrm{opt}_1 + i - 1$ queries provided all input intervals are open or trivial, simply by querying the intervals in the order of their left endpoints. This is the best possible and can easily be parallelized in $\mathrm{opt}_k + \lceil \frac{i-1}{k} \rceil$ rounds (we omit a proof). Thus we focus here on the task of finding v^* as well as identifying all intervals in $\mathcal{I}$ whose precise value equals v^*.

For ease of presentation, we assume that all the intervals in $\mathcal{I}$ are closed. The result can be generalized to arbitrary intervals without any significant new ideas, but the proofs become longer and require more cases, so we defer them to an extended version of the paper.

Let us begin by observing that the optimal query set is easy to characterize (proof omitted).

▶ **Lemma 5.1.** *Every feasible query set contains all non-trivial intervals that contain v^*. The optimal query set OPT_1 contains all non-trivial intervals that contain v^* and no other intervals.*

Let I_{j_1} be the interval with the i-th smallest left endpoint, and let I_{j_2} be the interval with the i-th smallest right endpoint. Then it is clear that v^* must lie in the interval $[\ell_{j_1}, u_{j_2}]$, which we call the **target area**. The following lemma was essentially shown by Kahan [22].

▶ **Lemma 5.2** (Kahan, 1991). *Assume that the current instance of* Selection *is not yet solved. Then there is at least one non-trivial interval I_j in $\mathcal{I}$ that contains the target area, i.e., satisfies $\ell_j \leq \ell_{j_1}$ and $u_j \geq u_{j_2}$.*

For $k = 1$, there is therefore an online algorithm that makes opt_1 queries: In each round, it determines the target area of the current instance and queries a non-trivial interval that contains the target area. (This algorithm was essentially proposed by Kahan [22] for determining all elements with value equal to v^*, without necessarily determining v^*.) For larger k, the difficulty is how to select additional intervals to query if there are fewer than k intervals that contain the target area.

The intervals that intersect the target area can be classified into four categories:

(1) a non-trivial intervals $[\ell_j, u_j]$ with $\ell_j \leq \ell_{j_1}$ and $u_j \geq u_{j_2}$; they **contain** the target area;

(2) b intervals $[\ell_j, u_j]$ with $\ell_j > \ell_{j_1}$ and $u_j < u_{j_2}$; they **are strictly contained** in the target area and contain neither endpoint of the target area;

(3) c intervals $[\ell_j, u_j]$ with $\ell_j \leq \ell_{j_1}$ and $u_j < u_{j_2}$; they intersect the target area on the **left**;

(4) d intervals $[\ell_j, u_j]$ with $\ell_j > \ell_{j_1}$ and $u_j \geq u_{j_2}$; they intersect the target area on the **right**.

We propose the following algorithm for rounds with k queries: Each round is filled with as many non-trivial intervals as possible, using the following order: first all intervals of category (1); then intervals of category (2); then picking intervals alternatingly from categories (3) and (4), starting with category (3). If one of the two categories (3) and (4) is exhausted, the rest of the k queries is chosen from the other category. Intervals of categories (3) and (4) are picked in order of non-increasing length of overlap with the target area, i.e., intervals of category (3) are chosen in non-increasing order of right endpoint, and intervals of category (4) in non-decreasing order of left endpoint. When a round is filled, it is queried, and the algorithm restarts, with a new target area and the intervals redistributed into the categories.

▶ **Proposition 5.3.** *At the start of any round, $a \geq 1$ and $b \leq a - 1$.*

Proof. Lemma 5.2 shows $a \geq 1$. If the target area is trivial, we have $b = 0$ and hence $b \leq a - 1$. From now on assume that the target area is non-trivial.

Let L be the set of intervals in $\mathcal{I}$ that lie to the left of the target area, i.e., intervals I_j with $u_j < \ell_{j_1}$. Similarly, let R be the set of intervals that lie to the right of the target area. Observe that $a + b + c + d + |L| + |R| = n$.

The intervals in L and the intervals of type (1) and (3) include all intervals with left endpoint at most ℓ_{j_1}. As ℓ_{j_1} is the i-th smallest left endpoint, we have $|L| + a + c \geq i$.

Similarly, the intervals in R and the intervals of type (1) and (4) include all intervals with right endpoint at least u_{j_2}. As u_{j_2} is the i-th smallest right endpoint, or equivalently the $(n - i + 1)$-th largest right endpoint, we have $|R| + a + d \geq n - i + 1$.

Adding the two inequalities derived in the previous two paragraphs, we get $2a + c + d + |L| + |R| \geq n + 1$. Combined with $a + b + c + d + |L| + |R| = n$, this yields $b \leq a - 1$. ◀

We omit the proof of the following lemma.

▶ **Lemma 5.4.** *If the current round of the algorithm is not the last one, then the following holds: If the algorithm queries at least one interval of categories (3) or (4), then the algorithm does not query all intervals of category (3) that contain v^*, or it does not query all intervals of category (4) that contain v^*.*

▶ **Theorem 5.5.** *There is a 2-round-competitive algorithm for* Selection*.*

Proof. Consider any round of the algorithm that is not the last one. Let A, B, C and D be the sets of intervals of categories (1), (2), (3) and (4) that are queried in this round, respectively. Let A^*, B^*, C^* and D^* be the subsets of A, B, C and D that are in OPT_1, respectively. By Lemmas 5.1 and 5.2, $|A| = |A^*| \geq 1$. Since the algorithm prioritizes category (1), by Proposition 5.3 we have $|B| \leq |A| - 1$, and thus $|A \cup B| \leq 2 \cdot |A| - 1 = 2 \cdot |A^*| - 1 \leq 2(|A^*| + |B^*|) - 1$.

For bounding the size of $C \cup D$, first note that the order in which the algorithm selects the elements of categories (3) and (4) ensures that, within each category, the intervals that contain v^* are selected first. By Lemma 5.4, there exists a category in which the algorithm does not query all intervals that contain v^* in the current round. If that category is (3), we have $|C| = |C^*|$ and, by the alternating choice of intervals from (3) and (4) starting with (3), $|D| \leq |C|$ and hence $|C \cup D| \leq 2 \cdot |C^*| \leq 2(|C^*| + |D^*|)$. If that category is (4), we have $|D| = |D^*|$ and $|C| \leq |D| + 1$, giving $|C \cup D| \leq 2 \cdot |D^*| + 1 \leq 2(|C^*| + |D^*|) + 1$. In both cases, we thus have $|C \cup D| \leq 2(|C^*| + |D^*|) + 1$.

Combining the bounds obtained in the two previous paragraphs, we get $|A \cup B \cup C \cup D| \leq 2(|A^*| + |B^*| + |C^*| + |D^*|)$. This shows that, among the queries made in the round, at most half are wasted. The total number of wasted queries in all rounds except the last one is hence bounded by opt_1. Since the algorithm fills each round except possibly the last one and also queries all intervals in OPT_1, the theorem follows by Proposition 2.1. ◀

We also have the following lower bound, which proves that our algorithm has the best possible multiplicative factor. We remark that it uses instances with $\mathrm{opt}_k = 1$, and we do not know how to scale it to larger values of opt_k. In its present form, it does not exclude the possibility of an algorithm using at most $\mathrm{opt}_k + 1$ rounds.

▶ **Lemma 5.6.** *There is a family of instances of* Selection *with $k = i \geq 2$ with $\mathrm{opt}_1 \leq i$ (and hence $\mathrm{opt}_k = 1$) such that any algorithm that makes k queries in the first round needs at least two rounds and performs at least $\mathrm{opt}_1 + \lceil (i-1)/2 \rceil$ queries.*

6 Relationship with the Parallel Model by Meißner

In [27, Section 4.5], Meißner describes a slightly different model for parallelization of queries. There, one is given a maximum number r of **batches** that can be performed, and there is no constraint on the number of queries that can be performed in a given batch. The goal is to minimize the total number of queries performed, and the algorithm is compared to an optimal query set. The number of uncertain elements in the input is denoted by n. In this section, we discuss the relationship between this model and the one we described in the previous sections.

Meißner argues that the sorting problem admits a 2-query-competitive algorithm for $r \geq 2$ batches. For the minimum problem with one set, she gives an algorithm which is $\lceil n^{1/r} \rceil$-query-competitive, with a matching lower bound. She also gives results for the selection and the minimum spanning tree problems.

▶ **Theorem 6.1.** *If there is an α-query-competitive algorithm that performs at most r batches, then there is an algorithm that performs at most $\alpha \cdot \mathrm{opt}_k + r - 1$ rounds of k queries. Conversely, if a problem has a lower bound of $\beta \cdot \mathrm{opt}_k + t$ on the number of rounds of k queries, then any algorithm running at most $t + 1$ batches has query-competitive ratio at least β.*

Proof. Given an α-query-competitive algorithm A on r batches, we construct an algorithm B for rounds of k queries in the following way. For each batch in A, algorithm B simply performs all queries in as many rounds as necessary. In between batches, we may have an incomplete round, but there are only $r - 1$ such rounds. ◀

In view of Meißner's lower bound for the minimum problem with one set mentioned above, the following result is close to being asymptotically optimal for that problem (using $\alpha = 1$). The proof of the following theorem is omitted due to space constraints.

▶ **Theorem 6.2.** *If there is an α-round-competitive algorithm for rounds of k queries, with α independent of k, then there is an algorithm that performs at most r batches with query-competitive ratio $\mathrm{O}(\alpha \cdot n^{\lfloor \alpha \rfloor/(r-1)})$, with $r \geq \lfloor \alpha \rfloor \cdot x + 1$ for an arbitrary natural number x. In particular, for $r \geq \lfloor \alpha \rfloor \cdot \lg n + 1$, the query-competitive factor is $\mathrm{O}(\alpha)$.*

Therefore, an algorithm that uses a constant number of batches implies an algorithm with the same asymptotic round-competitive ratio for rounds of k queries. On the other hand, some problems have worse query-competitive ratio if we require few batches, even if we have round-competitive algorithms for rounds of k queries, but the ratio is preserved by a constant if the number of batches is sufficiently large.

7 Final Remarks

We propose a model with parallel queries and the goal of minimizing the number of query rounds when solving uncertainty problems. Our results show that, even though the techniques developed for the sequential setting can be utilized in the new framework, they are not enough, and some problems are harder (have a higher lower bound on the competitive ratio).

One interesting open question is how to extend our algorithms for Minimum to the variant where it is not necessary to return the precise minimum value, but just to identify the minimum element. Another problem one could attack is the following generalization of Selection: Given multiple sets $S_1, \ldots, S_m \subseteq \mathcal{I}$ and indices $i_1, \ldots, i_m$, identify the i_j-smallest precise value and all elements with that value in S_j, for $j = 1, \ldots, m$. It would be interesting to see if the techniques we developed for Minimum with multiple sets can be adapted to Selection with multiple sets.

It would be nice to close the gaps in the round-competitive ratio, to understand if the analysis of Algorithm 1 is tight, and to study whether randomization can help to obtain better upper bounds. One could also study other problems in the parallel model, such as the minimum spanning tree problem.

References

1 M. Ajtai, V. Feldman, A. Hassidim, and J. Nelson. Sorting and selection with imprecise comparisons. *ACM Transactions on Algorithms*, 12(2):19:1–19:19, 2016. doi:10.1145/2701427.

2 S. Albers and A. Eckl. Explorable uncertainty in scheduling with non-uniform testing times. In *WAOA 2020: 18th International Workshop on Approximation and Online Algorithms*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2021. To appear. Also: arXiv preprint, arXiv:2009.13316, 2020.

3 L. Arantes, E. Bampis, A. V. Kononov, M. Letsios, G. Lucarelli, and P. Sens. Scheduling under uncertainty: A query-based approach. In *IJCAI 2018: 27th International Joint Conference on Artificial Intelligence*, pages 4646–4652, 2018. doi:10.24963/ijcai.2018/646.

4 Z. Beerliova, F. Eberhard, T. Erlebach, A. Hall, M. Hoffmann, M. Mihal'ák, and L. S. Ram. Network discovery and verification. *IEEE Journal on Selected Areas in Communications*, 24(12):2168–2181, 2006. doi:10.1109/JSAC.2006.884015.

5 A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

6 M. Braverman and E. Mossel. Sorting from noisy information. *arXiv preprint*, 2009. arXiv:0910.1191.

7 R. Bruce, M. Hoffmann, D. Krizanc, and R. Raman. Efficient update strategies for geometric computing with uncertainty. *Theory of Computing Systems*, 38(4):411–423, 2005. doi:10.1007/s00224-004-1180-4.

8 C. L. Canonne and T. Gur. An adaptivity hierarchy theorem for property testing. *computational complexity*, 27:671–716, 2018. doi:10.1007/s00037-018-0168-4.

9 G. Charalambous and M. Hoffmann. Verification problem of maximal points under uncertainty. In T. Lecroq and L. Mouchard, editors, *IWOCA 2013: 24th International Workshop on Combinatorial Algorithms*, volume 8288 of *Lecture Notes in Computer Science*, pages 94–105. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-45278-9_9.

10 C. Dürr, T. Erlebach, N. Megow, and J. Meißner. An adversarial model for scheduling with testing. *Algorithmica*, 2020. doi:10.1007/s00453-020-00742-2.

11 T. Erlebach and M. Hoffmann. Minimum spanning tree verification under uncertainty. In D. Kratsch and I. Todinca, editors, *WG 2014: International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 8747 of *Lecture Notes in Computer Science*, pages 164–175. Springer Berlin Heidelberg, 2014. doi:10.1007/978-3-319-12340-0_14.

12 T. Erlebach and M. Hoffmann. Query-competitive algorithms for computing with uncertainty. *Bulletin of the EATCS*, 116:22–39, 2015. URL: http://bulletin.eatcs.org/index.php/beatcs/article/view/335.

13 T. Erlebach, M. Hoffmann, and F. Kammer. Query-competitive algorithms for cheapest set problems under uncertainty. *Theoretical Computer Science*, 613:51–64, 2016. doi:10.1016/j.tcs.2015.11.025.

14 T. Erlebach, M. Hoffmann, D. Krizanc, M. Mihal'ák, and R. Raman. Computing minimum spanning trees with uncertainty. In S. Albers and P. Weil, editors, *STACS'08: 25th International Symposium on Theoretical Aspects of Computer Science*, volume 1 of *Leibniz International Proceedings in Informatics*, pages 277–288. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2008. doi:10.4230/LIPIcs.STACS.2008.1358.

15 T. Feder, R. Motwani, L. O'Callaghan, C. Olston, and R. Panigrahy. Computing shortest paths with uncertainty. *Journal of Algorithms*, 62(1):1–18, 2007. doi:10.1016/j.jalgor.2004.07.005.

16 T. Feder, R. Motwani, R. Panigrahy, C. Olston, and J. Widom. Computing the median with uncertainty. *SIAM Journal on Computing*, 32(2):538–547, 2003. doi:10.1137/S0097539701395668.

17 J. Focke, N. Megow, and J. Meißner. Minimum spanning tree under explorable uncertainty in theory and experiments. In C. S. Iliopoulos, S. P. Pissis, S. J. Puglisi, and R. Raman, editors, *SEA 2017: 16th International Symposium on Experimental Algorithms*, volume 75 of *Leibniz International Proceedings in Informatics*, pages 22:1–22:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.SEA.2017.22.

18 F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1(2):180–187, 1972. doi:10.1137/0201013.

19 M. Goerigk, M. Gupta, J. Ide, A. Schöbel, and S. Sen. The robust knapsack problem with queries. *Computers & Operations Research*, 55:12–22, 2015. doi:10.1016/j.cor.2014.09.010.

20 M. Gupta, Y. Sabharwal, and S. Sen. The update complexity of selection and related problems. *Theory of Computing Systems*, 59(1):112–132, 2016. doi:10.1007/s00224-015-9664-y.

21 M. M. Halldórsson and M. S. de Lima. Query-competitive sorting with uncertainty. In P. Rossmanith, P. Heggernes, and J.-P. Katoen, editors, *MFCS 2019: 44th International Symposium on Mathematical Foundations of Computer Science*, volume 138 of *Leibniz International Proceedings in Informatics*, pages 7:1–7:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.MFCS.2019.7.

22 S. Kahan. A model for data in motion. In *STOC'91: 23rd Annual ACM Symposium on Theory of Computing*, pages 265–277, 1991. doi:10.1145/103418.103449.

23 S. Khanna and W.-C. Tan. On computing functions with uncertainty. In *PODS'01: 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 171–182, 2001. doi:10.1145/375551.375577.

24 C. Lekkerkerker and J. Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51(1):45–64, 1962. URL: https://eudml.org/doc/213681.

25 T. Maehara and Y. Yamaguchi. Stochastic packing integer programs with few queries. *Mathematical Programming*, 182:141–174, 2020. doi:10.1007/s10107-019-01388-x.

26 N. Megow, J. Meißner, and M. Skutella. Randomization helps computing a minimum spanning tree under uncertainty. *SIAM Journal on Computing*, 46(4):1217–1240, 2017. doi:10.1137/16M1088375.

27 J. Meißner. *Uncertainty Exploration: Algorithms, Competitive Analysis, and Computational Experiments*. PhD thesis, Technische Universität Berlin, 2018. doi:10.14279/depositonce-7327.

28 A. I. Merino and J. A. Soto. The minimum cost query problem on matroids with uncertainty areas. In C. Baier, I. Chatzigiannakis, P. Flocchini, and S. Leonardi, editors, *ICALP 2019: 46th International Colloquium on Automata, Languages, and Programming*, volume 132 of *Leibniz International Proceedings in Informatics*, pages 83:1–83:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.83.

29 C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *VLDB 2000: 26th International Conference on Very Large Data Bases*, pages 144–155, 2000. URL: http://ilpubs.stanford.edu:8090/437/.

30 I. O. Ryzhov and W. B. Powell. Information collection for linear programs with uncertain objective coefficients. *SIAM Journal on Optimization*, 22(4):1344–1368, 2012. doi:10.1137/12086279X.

31 I. van der Hoog, I. Kostitsyna, M. Löffler, and B. Speckmann. Preprocessing ambiguous imprecise points. In G. Barequet and Y. Wang, editors, *SoCG 2019: 35th International Symposium on Computational Geometry*, volume 129 of *Leibniz International Proceedings in Informatics*, pages 42:1–42:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.SoCG.2019.42.

32 W. A. Welz. *Robot Tour Planning with High Determination Costs*. PhD thesis, Technische Universität Berlin, 2014. URL: https://www.depositonce.tu-berlin.de/handle/11303/4597.

Church Synthesis on Register Automata over Linearly Ordered Data Domains

Léo Exibard
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
Université libre de Bruxelles, Brussels, Belgium

Emmanuel Filiot
Université libre de Bruxelles, Brussels, Belgium

Ayrat Khalimov
Université libre de Bruxelles, Brussels, Belgium

Abstract

Register automata are finite automata equipped with a finite set of registers in which they can store data, i.e. elements from an unbounded or infinite alphabet. They provide a simple formalism to specify the behaviour of reactive systems operating over data ω-words. We study the synthesis problem for specifications given as register automata over a linearly ordered data domain (e.g. $(\mathbb{N}, \leq)$ or $(\mathbb{Q}, \leq)$), which allow for comparison of data with regards to the linear order. To that end, we extend the classical Church synthesis game to infinite alphabets: two players, Adam and Eve, alternately play some data, and Eve wins whenever their interaction complies with the specification, which is a language of ω-words over ordered data. Such games are however undecidable, even when the specification is recognised by a deterministic register automaton. This is in contrast with the equality case, where the problem is only undecidable for nondeterministic and universal specifications.

Thus, we study one-sided Church games, where Eve instead operates over a finite alphabet, while Adam still manipulates data. We show they are determined, and deciding the existence of a winning strategy is in ExpTime, both for $\mathbb{Q}$ and $\mathbb{N}$. This follows from a study of constraint sequences, which abstract the behaviour of register automata, and allow us to reduce Church games to ω-regular games. Lastly, we apply these results to the transducer synthesis problem for input-driven register automata, where each output data is restricted to be the content of some register, and show that if there exists an implementation, then there exists one which is a register transducer.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Automata over infinite objects; Theory of computation → Transducers

Keywords and phrases Synthesis, Church Game, Register Automata, Transducers, Ordered Data Words

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.28

Related Version *Full Version*: `https://arxiv.org/abs/2004.12141`

Funding This work was supported by the Fonds de la Recherche Scientifique - FNRS under Grant n°F.4510.9. Emmanuel Filiot is research associate of the Fonds de la Recherche Scientifique - FNRS.

1 Introduction

Synthesis is the problem of automatically constructing a system from a behavioral specification. It was first proposed by Church as a game problem: two players, Adam in the role of the environment and Eve in the role of the system, alternately pick the values from alphabets I and O. Adam starts with $i_0 \in I$, Eve responds with $o_0 \in O$, ad infinitum. Their interaction results in the infinite outcome $i_0 o_0 i_1 o_1 ... \in (I \cdot O)^\omega$. The winner is decided by a winning condition, represented as a language $S \subseteq (I \cdot O)^\omega$ called *specification*: if the outcome of Adam and Eve's interaction belongs to S, the play is won by Eve, otherwise by Adam. Eve

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 28; pp. 28:1–28:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

wins the game if she has a strategy $\lambda_E : I^+ \to O$ to pick values, depending on what has been played so far, allowing her to win against any Adam strategy. Similarly, Adam wins the game if he has a strategy $\lambda_A : O^* \to I$ to win against any Eve strategy. In the original Church problem, the alphabets I and O are finite, and specifications are ω-regular languages. The seminal papers [12, 33] connected Church games to zero-sum games on finite graphs. They also showed that Church games enjoy the property of *determinacy*: every game is either won by Eve or otherwise by Adam, and *finite-memoriness*: if Eve wins the game then she can win using a finite-memory strategy which can be executed by e.g. Mealy machines.

The synthesis and Church games were extensively studied in many settings, for example, quantitative, distributed, non-competitive, yet Adam and Eve usually interact via *finite* alphabets. But real-life systems often operate values from a large to *infinite* data domain. Examples include data-independent programs [39, 25, 32], software with integer parameters [10], communication protocols with message parameters [15], and more [9, 37, 14]. To address this challenge, recent works looked at synthesis where infinite-alphabet specifications are described by *register automata* and systems (corresponding to Eve strategies in Church games) by *register transducers* [16, 27, 28, 17].

Register automata extend finite-state automata to infinite alphabets $\mathcal{D}$ by introducing a finite number of *registers* [26]. In each step, the automaton reads a data from $\mathcal{D}$, compares it with the values held in its registers, depending on this comparison it decides to store the data into some of the registers, and then moves to a successor state. This way it builds a sequence of *configurations* (pairs of state and register values) representing its run on reading a word from $\mathcal{D}^\omega$: it is accepted if the visited states satisfy a certain condition, e.g. parity. Transducers are similar except that in each step they also output the content of one register.

Previous synthesis works [16, 27, 28, 17] focused on register automata and transducers operating in the domain $(\mathcal{D}, =)$ equipped with *equality* tests only. Related works [22, 31] on synthesis of data systems and which do not rely on register automata are also limited to equality tests or do not allow for data comparison. Thus, we cannot synthesise systems that output the largest value seen so far, grant a resource to a process with the lowest id, or raise an alert when a heart sensor reads values forming a dangerous curve. These tasks require $\leq$.

We study Church games where Adam and Eve have infinite alphabet $(\mathcal{D}, \leq)$, namely the dense domain $(\mathbb{Q}, \leq)$ or the nondense domain $(\mathbb{N}, \leq)$, and specifications are given as register automata. Already in the case of infinite alphabets $(\mathcal{D}, =)$, finding a winner is undecidable when specifications are given as nondeterministic or universal register automata [16, 17], so the works either restricted Eve strategies to register transducers with an a-priori fixed number of registers or considered specifications given as deterministic automata. The case of $(\mathbb{N}, \leq)$ is even harder. Here, Church games are undecidable already for specifications given as deterministic register automata, because they can simulate two-counter machines (Theorem 10). For example, to simulate an increment of a counter, whose value is currently kept in a register c, the automaton asks Adam to provide a data $\mathscr{d}$ above the value $\nu(c)$ of the counter, saves it into a register c_{new}, and asks Eve to provide the value between $\nu(c)$ and $\nu(c_{new})$. If Eve can do this, then Adam cheated and Eve wins, otherwise the game continues. Adam wins if eventually the halting state is reached. However, this proof breaks in the *asymmetric* setting, where Adam provides data but Eve picks labels from a finite alphabet only. We now give an example to better illustrate the one-sided setting.

▶ **Example.** Figure 1 illustrates a game arena where Adam's states are squares and Eve's states are circles. Eve's objective is to reach the top, while Adam tries to avoid it. There are two registers, r_M and r_l, and Eve's finite alphabet is $\{a, b\}$. The test $\top$ (true) means that the comparison of the input data with the register values is not important, the test $r_l < * < r_M$

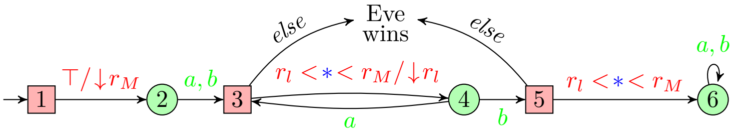


Figure 1 Eve wins this game in $\mathbb{N}$ but loses in $\mathbb{Q}$.

means that the data should be between the values of registers r_l and r_M, and the test "else" means the opposite. The writing ${\downarrow}r$ means that the data is stored into the register r. At first, Adam provides some data d_M, serving as an upper bound stored in r_M. Register r_l, initially 0, holds the last data d_l played by Adam. Consider state 3: if Adam provides a data outside of the interval $]d_l, d_M[$, he loses; if it is strictly between d_l and d_M, it is stored into register r_l and the game proceeds to state 4. There, Eve can either respond with label b and move to state 5, or with a to state 3. In state 5, Adam wins if he can provide a data strictly between d_l and d_M, otherwise he loses. Eve wins this game in $\mathbb{N}$: for example, she could always respond with label a, looping in states 3–4. After a finite number of steps, Adam is forced to provide a data $\geq d_M$, losing the game. An alternative Eve winning strategy, that does depend on Adam data, is to loop in 3–4 until $d_M - d_l = 1$ (hence she has to memorise the first Adam value d_M), then move to state 5, where Adam will lose. In the dense domain $\mathbb{Q}$, however, the game is won by Adam, because he can always provide a value within $]d_l, d_M[$ for any $d_l < d_M$, so the game either loops in 3–4 forever or reaches state 6. ⌟

Despite being asymmetric, one-sided Church games are quite expressive. For example, they enable synthesis of runtime data monitors that monitor the input data stream and raise a Boolean flag when a critical trend happens, like oscillations above a certain amplitude. Another example: they allow for synthesis of register transducers which can output data present in one of the registers of the specification automaton (also studied in [17]). Register-transducer synthesis serves as our main motivation for studying Church games.

The key idea used to solve problems about register automata is to forget the precise values of input data and registers, and track instead the constraints (also called types) describing the relations between them. In our example, all registers start in 0 so the initial constraint is $r_l^1 = r_M^1$, where r^i abstracts the value of register r at step i. Then, if Adam provides a data above the value of r_l, the constraint becomes $r_l^2 < r_M^2$ in state 2. Otherwise, if Adam had provided a data equal to the value in r_l, the constraint would be $r_l^2 = r_M^2$. In this way the constraints evolve during the play, forming an infinite sequence. Looping in states 3–4 induces the constraint sequence $\left(r_l^i < r_l^{i+1} < r_M^i = r_M^{i+1}\right)_{i>2}$. It forms an infinite chain $r_l^3 < r_l^4 < \dots$ bounded by constant $r_M^3 = r_M^4 = \dots$ from above. In $\mathbb{N}$, as it is a well-founded order, it is not possible to assign values to the registers at every step to satisfy all constraints, so the sequence is not satisfiable. Before elaborating on how this information can be used to solve Church games, we describe our results on satisfiability of constraint sequences. This topic was inspired by the work [35] which studies, among others, the nonemptiness problem of constraint automata, whose states and transitions are described by constraints. In particular, they show [35, Appendix C] that satisfiability of constraint sequences can be checked by *nondeterministic* ωB-automata [4]. Nondeterminism however poses a challenge in synthesis, and it is not known whether games with winning objectives as nondeterministic ωB-automata are decidable. In contrast, we describe a *deterministic* max-automaton [7] characterising the satisfiable constraint sequences in $\mathbb{N}$. As a consequence of [8], games over such automata are decidable. Then we study two kinds of constraint sequences inspired by Church games with register automata. First, we show that the satisfiable lasso-shaped (regular) constraint sequences, of the form uv^ω, are recognisable by deterministic *parity* automata. Second, we show how to assign values to registers on-the-fly in order to satisfy a constraint sequence induced by a play in the Church game.

To solve one-sided Church games with a specification given as a register automaton S for $(\mathbb{N}, \leq)$ and $(\mathbb{Q}, \leq)$, we reduce them to certain finite-arena zero-sum games, which we call feasibility games. The states and transitions of the game are those of the specification automaton S. The winning condition requires Eve to satisfy the original objective of S only on feasible plays, i.e. those that induce satisfiable constraint sequences. In our example, the play 1 2 $(3\ 4)^\omega$ does not satisfy the parity condition, yet it is won by Eve in the feasibility game since it is not satisfiable in $\mathbb{N}$, and therefore there is no corresponding play in the Church game. We show that if Eve wins the feasibility game, then she wins the Church game, using a strategy that simulates the register automaton S and simply picks one of its transitions. It is also sufficient: if Adam wins the feasibility game then he wins the Church game. To prove this, we construct, from an Adam strategy winning in the feasibility game, an Adam *data* strategy winning in the Church game. This step uses the previously mentioned results on satisfiability of constraint sequences of two special kinds. Overall, our results on one-sided Church games in $(\mathbb{N}, \leq)$ and $(\mathbb{Q}, \leq)$ are:

- they are decidable in time exponential in the number of registers of the specification,
- they are determined: every game is either won by Eve or by Adam, and
- if Eve wins, then she has a winning strategy that can be described by a register transducer with a finite number of states and which picks transitions in the specification automaton.

Finally, these results allow us to solve the register-transducer synthesis problem from input-driven output specifications [17] over ordered data.

Related works. [19] studies synthesis from variable automata with arithmetics (we only have $\leq$) which are incomparable with register automata; they only consider the dense domain. The paper [20] studies strategy synthesis but, again, mainly in the dense domain. A similar one-sided setting was studied in [21] for Church games with a winning condition given by logical formulas, but only for $(\mathcal{D}, =)$. The work on automata with atoms [30] implies our decidability result for $(\mathbb{Q}, \leq)$, even in the two-sided setting, but not the complexity result, and it does not apply to $(\mathbb{N}, \leq)$. Our setting in $\mathbb{N}$ is loosely related to monotonic games [2]: they both forbid infinite descending behaviours, but the direct conversion is unclear. Games on infinite arenas induced by pushdown automata [38, 11, 1] or one-counter systems [36, 23] are orthogonal to our games.

Outline. We start with Section 2 on satisfiability of constraint sequences, which is the main technical tool, then describe our results on Church games in Section 3 and synthesis in Sect.4.

2 Satisfiability of Constraint Sequences

In this paper, $\mathbb{N} = \{0, 1, \dots\}$. A *data domain* $\mathcal{D}$ is an infinite countable set of elements called *data*, linearly ordered by some order denoted $<$. We consider two data domains, $\mathbb{N}$ and $\mathbb{Q}$, with their usual order. We also distinguish a special element 0 of $\mathcal{D}$: in $\mathbb{Q}$ its choice is not important, in $\mathbb{N}$ it is the expected zero (the minimal element).

Registers and their valuations. Let R be a finite set of elements called *registers*, intended to contain data values, i.e. values in $\mathcal{D}$. A *register valuation* is a mapping $\nu : R \to \mathcal{D}$ (also written $\nu \in \mathcal{D}^R$). We write 0^R to denote the constant valuation $\nu_0(r) = 0$ for all $r \in R$.

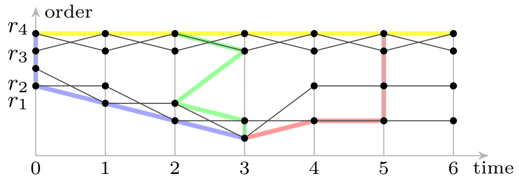


Figure 2 Visualisation of a constraint sequence. Individual register values are depicted by black dots, and dots are connected by black lines when they talk about the same register. Blue/red/green/yellow paths depict chains.

Constraint sequences, consistency and satisfiability. Fix a set of registers R (which can also be thought of as variables), and let $R' = \{r' \mid r \in R\}$ be the set of their *primed* versions. Fix a data domain $\mathcal{D}$. In what follows, the symbol $\bowtie$ denotes one of $>$, $<$, or $=$. A *constraint* is a maximal consistent set of atoms of the form $t_1 \bowtie t_2$ where $t_1, t_2 \in R \cup R'$. It describes how register values change in one step: their relative order at the beginning (when $t_1, t_2 \in R$), at the end (when $t_1, t_2 \in R'$), and between each other (with $t_1 \in R$ and $t_2 \in R'$). E.g., $C = \{r_1 < r_2, r_1 < r_1', r_2 > r_2', r_1' < r_2'\}$ is a constraint over $R = \{r_1, r_2\}$, which is satisfied, for instance, by the two successive valuations $\nu_a \colon \{r_1 \mapsto 1, r_2 \mapsto 4\}$ and $\nu_b \colon \{r_1 \mapsto 2, r_2 \mapsto 3\}$. However, the set $\{r_1 < r_2, r_1 > r_1', r_2 < r_2', r_1' > r_2'\}$ is not consistent.

Given a constraint C, the writing $C_{|R}$ denotes the subset of its atoms $r \bowtie s$ for $r, s \in R$, and $C_{|R'}$ – the subset of atoms over primed registers. Given a set S of atoms $r' \bowtie s'$ over $r', s' \in R'$, let $\mathit{unprime}(S)$ be the set of atoms derived by replacing every $r' \in R'$ by r.

A *constraint sequence* is an infinite sequence of constraints $C_0C_1\dots$ (when we use finite sequences, we explicitly state it). It is *consistent* if for every i: $\mathit{unprime}(C_{i|R'}) = C_{i+1|R}$, i.e. the register order at the end of step i equals the register order at the beginning of step $i+1$. Given a valuation $\nu \in \mathcal{D}^R$, define $\nu' \in \mathcal{D}^{R'}$ to be the valuation that maps $\nu'(r') = \nu(r)$ for every $r \in R$. A valuation $w \in \mathcal{D}^{R \cup R'}$ *satisfies* a constraint C, written $w \models C$, if every atom holds when we replace every $r \in R \cup R'$ by $w(r)$. A constraint sequence is *satisfiable* if there exists a sequence of valuations $\nu_0\nu_1\ldots \in (\mathcal{D}^R)^\omega$ such that $\nu_i \cup \nu_{i+1}' \models C_i$ for all $i \geq 0$. If, additionally, $\nu_0 = 0^R$, then it is 0-*satisfiable*. Notice that satisfiability implies consistency.

▶ **Examples.** Let $R = \{r_1, r_2, r_3, r_4\}$. Let a consistent constraint sequence $C_0C_1\dots$ start with

$$\big\{r_1 < r_2 < r_3 < r_4, r_4 = r_3', r_3 = r_4', r_1 = r_1', r_1 > r_2'\big\}\big\{r_2 < r_1 < r_4 < r_3, r_4 = r_3', r_3 = r_4', r_1 = r_1', r_2 > r_1'\big\}$$

Note that we omit some atoms in C_0 and C_1 for readability: although they are not maximal (e.g. C_0 does not contain $r_2' < r_1' < r_4' < r_3'$), they can be uniquely completed to maximal sets. Figure 2 (ignore the colored paths for now) visualises C_0C_1 plus a bit more constraints. The black lines represent the evolution of the same register. The constraint C_0 describes the transition from moment 0 to 1, and C_1—from 1 to 2. This finite constraint sequence is satisfiable in $\mathbb{Q}$ and in $\mathbb{N}$. For example, the valuations can start with $\nu_0 = \{r_4 \mapsto 6, r_3 \mapsto 5, r_2 \mapsto 4, r_1 \mapsto 3\}$. But no valuations starting with $\nu_0(r_3) < 5$ can satisfy the sequence in $\mathbb{N}$. Also, the constraint C_0 requires all registers in R to differ, hence the sequence is not 0-satisfiable in $\mathbb{Q}$ nor in $\mathbb{N}$. Another example is given by the sequence $(\{r > r'\})^\omega$ with $R = \{r\}$: it is satisfiable in $\mathbb{Q}$ but not in $\mathbb{N}$. ⌟

Satisfiability of constraint sequences in $\mathbb{Q}$. The following result is glimpsed in several places (e.g. in [35, Appendix C]): a constraint sequence is satisfiable in $\mathbb{Q}$ iff it is consistent. This is a consequence of the following property which holds because $\mathbb{Q}$ dense: for every constraint C and $\nu \in \mathbb{Q}^R$ such that $\nu \models C_{|R}$, there exists $\nu' \in \mathbb{Q}^{R'}$ such that $\nu \cup \nu' \models C$. Consistency can be checked by comparing every two consecutive constraints of the sequence. Thus it is not hard to show that consistent, hence satisfiable, constraint sequences in $\mathbb{Q}$ are recognizable by deterministic parity automata (see [18]).

▶ **Theorem 1.** *There is a deterministic parity automaton of size exponential in $|R|$ that accepts exactly all constraint sequences satisfiable in $\mathbb{Q}$. The same holds for* 0*-satisfiability.*

Satisfiability of constraint sequences in $\mathbb{N}$. Fix R and a constraint sequence $C_0C_1\dots$ over R. A (decreasing) *two-way chain* is a finite or infinite sequence $(r_0, m_0) \rhd_0 (r_1, m_1) \rhd_1 ... \in \big((R \times \mathbb{N}) \cdot \{=, >\}\big)^{*,\omega}$ satisfying the following (note that m_0 can differ from 0).

- $m_{i+1} = m_i$, or $m_{i+1} = m_i + 1$ (time flows forward), or $m_{i+1} = m_i - 1$ (time goes backwards).
- If $m_{i+1} = m_i$ then $(r_i \rhd_i r_{i+1}) \in C_{m_i}$.
- If $m_{i+1} = m_i + 1$ then $(r_i \rhd_i r'_{i+1}) \in C_{m_i}$.
- If $m_{i+1} = m_i - 1$ then $(r_{i+1} \rhd_i r'_i) \in C_{m_i - 1}$.

The *depth* of a chain is the number of $>$; when it is infinity, the chain is *infinitely* decreasing. Figure 2 shows four two-way chains: e.g., the green-colored chain $(r_4, 2) > (r_3, 3) > (r_2, 2) > (r_1, 3) > (r_2, 3)$ has depth 4. Similarly, we define *one-way* chains except that (a) they are either increasing (then $\rhd \in \{<, =\}$) or decreasing ($\rhd \in \{>, =\}$), and (b) time flows forward ($m_{i+1} = m_i + 1$) or stays ($m_{i+1} = m_i$). In Figure 2, the blue chain is one-way decreasing, the red chain is one-way increasing.

A *stable chain* is an infinite chain $(r_0, m) \rhd_0 (r_1, m+1) \rhd_1 (r_2, m+2) \rhd_2 ...$ with all $\rhd_i$ being the equality $=$; it can also be written as $(m, r_0r_1r_2...)$. Given a stable chain $\chi_r = (m, r_0r_1...)$ and a chain $\chi_s = (s_0, n_0) \bowtie_0 (s_1, n_1) \bowtie_1 ...$, such that $n_i \geq m$ for all plausible i, the chain χ_r is *non-strictly above* χ_s if for all n_i the constraint C_{n_i} contains $r_{n_i - m} > s_{n_i}$ or $r_{n_i - m} = s_{n_i}$. A stable chain $(m, r_0r_1...)$ is *maximal* if it is non-strictly above all other stable chains starting after m. In Figure 2, the yellow chain $(0, (r_4r_3)^\omega)$ is stable, non-strictly above all other chains, and maximal. A *trespassing chain* is a chain that is below a maximal stable chain.

▶ **Lemma 2.** *A consistent constraint sequence is satisfiable in $\mathbb{N}$ iff*
(A') *it has no infinite-depth two-way chains; and*
(B') $\exists \textsc{b} \in \mathbb{N}$*: all trespassing two-way chains have depth $\leq \textsc{b}$ (i.e. they have* bounded *depth).*

Proof idea. The left-to-right direction is trivial: if A' is not satisfied, then one needs infinitely many values below the maximal initial value of a register to satisfy the sequence, which is impossible in $\mathbb{N}$. Likewise, if B' is not satisfied, then one also needs infinitely many values below the value of a maximal stable chain, which is impossible. For the other direction, we show that if A and B hold, then one can construct a sequence of valuations $\nu_0\nu_1\dots$ satisfying the constraint sequence, such that for all $r \in R$, $\nu_i(r)$ is the largest depth of a (decreasing) two-way chain starting in r at moment i. The full proof is in [18]. ◀

The previous lemma characterises satisfiability in terms of two-way chains, but our final goal is recognise it with an automaton. It is hard to design a *one*-way automaton tracing *two*-way chains, so we use a Ramsey argument to lift the previous lemma to one-way chains.

▶ **Lemma 3.** *A consistent constraint sequence is satisfiable in $\mathbb{N}$ iff*
(A) *it has no infinitely decreasing one-way chains and*
(B) *the trespassing one-way chains have a bounded depth.*

Proof idea. We show that $A \wedge B$ implies $A' \wedge B'$ (the other direction is simple). Consider $\neg A' \Rightarrow \neg A$. From an infinite (decreasing) two-way chain, we can always extract an infinite decreasing one-way chain, since two-way chains are infinite to the right and not to the left.

Hence, for all moment i, there always exists a moment $j > i$ such that one register of the chain is smaller at step j than a register of the chain at step i. We also prove that $\neg B' \Rightarrow \neg B$. Given a sequence of trespassing two-way chains of unbounded depth, we are able to construct a sequence of one-way chains of unbounded depth. This construction is more difficult than in the case $\neg A' \Rightarrow \neg A$. Indeed, even though there are by hypothesis deeper and deeper trespassing two-way chains, they may start at later and later moments in the constraint sequence and go to the left, and so one cannot just take an arbitrarily deep two-way chain and extract from it an arbitrarily deep one-way chain. However, we show, using a Ramsey argument, that it is still possible to extract arbitrarily deep one-way chains as the two-way chains are not completely independent. The full proof is in [18]. ◀

The next lemma proved in [18] refines the characterisation to 0-satisfiability.

▶ **Lemma 4.** *A consistent constraint sequence is* 0*-satisfiable in* $\mathbb{N}$ *iff it satisfies conditions* $A \wedge B$ *from Lemma 3, starts in* C_0 *s.t.* $C_{0|R} = \{r = s \mid r, s \in R\}$*, and has no decreasing one-way chains of depth* ≥ 1 *from* $(r, 0)$ *for any* r*.*

We now state the main result about recognisability of satisfiable constraint sequences by *max-automata* [7]. These automata extend standard finite-alphabet automata with a finite set of counters $c_1, \ldots, c_n$ which can be incremented, reset to 0, or updated by taking the maximal value of two counters, but they cannot be tested. The acceptance condition is given as a Boolean combination of conditions "counter c_i is bounded along the run". Such a condition is satisfied by a run if there exists a bound $\mathrm{B} \in \mathbb{N}$ such that counter x_i has value at most B along the run. By using negation, conditions such as "x_i is unbounded along the run" can also be expressed. Deterministic max-automata are more expressive than ω-regular automata. For instance, they can express the non-ω-regular set of words $w = a^{n_1}ba^{n_2}b\ldots$ such that $n_i \leq \mathrm{B}$ for all $i \geq 0$, for some $\mathrm{B} \in \mathbb{N}$ that can vary from word to word.

▶ **Theorem 5.** *For every* R*, there is a deterministic max-automaton accepting exactly all constraint sequences satisfiable in* $\mathbb{N}$*. The number of states is exponential in* $|R|$*, and the number of counters is* $O(|R|^2)$*. The same holds for* 0*-satisfiability in* $\mathbb{N}$*.*

Proof idea. We design a deterministic max-automaton that checks conditions A and B of Lemma 3. Condition A, namely the absence of infinitely decreasing one-way chains, is checked as follows. We construct a nondeterministic Büchi automaton that guesses a chain and verifies that it is infinitely decreasing ("sees > infinitely often"). Determinising and complementing gives the sought deterministic parity automaton. Checking condition B (the absence of trespassing one-way chains of unbounded depth) is more involved. We design a master automaton that tracks every chain χ that currently exhibits a stable behaviour. To every such chain χ, the master automaton assigns a tracer automaton whose task is to ensure the absence of unbounded-depth trespassing chains below χ. For that, it uses $2|R|$ counters and requires them to be bounded. The overall acceptance condition ensures that if the chain χ is stable, then there are no trespassing chains below χ of unbounded depth. Since the master automaton tracks *every* such potential chain, we are done. Finally, we take a product of all these automata, which preserves determinism. (See [18].) ◀

▶ Remark. [35, Appendix C] shows that satisfiable constraint sequences in $\mathbb{N}$ are characterised by nondeterministic ωB-automata [4], which are strictly more expressive than max-automata.

The next results will come handy for game-related problems.

Lasso-shaped sequences (ω-regularity). An infinite sequence is *lasso-shaped* (or *regular*) if it is of the form $w = uv^\omega$. Notice that the number of constraints over a finite number of registers R is finite. Thus, using the standard pumping argument, one can show that in regular sequences an unbounded chain eventually loops (the proof is in [18]):

▶ **Lemma 6.** *For every lasso-shaped consistent constraint sequence, it has trespassing one-way chains of* unbounded *depth iff it has trespassing one-way chains of* infinite *depth.*

The above lemma together with Lemma 4 yields the following result:

▶ **Lemma 7.** *A lasso-shaped consistent constraint sequence is* 0*-satisfiable iff it has*

- *no infinite-depth decreasing one-way chains,*
- *no trespassing infinite-depth increasing one-way chains,*
- *no decreasing one-way chains of depth* ≥ 1 *from moment* 0*, and starts with* C_0 *s.t.* $C_{0|R} = \{r = s \mid r, s \in R\}$*.*

The conditions of this lemma can be checked by an ω-regular automaton:

▶ **Theorem 8.** *For every R, there is a deterministic parity automaton that accepts a lasso-shaped constraint sequence iff it is* 0*-satisfiable in* $\mathbb{N}$*; its number of states is exp. in* $|R|$*.*

Bounded sequences (data-assignment function). Fix a constraint sequence. Given a moment i and a register x, a *right two-way chain starting in* (x, i) *(r2w)* is a two-way chain $(x, i) \triangleright (r_1, m_1) \triangleright (r_2, m_2) \triangleright \dots$ such that $m_j \geq i$ for all plausible j. Note that r2w chains are *two*-way, meaning in particular that they can start and end in the same time moment i.

We design a data-assignment function that maps satisfiable constraint sequence prefixes to register valuations satisfying it. The function assumes that the r2w chains in the prefixes are bounded. It also assumes every constraint C_i in the sequence satisfies the following: for all $\nu \in \mathcal{D}^R, \nu' \in \mathcal{D}^{R'}$ s.t. $\nu \cup \nu' \models C_i$: $\big|\{r' \in R' \mid \forall s \in R.\, \nu'(r') \neq \nu(s)\}\big| \leq 1$ (*assumption* †). Intuitively: at most one new value can appear (but many disappear) during the step of the constraint (see also [18]). This assumption is used to simplify the proofs, yet it is satisfied by all constraint sequences induced by plays in Church games studied in the next section. A constraint sequence is *meaningful* if it is consistent, starts in C_0 with $C_{0|R} = \{r = s \mid r, s \in R\}$, and has no decreasing chains of depth ≥ 1 starting at moment 0.

▶ **Lemma 9** (data-assignment function). *For every* $\mathrm{B} \geq 0$*, there exists a data-assignment function* $f : (\mathsf{C}_{|R} \cup \mathsf{C}^+) \to \mathbb{N}^R$ *such that for every finite or infinite meaningful constraint sequence* $C_0 C_1 C_2 ...$ *satisfying assumption* † *and whose r2w chains are depth-bounded by* B*, the register valuations* $f(C_{0|R}) f(C_0) f(C_0 C_1) ...$ *satisfy the constraint sequence.*

Proof idea. We define a special kind of $xy^{(m)}$-chains that help to estimate how many insertions between the values of x and y at moment m we can expect in future. As it turns out, without knowing the future, the distance between x and y has to be exponential in the maximal depth of $xy^{(m)}$-chains. We describe a data-assignment function that maintains such exponential distances (the proof is by induction). The function is surprisingly simple: if the constraint inserts a register x between two registers r and s with already assigned values $\mathscr{d}_r$ and $\mathscr{d}_s$, then set $\mathscr{d}_x = \lfloor \frac{\mathscr{d}_r + \mathscr{d}_s}{2} \rfloor$; and if the constraint puts a register x above all other registers, then set $\mathscr{d}_x = \mathscr{d}_M + 2^{\mathrm{B}}$ where $\mathscr{d}_M$ the largest value currently held in the registers and B is the given bound on the depth of r2w chains. Full proof is in [18]. ◀

3 Church Synthesis Games

A *Church synthesis game* is a tuple $G = (I, O, S)$, where I is an *input* alphabet, O is an *output* alphabet, and $S \subseteq (I \cdot O)^\omega$ is a specification. Two players, Adam (the environment, who provides inputs) and Eve (the system, who controls outputs), interact. Their strategies are respectively represented as mappings $\lambda_A : O^* \to I$ and $\lambda_E : I^+ \to O$. Given λ_A and λ_E, the *outcome* $\lambda_A \| \lambda_E$ is the infinite sequence $i_0 o_0 i_1 o_1 ...$ such that for all $j \geq 0$: $i_j = \lambda_A(o_0 ... o_{j-1})$ and $o_j = \lambda_E(i_0 ... i_j)$. If $\lambda_A \| \lambda_E \in S$, the outcome is won by Eve, otherwise by Adam. Eve wins the game if she has a strategy λ_E such that for every Adam strategy λ_A, the outcome $\lambda_A \| \lambda_E$ is won by Eve. Solving a synthesis game amounts to finding whether Eve has a winning strategy. Synthesis games are parameterised by classes of alphabets and specifications. A game class is *determined* if every game in the class is either won by Eve or by Adam.

The class of synthesis games where I and O are finite and where S is an ω-regular language is known as *Church games*; they are decidable and determined. They also enjoy the finite-memoriness property: if Eve wins a game then there is an Eve winning strategy that can be represented as a finite-state machine.

We study synthesis games where the alphabets I and O are infinite and equipped with a linear order, and the specifications are described by deterministic register automata.

Register automata. Fix a set of registers R. A *test* is a maximally consistent set of atoms of the form $* \bowtie r$ for $r \in R$ and $\bowtie \in \{=, <, >\}$. We may represent tests as conjunctions of atoms instead of sets. The symbol "$*$" is used as a placeholder for incoming data. For example, for $R = \{r_1, r_2\}$, the expression $r_1 < *$ is not a test because it is not maximal, but $(r_1 < *) \wedge (* < r_2)$ is a test. We denote Tst_R the set of all tests and just Tst if R is clear from the context. A register valuation $\nu \in \mathcal{D}^R$ and data $d \in \mathcal{D}$ *satisfy* a test $\mathsf{tst} \in \mathsf{Tst}$, written $(\nu, d) \models \mathsf{tst}$, if all atoms of tst get satisfied when we replace the placeholder $*$ by d and every register $r \in R$ by $\nu(r)$. An *assignment* is a subset $\mathsf{asgn} \subseteq R$. Given an assignment asgn, a data $d \in \mathcal{D}$, and a valuation ν, we define $\mathit{update}(\nu, d, \mathsf{asgn})$ to be the valuation ν' s.t. $\forall r \in \mathsf{asgn} \colon \nu'(r) = d$ and $\forall r \notin \mathsf{asgn} \colon \nu'(r) = \nu(r)$.

A *deterministic register automaton* is a tuple $S = (Q, q_0, R, \delta, \alpha)$ where $Q = Q_A \uplus Q_E$ is a set of *states* partitioned into Adam and Eve states, the state $q_0 \in Q_A$ is *initial*, R is a set of *registers*, $\delta = \delta_A \uplus \delta_E$ is a (total and deterministic) *transition function* $\delta_P : (Q_P \times \mathsf{Tst} \to \mathsf{Asgn} \times Q_{P'})$ for $P \in \{A, E\}$ and the other player P', and $\alpha : Q \to \{1, ..., c\}$ is a *priority function* where c is the *priority index.*

A *configuration* of A is a pair $(q, \nu) \in Q \times \mathcal{D}^R$, describing the state and register content; the *initial configuration* is $(q_0, 0^R)$. A *run* of S on a word $w = d_0 d_1 ... \in \mathcal{D}^\omega$ is a sequence of configurations $\rho = (q_0, \nu_0)(q_1, \nu_1)...$ starting in the initial configuration and such that for every $i \geq 0$: by letting tst_i be a unique test for which $(\nu_i, d_i) \models \mathsf{tst}_i$, we have $\delta(q_i, \mathsf{tst}_i) = (\mathsf{asgn}_i, q_{i+1})$ for some asgn_i and $\nu_{i+1} = \mathit{update}(\nu_i, d_i, \mathsf{asgn}_i)$. Because the transition function δ is deterministic and total, every word induces a unique run in S. The run ρ is *accepting* if the maximal priority visited infinitely often is even. A word is *accepted* by S if it induces an accepting run. The *language* $L(S)$ of S is the set of all words it accepts.

Church games on register automata. If the data domain is $(\mathbb{N}, \leq)$, Church games are undecidable. Indeed, if the two players pick data values, it is easy to simulate a two-counter machine, where one player provides the values of the counters and the other verifies that no cheating happens on the increments and decrements, using the fact that $c' = c + 1$ whenever there does not exist d such that $c < d < c'$ (the formal proof can be found in [18]).

▶ **Theorem 10.** *Deciding the existence of a winning strategy for Eve in a Church game whose specification is a deterministic register automaton over* $(\mathbb{N}, \leq)$ *is undecidable.*

Church games on one-sided register automata. In light of this undecidability result, we consider one-sided synthesis games, where Adam provides data but Eve reacts with labels from a *finite* alphabet (a similar restriction was studied in [21] for domain $(\mathcal{D}, =)$). Specifications are now given as a language $S \subseteq (\mathcal{D} \cdot \Sigma)^\omega$. Such games are still quite expressive, as they enable the synthesis of "relaying" register transducers, which can only output data that is present in the specification automaton; we elaborate on this in Section 4.

A *one-sided register automaton* $S = (\Sigma, Q, q_0, R, \delta, \alpha)$ is a register automaton that additionally has a finite alphabet Σ of Eve *labels*, and its transition function $\delta = \delta_A \uplus \delta_E$ now has $\delta_E : Q_E \times \Sigma \to Q_A$ while $\delta_A : Q_A \times \mathsf{Tst} \to \mathsf{Asgn} \times Q_E$ stays as before. Runs on words in $(\mathcal{D} \cdot \Sigma)^\omega$ are defined as before except that register valuations are updated only in Adam states. We omit the formal definitions. Figure 1 shows an example of a one-sided automaton. For instance, it rejects the words $3a1b2(\Sigma\mathcal{D})^\omega$ and accepts the words $3a1a2b(\mathcal{D}\Sigma)^\omega$.

▶ **Theorem 11.** *For every Church game G on a one-sided automaton S over $\mathbb{N}$ or $\mathbb{Q}$:*

1. *Deciding if Eve wins G is doable in time polynomial in $|Q|$ and exponential in c and $|R|$.*
2. *The game is either won by Eve or otherwise by Adam.*

The proof of the theorem relies on the notion of action words. An *action word* is a sequence $(\mathsf{tst}_0, \mathsf{asgn}_0)(\mathsf{tst}_1, \mathsf{asgn}_1)...$ from $(\mathsf{Tst} \times \mathsf{Asgn})^{*,\omega}$. An action word is *$\mathcal{D}$-feasible* if there exists a sequence $\nu_0 d_0 \nu_1 d_1 \dots$ of register valuations ν_i and data d_i over $\mathcal{D}$ such that $\nu_0 = 0^R$ and for all plausible i: $\nu_{i+1} = \mathit{update}(\nu_i, d_i, \mathsf{asgn}_i)$ and $(\nu_i, d_i) \models \mathsf{tst}_i$. We first outline the proof structure and then provide the details.

Proof structure. We reduce the Church game G to a finite-arena game G_f called *feasibility game*. The states and transitions in G_f are those of S, and a play is winning if it either satisfies the parity condition of S or if the corresponding action word is not feasible.

In $\mathbb{Q}$, feasibility of action words can be checked by a deterministic parity automaton (Theorem 1). We then show that Eve wins the Church game G iff she wins the finite-arena game G_f. The direction $\Leftarrow$ is easy, because Eve winning strategy λ_E^f in G_f, which picks finite labels in Σ depending on the history of transitions of S, can be used to construct Eve winning strategy $\lambda_E : \mathbb{Q}^+ \to \Sigma$ in G by simulating the automaton S. To prove the other direction, we assume that Adam has a winning strategy λ_A^f in G_f, which picks tests depending on the history of transitions of S, then construct an Adam *data* strategy $\lambda_A : \Sigma^* \to \mathbb{Q}$ that concretises these tests into data values. This data instantiation is easy because $\mathbb{Q}$ is dense.

The case of $\mathbb{N}$ is treated similarly. However, checking feasibility of action words now requires a deterministic max-automaton (see page 7). From [8], we can deduce that games with a winning objective given as deterministic max-automata are decidable, yet the algorithm is involved, its complexity is high and does not yield finite-memory strategies that rely on picking transitions in S. Moreover, their determinacy is unknown. (For the same reasons we cannot rely on [6].) Therefore, we define quasi-feasible words, an ω-regular subset of feasible words sufficient for our purpose, and correspondingly define an ω-regular game G_f^{reg} by strengthening the winning condition of G_f. We then show that the Church game G and the finite-arena game G_f^{reg} are equi-realisable. The hard direction is again to prove that if Eve wins in G, then she wins in G_f^{reg}. As for $\mathbb{Q}$, assuming that Adam wins in G_f^{reg} with strategy λ_A^f, we construct Adam data strategy $\lambda_A : \Sigma^* \to \mathbb{N}$, relying on the finite-memoriness of the strategy λ_A^f and on the data-assignment function for constraint sequences from Lemma 9. ◀

▶ Remark 12. From the reduction of Church games to (quasi-)feasibility games, we get that if Eve wins a Church game G, then she has a winning strategy that simulates the run of the automaton S and simply picks its transitions. In this sense, Eve's strategy is "finite-memory" as it can be expressed by a register automaton with outputs with a finite number of states.

Games on finite arenas. A *two-player zero-sum finite-arena game* (or just finite-arena game) is a tuple $G = (V_\forall, V_\exists, v_0, E, W)$ where $V_\forall$ and $V_\exists$ are disjoint finite sets of *vertices* controlled by Adam and Eve, $v_0 \in V_\forall$ is *initial*, $E \subseteq (V_\forall \times V_\exists) \cup (V_\exists \times V_\forall)$ is a turn-based *transition relation*, and $W \subseteq (V_\forall \cup V_\exists)^\omega$ is a *winning objective.* An *Eve strategy* is a mapping $\lambda : (V_\forall \cdot V_\exists)^+ \to V_\forall$ such that $(v_\exists, \lambda(v_0 ... v_\exists)) \in E$ for all paths $v_0 ... v_\exists$ of G starting in v_0 and ending in $v_\exists \in V_\exists$. Adam strategies are defined similarly, by inverting the roles of $\exists$ and $\forall$. A *play* is a sequence of vertices starting in v_0 and satisfying the edge relation E. It is *won* by Eve if it belongs to W (otherwise it is won by Adam). An infinite play $\pi = v_0 v_1 \dots$ is *compatible* with an Eve strategy λ when for all $i \geq 0$ s.t. $v_i \in V_\exists$: $v_{i+1} = \lambda(v_0 \dots v_i)$. An Eve strategy is *winning* if all infinite plays compatible with it are winning.

It is well-known that parity games can be solved in n^c [24] (see also [13]), with n the size of the game and c the priority index.

Feasibility games. For the rest of this section, fix a one-sided register automaton $S = (\Sigma, Q, q_0, R, \delta, \alpha)$. With its Church game, we associate the following *feasibility game*, which is a finite-arena game $G_f = (V_\forall, V_\exists, v_0, E, W_f)$. Essentially, it memorises the transitions taken by the automaton S during the play of Adam and Eve. It has $V_\forall = \{q_0\} \cup (\Sigma \times Q_A)$, $V_\exists = \mathsf{Tst} \times \mathsf{Asgn} \times Q_E$, $v_0 = q_0$, $E = E_0 \cup E_\forall \cup E_\exists$ where:

- $E_0 = \big\{\big(v_0, (\mathsf{tst}, \mathsf{asgn}, u_0)\big) \mid \delta(v_0, \mathsf{tst}) = (\mathsf{asgn}, u_0)\big\}$,
- $E_\forall = \big\{\big((\sigma, v), (\mathsf{tst}, \mathsf{asgn}, u)\big) \mid \delta(v, \mathsf{tst}) = (\mathsf{asgn}, u)\big\}$, and
- $E_\exists = \big\{\big((\mathsf{tst}, \mathsf{asgn}, u), (\sigma, v)\big) \mid \delta(u, \sigma) = v\big\}$.

Let $\mathsf{Feasible}_{\mathcal{D}}(R)$ denote the set of action words over R feasible in $\mathcal{D}$. We let:

$$W_f = \big\{v_0(\mathsf{tst}_0, \mathsf{asgn}_0, u_0)(\sigma_0, v_1)\dots \mid (\mathsf{tst}_0 \mathsf{asgn}_0)\dots \in \mathsf{Feasible}_{\mathcal{D}}(R) \Rightarrow v_0 u_0 v_1 u_1 \dots \models \alpha\big\}$$

Later we will show that Eve wins the Church game G iff she wins the feasibility game G_f.

Action words and constraint sequences. A constraint C (cf Section 2) relates the values of the registers between the current moment and the next moment. A *state constraint* relates registers in the current moment only: it contains atoms over non-primed registers, so it has no atoms over primed registers. Note that both $C_{|R}$ and $\mathit{unprime}(C_{|R'})$ are state constraints.

Every action word naturally induces a unique constraint sequence. For instance, for registers $R = \{r, s\}$, an action word starting with $(\{r < *, s < *\}, \{s\})$ (test whether the current data d is above the values of r and s, store it in s) induces a constraint sequence starting with $\{r = s, r = r', s < s', r' < s'\}$ (the atom $r = s$ is due to all registers being equal initially). This is formalised in the next lemma, which is notation-heavy but says a simple thing: given an action word, we can construct, on the fly, a constraint sequence that is 0-satisfiable iff the action word is feasible. For technical reasons, we need a new register r_d to remember the last Adam data. The proof is direct and can be found in [18].

▶ **Lemma 13.** *Let R be a set of registers, $R_d = R \uplus \{r_d\}$, and $\mathcal{D} \in \{\mathbb{N}, \mathbb{Q}\}$. There exists a mapping $\mathit{constr} : \Pi \times \mathsf{Tst} \times \mathsf{Asgn} \to \mathsf{C}$ from state constraints Π over R_d and tests-assignments over R to constraints C over R_d, such that for all action words $a_0 a_1 a_2 ... \in (\mathsf{Tst} \times \mathsf{Asgn})^\omega$, $a_0 a_1 a_2 ...$ is feasible iff $C_0 C_1 C_2 ...$ is 0-satisfiable, where $\forall i \geq 0$: $C_i = \mathit{constr}(\pi_i, a_i)$, $\pi_{i+1} = \mathit{unprime}(C_{i|R'_d})$, $\pi_0 = \{r = s \mid r, s \in R_d\}$.*

Expressing the winning condition of G_f by deterministic automata. By converting an action word to a constraint sequence and then testing its satisfiability, we can test whether the action word is feasible. This allows us to express the winning condition W_f as a deterministic parity automaton for $\mathcal{D} = \mathbb{Q}$ and as a deterministic max-automaton for $\mathcal{D} = \mathbb{N}$. As a consequence of Theorem 1 (resp. 5), we get (see full proof in [18]):

▶ **Lemma 14.** *W_f is definable by a deterministic parity automaton if $\mathcal{D} = \mathbb{Q}$ and a deterministic max-automaton if $\mathcal{D} = \mathbb{N}$. Moreover, these automata are polynomial in $|Q|$ and exponential in $|R|$, and for $\mathcal{D} = \mathbb{Q}$, the index of the priority function is linear in c.*

Solving synthesis games on $(\mathbb{Q}, \leq)$

We outline the proof of Theorem 11 for $(\mathbb{Q}, \leq)$; the full proof can be found in [18].

The main goal is to show that Eve wins G iff she wins G_f. The direction $\Leftarrow$ is easy: Eve has less information in G_f, as she only has access to the tests satisfied by the input data, so she is stronger in G. Conversely, assume by contraposition that Eve does not win G_f. As ω-regular games are determined, Adam has a winning strategy λ_A^f in G_f. It induces a strategy λ_A for Adam in G: when the test is an equality, pick the corresponding data, and when it is of the form $r < * < r'$, take some rational number strictly in the interval. Then, each play consistent with this strategy in G corresponds to a unique run in S, which is also a play in G_f. As λ_A^f is winning, such run is accepting, so λ_A is winning: Eve does not win G.

Since the feasibility game G_f is of size polynomial in $|Q|$ and exponential in $|R|$, and has a number of priorities linear in c, we obtain item 1 of the theorem. Item 2 (determinacy) and Remark 12 are then a consequence of the finite-memory determinacy of ω-regular games.

Solving synthesis games on $(\mathbb{N}, \leq)$

We now outline the proof of Theorem 11 for $(\mathbb{N}, \leq)$; the full proof is in [18].

Using *ω-regular* game G_f^{reg} instead of G_f. W_f is not ω-regular, and the known results over deterministic max-automata do not suffice to obtain determinacy nor finite-memoriness, which will both prove useful for the transducer synthesis problem (cf Section 4).

We thus define an ω-regular subset $W_f^{reg} \subseteq W_f$ which is equi-realisable to W_f. Let $\mathsf{QFeasible}_{\mathbb{N}}(R)$ be the set of *quasi-feasible* action words over R, defined as the set of words $\overline{a}$ such that its induced constraint sequence (through the mapping *constr* of Lemma 13) starts with C_0, has no infinite-depth decreasing one-way chain nor trespassing increasing one-way chain, and no decreasing one-way chain of depth ≥ 1 from moment 0; by Lemma 7, this entails 0-satisfiability of lasso-shaped constraint sequences. We then let:

$$W_f^{reg} = \left\{ v_0(\mathsf{tst}_0, \mathsf{asgn}_0, u_0)(\sigma_0, v_1)\dots \mid (\mathsf{tst}_0, \mathsf{asgn}_0)\dots \in \mathsf{QFeasible}_{\mathbb{N}}(R) \Rightarrow v_0u_0v_1u_1\dots \models \alpha \right\}$$

From Lemma 8, we can build a deterministic parity automaton with a number of states exponential in $|R|$ and polynomial in $|Q|$ and a priority index linear in c recognising W_f^{reg}. Let G_f^{reg} be the finite-arena game with the same arena as G_f, with winning condition W_f^{reg}. We now show that the Church game G reduces to G_f^{reg} (full proof in [18]).

▶ **Proposition 15.** *Eve has a winning strategy in G iff she has a winning strategy in G_f^{reg}.*

Proof idea. If Eve has a winning strategy in G_f^{reg}, then, since $\mathsf{Feasible}_{\mathbb{N}}(R) \subseteq \mathsf{QFeasible}_{\mathbb{N}}(R)$, we have that $W_f^{reg} \subseteq W_f$, so it is also winning in G_f. Now, the argument for $\mathbb{Q}$ applies again for $\mathbb{N}$: as Eve has more information in G, if she wins in G_f, she wins in G.

The converse implication is harder; we show it by contraposition. Assume Eve does not have a winning strategy in G_f^{reg}. As ω-regular games are finite-memory determined, Adam has a finite-memory winning strategy λ_A^f in G_f^{reg}. It is not clear a priori that such strategy can be instantiated to a winning *data* strategy in G. However, we show that for finite-memory strategies, the depth of so-called right two-way chains is uniformly bounded, which by Lemma 9 allows us to instantiate the tests with concrete data:

▶ **Lemma 16.** *There is a number* $\text{B} \geq 0$ *that bounds the depths of all r2w chains coming from* λ_A^f*: for all constraint sequences resulting from playing with* λ_A^f*, for all* $x \in R$*, for all* $i \geq 0$*, we have that for all r2wch from* (x, i)*,* $depth(r2wch) \leq \text{B}$.

Proof idea of the lemma. Fix a moment i and a register x. After the moment i, only a bounded number of values can be inserted below the value of register x at moment i. Similarly, if we fix two registers at moment i, there can only be a bounded number of insertions between the values of x and y at moment i. Indeed, by finite-memoriness of Adam strategy, once the number of such insertions is larger than the memory of Adam, Eve can repeat her actions to force an infinite number of such insertions, leading to a play with an unfeasible action sequence and hence won by Eve. This intuition is captured by r2w chains defined in Section 2.

We prove the lemma by contradiction, by constructing a play consistent with λ_A^f which induces an unsatisfiable constraint sequence and therefore is losing for Adam. Assume that the constraint sequences induced by the plays with λ_A^f have unbounded-depth 2w chains. By Ramsey argument from Lemma 2, the constraint sequences have unbounded-depth *1w* chains. Along those chains, as λ_A^f is finite-memory, there is a repeating configuration with same constraints and states, and where the chain decrements or increments at least once and goes through the same registers. Thus, we can define a strategy λ_E^f of Eve which loops there forever. This induces an infinite chain. If it is decreasing, the corresponding play is not feasible, and is thus losing for Adam. If it is increasing, recall that this chain is actually a part of a r2w chain. By gluing them together, we get a r2w chain of infinite depth, which is not feasible either (recall that r2w chains start and end at the same point of time), so it is again losing for Adam. In both cases, this contradicts the assumption that λ_A^f is winning. ◀

Now, thanks to this uniform bound B and Lemma 9, we can construct $\lambda_A^{\mathbb{N}}$ from λ_A^f by translating the currently played action-word prefix $(\mathsf{tst}_0, \mathsf{asgn}_0)\ldots(\mathsf{tst}_m, \mathsf{asgn}_m)$ into a constraint-sequence prefix and applying the data-assignment function to it. By construction, for each play in G consistent with $\lambda_A^{\mathbb{N}}$, the corresponding run in S is a play consistent with λ_A^f in G_f^{reg}. As λ_A^f is winning, such run is not accepting, i.e. the play is winning for Adam in G. Therefore, $\lambda_A^{\mathbb{N}}$ is a winning Adam's strategy in G, meaning that Eve loses G. ◀

Since G_f^{reg} is of size polynomial in $|Q|$ and exponential in $|R|$, Theorem 11 follows.

4 Application to Transducer Synthesis

We now apply the above results to the transducer synthesis problem for specifications defined by input-driven register automata [17], i.e. two-sided automata where the output data is restricted to be the content of some register. Formal definitions of input-driven register automata and of register transducers are omitted as they are straightforward generalisations to the ordered case. Given a register automaton specification S, the transducer synthesis problem asks whether there exists a register transducer T such that $L(T) \subseteq L(S)$. A priori, T and S can have different sets of registers, but we show that it suffices to consider implementations that are subautomata of S, a result reminiscent of [17, Proposition 5]. Definitions and full proof of the theorem can be found in [18].

▶ **Theorem 17.** *For specifications defined by deterministic input-driven output register automata over data domains $\mathbb{Q}$ and $\mathbb{N}$, the register transducer synthesis problem can be solved in time polynomial in $|Q|$ and exponential in c and $|R|$.*

Proof idea. The transducer synthesis problem reduces to solving a one-sided Church game G. Indeed, output registers can be treated as finite labels, up to remembering equality constraints between registers in the states (this is exponential in $|R|$, but the exponentials do not stack). Moreover, we know by Proposition 15 that G itself reduces to G_f^{reg}. If Eve wins G_f^{reg}, she has a finite-memory winning strategy, which corresponds to a register transducer implementation of S which behaves like a subautomaton of S. ◀

5 Conclusion

In this paper, our main result states that 1-sided Church games for specifications given as *deterministic* register automata over $(\mathbb{N}, \leq)$ are decidable, in ExpTime. Moreover, we show that those games are determined. 1-sided Church games are motivated by register transducer synthesis, and the above result provides an ExpTime algorithm for this problem. As a future direction, it seems important to consider more expressive specification languages. Indeed, deterministic register automata are known to be strictly less expressive than nondeterministic or universal register automata. Such extensions are known to yield undecidability when used as specification formalisms in 1-sided Church games, already in the case of data equality only [17]. In [17, 29], a parameterized version of 1-sided Church games is shown to be decidable for universal register automata specifications. The parameter is a positive integer k and the goal is to decide whether there exists a strategy which can be implemented as a transducer with k registers. We plan to extend this result to linear orders. Universal register automata, thanks to their universal transitions, are better suited to specify properties of reactive systems. As an example, they can easily model properties such as "every request of client i is eventually granted", for every client id $i \in \mathbb{N}$. Such properties are not expressible by deterministic nor nondeterministic register automata. On the data part, while equality tests are sufficient for such properties, having a linear order could allow us to express more complex but natural properties, e.g. involving priorities between clients.

An important future direction is to consider logical formalisms instead of automata to describe specifications in a more declarative and high-level manner. Data-word first-order logics [5, 34] have been studied with respect to the satisfiability problem but when used as specification languages for synthesis, only few results are known. For slightly different contexts, see for example [3] for parameterized synthesis and [21] for games with temporal specifications and data.

References

1 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Piotr Hofman, Richard Mayr, K. Narayan Kumar, and Patrick Totzke. Infinite-state energy games. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 7:1–7:10, 2014.

2 Parosh Aziz Abdulla, Ahmed Bouajjani, and Julien d'Orso. Deciding monotonic games. In *International Workshop on Computer Science Logic*, pages 1–14. Springer, 2003.

3 Béatrice Bérard, Benedikt Bollig, Mathieu Lehaut, and Nathalie Sznajder. Parameterized synthesis for fragments of first-order logic over data words. In *FOSSACS*, volume 12077 of *Lecture Notes in Computer Science*, pages 97–118. Springer, 2020.

4 M. Bojańczyk and T. Colcombet. Bounds in ω-regularity. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 285–296, 2006.

5 M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 7–16, 2006.

6 Mikołaj Bojańczyk. A bounding quantifier. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *Computer Science Logic*, pages 41–55, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

7 Mikołaj Bojańczyk. Weak MSO with the unbounding quantifier. *Theory of Computing Systems*, 48(3):554–576, 2011.

8 Mikołaj Bojańczyk. Weak MSO+U with path quantifiers over infinite trees. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, pages 38–49, 2014.

9 A. Bouajjani, P. Habermehl, Y. Jurski, and M. Sighireanu. Rewriting systems with data. In *FCT*, pages 1–22, 2007.

10 A. Bouajjani, P. Habermehl, and R R. Mayr. Automatic verification of recursive procedures with one integer parameter. *Theoretical Computer Science*, 295:85–106, 2003.

11 A.-J. Bouquet, O. Serre, and I. Walukiewicz. Pushdown games with unboundedness and regular conditions. In *Proc. 23rd Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 2914 of *Lecture Notes in Computer Science*, pages 88–99. Springer, 2003.

12 J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. AMS*, 138:295–311, 1969.

13 C.S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding parity games in quasipolynomial time. In *Proc. 49th ACM Symp. on Theory of Computing*, pages 252–263, 2017.

14 S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing Data-Intensive Web Applications.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

15 G. Delzanno, A. Sangnier, and R. Traverso. Parameterized verification of broadcast networks of register automata. In P. A. Abdulla and I. Potapov, editors, *Reachability Problems*, pages 109–121, Berlin, Heidelberg, 2013. Springer.

16 R. Ehlers, S. Seshia, and H. Kress-Gazit. Synthesis with identifiers. In *Proc. 15th Int. Conf. on Verification, Model Checking, and Abstract Interpretation*, volume 8318 of *Lecture Notes in Computer Science*, pages 415–433. Springer, 2014.

17 L. Exibard, E. Filiot, and P-A. Reynier. Synthesis of data word transducers. In *Proc. 30th Int. Conf. on Concurrency Theory*, 2019.

18 Léo Exibard, Emmanuel Filiot, and Ayrat Khalimov. Church synthesis on register automata over infinite ordered alphabets, 2020. `arXiv:2004.12141`.

19 Rachel Faran and Orna Kupferman. On synthesis of specifications with arithmetic. In Alexander Chatzigeorgiou, Riccardo Dondi, Herodotos Herodotou, Christos Kapoutsis, Yannis Manolopoulos, George A. Papadopoulos, and Florian Sikora, editors, *SOFSEM 2020: Theory and Practice of Computer Science*, pages 161–173, Cham, 2020. Springer International Publishing.

20 Azadeh Farzan and Zachary Kincaid. Strategy synthesis for linear arithmetic games. *Proceedings of the ACM on Programming Languages*, 2(POPL):1–30, 2017.

21 Diego Figueira, Anirban Majumdar, and M. Praveen. Playing with repetitions in data words using energy games. *Log. Methods Comput. Sci.*, 16(3), 2020. URL: `https://lmcs.episciences.org/6614`.

22 B. Finkbeiner, F. Klein, R. Piskac, and M. Santolucito. Temporal stream logic: Synthesis beyond the bools. In *Proc. 31st Int. Conf. on Computer Aided Verification*, 2019.

23 Stefan Göller, Richard Mayr, and Anthony Widjaja To. On the computational complexity of verifying one-counter processes. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, pages 235–244, 2009.

24 E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

25 R. Hojati, D.L. Dill, and R.K. Brayton. Verifying linear temporal properties of data insensitive controllers using finite instantiations. In *Hardware Description Languages and their Applications*, pages 60–73. Springer, 1997.

26 M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.

27 A. Khalimov, B. Maderbacher, and R. Bloem. Bounded synthesis of register transducers. In *16th Int. Symp. on Automated Technology for Verification and Analysis*, volume 11138 of *Lecture Notes in Computer Science*, pages 494–510. Springer, 2018.

28 Ayrat Khalimov and Orna Kupferman. Register-bounded synthesis. In *30th International Conference on Concurrency Theory (CONCUR 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

29 Ayrat Khalimov and Orna Kupferman. Register-bounded synthesis. In Wan Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 25:1–25:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CONCUR.2019.25.

30 Bartek Klin and Mateusz Łełyk. Scalar and Vectorial mu-calculus with Atoms. *Logical Methods in Computer Science*, Volume 15, Issue 4, October 2019. doi:10.23638/LMCS-15(4:5)2019.

31 Paul Krogmeier, Umang Mathur, Adithya Murali, P. Madhusudan, and Mahesh Viswanathan. Decidable synthesis of programs with uninterpreted functions. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification*, pages 634–657, Cham, 2020. Springer International Publishing.

32 R. Lazić and D. Nowak. A unifying approach to data-independence. In *Proc. 11th Int. Conf. on Concurrency Theory*, pages 581–596. Springer Berlin Heidelberg, 2000.

33 M.O. Rabin. Automata on infinite objects and Church's problem. *Amer. Mathematical Society*, 1972.

34 Thomas Schwentick and Thomas Zeume. Two-variable logic with two order relations. *Log. Methods Comput. Sci.*, 8(1), 2012.

35 Luc Segoufin and Szymon Torunczyk. Automata-based verification over linearly ordered data domains. In *28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2011.

36 Olivier Serre. Parity games played on transition graphs of one-counter processes. In *Foundations of Software Science and Computation Structures, 9th International Conference, FOSSACS 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25-31, 2006, Proceedings*, pages 337–351, 2006.

37 V. Vianu. Automatic verification of database-driven systems: a new frontier. In *ICDT '09*, pages 1–13, 2009.

38 I. Walukiewicz. Model checking CTL properties of pushdown systems. In *Proc. 20th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 1974 of *Lecture Notes in Computer Science*, pages 127–138. Springer, 2000.

39 P. Wolper. Expressing interesting properties of programs in propositional temporal logic. In *Proc. 13th ACM Symp. on Principles of Programming Languages*, pages 184–192, 1986.

A Faster Algorithm for Finding Tarski Fixed Points

John Fearnley
Department of Computer Science, University of Liverpool, UK

Rahul Savani
Department of Computer Science, University of Liverpool, UK

Abstract

Dang et al. have given an algorithm that can find a Tarski fixed point in a k-dimensional lattice of width n using $O(\log^k n)$ queries [2]. Multiple authors have conjectured that this algorithm is optimal [2, 7], and indeed this has been proven for two-dimensional instances [7]. We show that these conjectures are false in dimension three or higher by giving an $O(\log^2 n)$ query algorithm for the three-dimensional Tarski problem, which generalises to give an $O(\log^{k-1} n)$ query algorithm for the k-dimensional problem when $k \geq 3$.

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

Keywords and phrases query complexity, Tarski fixed points, total function problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.29

Related Version *Full Version*: https://arxiv.org/abs/2010.02618

Acknowledgements We would like to thank Kousha Etessami, Thomas Webster, and an anonymous reviewer for pointing out that the proof of Lemma 12 could be drastically simplified from its original version.

1 Introduction

Tarski's fixed point theorem states that every order preserving function on a complete lattice has a greatest and least fixed point [11], and therefore in particular, every such function has at least one fixed point. Recently, there has been interest in the complexity of finding such a fixed point. This is due to its applications, including computing Nash equilibria of supermodular games and finding the solution of a simple stochastic game [7].

Prior work has focused on the complete lattice L defined by a k-dimensional grid of width n. Dang, Qi, and Ye [2] give an algorithm that finds a fixed point of a function $f : L \to L$ using $O(\log^k n)$ queries to f. This algorithm uses recursive binary search, where a k-dimensional problem is solved by making $\log n$ recursive calls on $(k-1)$-dimensional sub-instances. They conjectured that this algorithm is optimal.

Later work of Etessami, Papadimitriou, Rubinstein, and Yannakakis took the first step towards proving this [7]. They showed that finding a Tarski fixed point in a two-dimensional lattice requires $\Omega(\log^2 n)$ queries, meaning that the Dang et al. algorithm is indeed optimal in the two-dimensional case. Etessami et al. conjectured that the Dang et al. algorithm is optimal for constant k, and they leave as an explicit open problem the question of whether their lower bound can be extended to dimension three or beyond.

Our contribution. In this paper we show that, surprisingly, the Dang et al. algorithm is not optimal in dimension three, or any higher dimension, and so we falsify the prior conjectures. We do this by giving an algorithm that can find a Tarski fixed point in three dimensions using $O(\log^2 n)$ queries, thereby beating the $O(\log^3 n)$ query algorithm of Dang et al. Our new algorithm can be used as a new base case for the Dang et al. algorithm, and this leads to a $O(\log^{k-1} n)$ query algorithm for k-dimensional instances when $k \geq 3$, which saves a $\log n$ factor over the $O(\log^k n)$ queries used by the Dang et al. algorithm.

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 29; pp. 29:1–29:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The Dang et al. algorithm solves a three-dimensional instance by making recursive calls to find a fixed point of $\log n$ distinct two-dimensional sub-instances. Our key innovation is to point out that one does not need to find a fixed point of the two-dimensional sub-instance to make progress. Instead, we define the concept of an *inner algorithm* (Definition 3) that, given a two-dimensional sub-instance, is permitted to return any point that lies in the up or down set of the three-dimensional instance (defined formally later). This is a much larger set of points, so whereas finding a fixed point of a two-dimensional instance requires $\Omega(\log^2 n)$ queries [7], we give a $O(\log n)$ query inner algorithm for two-dimensional instances. This inner algorithm is quite involved, and is the main technical contribution of the paper.

We show that, given an inner algorithm for dimension $k-1$, a reasonably straightforward *outer algorithm* can find a Tarski fixed point by making $O(k \cdot \log n)$ calls to the inner algorithm. Thus we obtain a $O(\log^2 n)$ query algorithm for the case where $k = 3$. We leave as an open problem the question of whether efficient inner algorithms exist in higher dimensions.

Though we state our results in terms of query complexity for the sake of simplicity, it should be pointed out that both our outer and inner algorithms run in polynomial time. Specifically, our algorithms will run in $O(\text{poly}(\log n, k) \cdot \log^{k-1} n)$ time when the function is presented as a Boolean circuit of size $\text{poly}(\log n, k)$.

Related work. Etessami et al. also studied the computational complexity of the Tarski problem [7], showing that the problem lies in PPAD and PLS. However, the exact complexity of the problem remains open. It is not clear whether the problem is PPAD $\cap$ PLS-complete [8], or contained in some other lower class such as EOPL or UEOPL [9].

Tarski's fixed point theorem has been applied in a wide range of settings within Economics [12, 10, 13], and in particular to settings that can be captured by supermodular games, which are in fact equivalent to the Tarski problem [7]. In terms of algorithms, Echenique [6] studied the problem of computing all pure equilibria of a supermodular game, which is at least as hard as finding the greatest or least fixed point of the Tarski problem, which is itself NP-hard [7]. There have also been several papers that study properties of Tarski fixed points, such as the complexity of deciding whether a fixed point is unique [2, 4, 3, 5]. The Tarski problem has also been studied in the setting where the partial order is given by an oracle [1].

2 Preliminaries

Lattices. We work with a complete lattice defined over a k-dimensional grid of points. We define $\text{Lat}(n_1, n_2, \ldots, n_k)$ to be the k-dimensional lattice with side-lengths given by $n_1, \ldots, n_k$. That is, $\text{Lat}(n_1, n_2, \ldots, n_k)$ contains every $x \in \mathbb{N}^k$ such that $1 \leq x_i \leq n_i$ for all $i = 1, \ldots, k$. Throughout, we use k to denote the dimensionality of the lattice, and $n = \max_{i=1}^{k} n_i$ to be the width of the widest dimension. We use $\preceq$ to denote the natural partial order over this lattice with $x \preceq y$ if and only if $x, y \in L$ and $x_i \leq y_i$ for all $i \leq k$.

The Tarski fixed point problem. Given a lattice L, a function $f : L \to L$ is *order preserving* if $f(x) \preceq f(y)$ whenever $x \preceq y$. A point $x \in L$ is *fixed point* of f if $f(x) = x$. A weak version of Tarski's theorem can be stated as follows.

▶ **Theorem 1** ([11]). *Every order preserving function on a complete lattice has a fixed point.*

Thus, we can define a total search problem for Tarski's fixed point theorem.

▶ **Definition 2** (Tarski). *Given a lattice L, and a function $f : L \to L$, find one of:*
(T1) *A point $x \in L$ such that $f(x) = x$.*
(T2) *Two points $x, y \in L$ such that $x \preceq y$ and $f(x) \not\preceq f(y)$.*

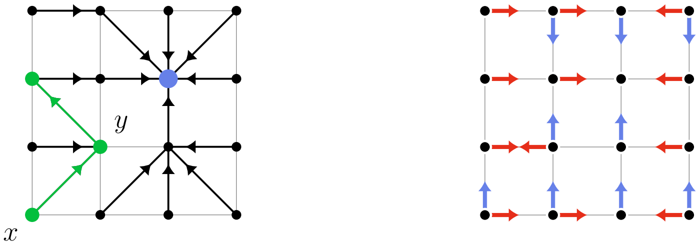


Figure 1 Left: a Tarski instance. Right: our diagramming notation for the same instance.

Solutions of type 1 are fixed points of f, whereas solutions of type 2 witness that f is not an order preserving function. By Tarski's theorem, if a function f has no solutions of type 2, then it must have a solution of type 1, and so Tarski is a total problem.

The left-hand picture in Figure 1 gives an example of a two-dimensional Tarski instance. The blue point is a fixed point, and so is a 1 solution, while the highlighted green arrows give an example of an order preservation violation, and so (x, y) is a 2 solution.

Throughout the paper we will use a diagramming notation, shown on the right in Figure 1, that decomposes the dimensions of the instance. The red arrows correspond to dimension 1, where an arrow pointing to the left indicates that $f(x)_1 \leq x_1$, while an arrow to the right[1] indicates that $x_1 \leq f(x)_1$. Blue arrows do the same thing for dimension 2, and we will use green arrows for dimension 3 in the cases where this is relevant.

The up and down sets. Given a function f over a lattice L, we define $\mathrm{Up}(f) = \{x \in L : x \preceq f(x)\}$, and $\mathrm{Down}(f) = \{x \in L : f(x) \preceq x\}$. We call $\mathrm{Up}(f)$, the *up set*, which contains all points in which f goes up according to the ordering $\preceq$, and likewise we call $\mathrm{Down}(f)$ the *down set*. Note that the set of fixed points of f is exactly $\mathrm{Up}(f) \cap \mathrm{Down}(f)$.

Slices. A *slice* of the lattice L is defined by a tuple $s = (s_1, s_2, \dots, s_k)$, where each $s_i \in \mathbb{N} \cup \{*\}$. The idea is that, if $s_i \neq *$, then we fix dimension i of L to be s_i, and if $s_i = *$, then we allow dimension i of L to be free. Formally, we define the sliced lattice $L_s = \{x \in L : x_i = s_i \text{ whenever } s_i \neq *\}$. We say that a slice is a *principle slice* if it fixes exactly one dimension and leaves the others free. For example $(1, *, *)$, $(*, 33, *)$, and $(*, *, 261)$ are all principle slices of a three-dimensional lattice.

Given a slice s, and a function $f : L \to L$, we define $f_s : L_s \to L_s$ to be the *restriction* of f to L_s. Specifically, for each $x \in L_s$, we define $(f_s(x))_i = f(x)_i$ if $s_i = *$, and $(f_s(x))_i = s_i$ otherwise. This definition projects the function f down onto the slice s.

A fact that we will use repeatedly in the paper is that an order preservation violation in a slice s is also an order preservation violation for the whole instance. More formally, if $x, y \in L_s$ satisfy $x \preceq y$ and $f_s(x) \not\preceq f_s(y)$, then we also have $f(x) \not\preceq f(y)$, since there exists a dimension i such that $f(x)_i = f_s(x)_i > f_s(y)_i = f(y)_i$.

Sub-instances. A *sub-instance* of a lattice L is defined by two points $x, y \in L$ that satisfy $x \preceq y$. Informally, the sub-instance defined by x and y is the lattice containing all points between x and y. Formally, we define $L_{x,y} = \{a \in L : x \preceq a \preceq y\}$.

[1] If $x_1 = f(x)_1$ we could use either arrow, but will clarify in the text whenever this ambiguity matters.

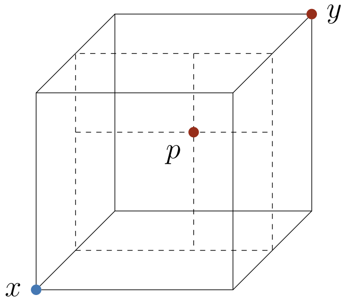


Figure 2 One iteration of the outer algorithm. The dashed lines show the principle slice chosen by the algorithm, and the point p is the point returned by the inner algorithm. In this case $p \in \mathrm{Down}(f)$, and so the algorithm focuses on the sub-instance $L_{x,p}$.

3 The Outer Algorithm

The task of the outer algorithm is to find a solution to the Tarski instance by making $O(k \cdot \log n)$ calls to the inner algorithm. We state our results for dimension k, even though we only apply the outer algorithm with $k = 3$, since, in the future, an efficient inner algorithm in higher dimensions may be found. Formally, an inner algorithm is defined as follows.

▶ **Definition 3** (Inner algorithm). *An inner algorithm for a Tarski instance takes as input a sub-instance $L_{a,b}$ with $a \in \mathrm{Up}(f)$ and $b \in \mathrm{Down}(f)$, and a principle slice s of that sub-instance. It outputs one of the following.*

- *A point $x \in L_{a,b} \cap L_s$ such that $x \in \mathrm{Up}(f)$ or $x \in \mathrm{Down}(f)$.*
- *Two points $x, y \in L_{a,b}$ that witness a violation of the order preservation of f.*

It is important to understand that here we are looking for points that lie in the up or down set of the *three-dimensional* instance, a point that lies in $\mathrm{Up}(f_s)$ but for which f goes down in the third dimension would not satisfy this criterion.

The algorithm. Throughout the outer algorithm, we will maintain two points $x, y \in L$ with the invariant that $x \preceq y$ and $x \in \mathrm{Up}(f)$ and $y \in \mathrm{Down}(f)$. The following lemma implies that if x and y satisfy the invariant, then $L_{x,y}$ must contain a solution to the Tarski problem. This will allow us to focus on smaller and smaller instances that are guaranteed to contain a solution.

▶ **Lemma 4.** *Let L be a lattice and $f : L \to L$ be a Tarski instance. If there are two points $a, b \in L$ satisfying $a \preceq b$, $a \in \mathrm{Up}(f)$, and $b \in \mathrm{Down}(f)$, then one of the following exists.*

- *A point $x \in L_{a,b}$ satisfying $f(x) = x$.*
- *Two points $x, y \in L_{a,b}$ satisfying $x \preceq y$ and $f(x) \not\preceq f(y)$.*

Moreover, there is an algorithm that finds one of the above using $O(\sum_{i=1}^{k}(a_i - b_i))$ queries.

Initially we set $x = (1, 1, \ldots, 1)$, which is the least element, and $y = (n_1, n_2, \ldots, n_k)$, which is the greatest element. Note that $x \preceq f(x)$ holds because x is the least element, and likewise $f(y) \preceq y$ holds because y is the greatest element, so the invariant holds for these two points.

Each iteration of the outer algorithm will reduce the number of points in $L_{x,y}$ by a factor of two. To do this, the algorithm selects a largest dimension of that sub-instance, which is a dimension i that maximizes $y_i - x_i$. It then makes a call to the inner algorithm for the principle slice s defined so that $s_i = \lfloor (y_i - x_i)/2 \rfloor$ and $s_j = *$ for all $j \neq i$.

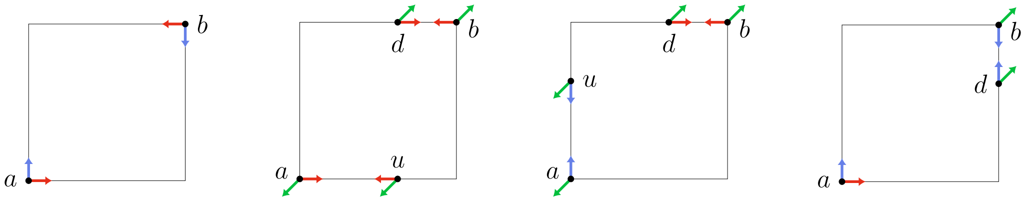


Figure 3 Four example sub-instances that satisfy the inner algorithm invariant.

1. If the inner algorithm returns a violation of order preservation in the slice, then this is also an order preservation violation in L, and so the algorithm returns this and terminates.
2. If the inner algorithm returns a point p in the slice such that $p \in \text{Up}(f)$, then the algorithm sets $x := p$ and moves to the next iteration.
3. If the inner algorithm returns a point p such that $p \in \text{Down}(f)$, then the algorithm sets $y := p$ and moves to the next iteration.

Figure 2 gives an example of this procedure.

The algorithm can continue as long as there exists a dimension i such that $y_i - x_i \geq 2$, since this ensures that there will exist a principle slice strictly between x and y in dimension i that cuts the sub-instance in half. Note that there can be at most $k \cdot \log n$ iterations of the algorithm before we arrive at the final sub-instance $L_{x,y}$ with $y_i - x_i < 2$ for all i. Lemma 4 gives us an efficient algorithm to find a solution in this final instance, which uses at most $O(\sum_{i=1}^{k}(y_i - x_i)) = O(k)$ queries. So we have proved the following theorem.

▶ **Theorem 5.** *Suppose that there exists an inner algorithm that makes at most q queries. Then a solution to the* Tarski *problem can be found by making $O(q \cdot k \cdot \log n + k)$ queries.*

4 The Inner Algorithm

We now describe an inner algorithm for three dimensions that makes $O(\log n)$ queries. Throughout this section we assume that the inner algorithm has been invoked on a sub-instance $L_{u,d}$ and principle slice s, and without loss of generality we assume that $s = (*, *, s_3)$.

Down set witnesses. Like the outer algorithm, the inner algorithm will also focus on smaller and smaller sub-instances that are guaranteed to contain a solution by an invariant, but now the invariant is more complex. To define the invariant, we first introduce the concept of a *down set witness* and an *up set witness*. The points d and b in the second example in Figure 3 give an example of a down set witness. Note that the following properties are satisfied.

- f weakly increases at d and b in dimension 3.
- d and b have the same coordinate in dimension 2.
- d weakly increases in dimension 1 while b weakly decreases in dimension 1.

We also allow down set witnesses like those given by d and b in the fourth example of Figure 3 that satisfy the same properties with dimensions 1 and 2 swapped. Thus, the formal definition of a down set witness abstracts over dimensions 1 and 2.

▶ **Definition 6** (Down set witness). *A down set witness is a pair of points (d, b) with $d, b \in L_s$ such that both of the following are satisfied.*

- *$d_3 \leq f(d)_3$ and $b_3 \leq f(b)_3$.*
- *$\exists\, i, j \in \{1, 2\}$ with $i \neq j$ s.t. $d_i = b_i$ and $d_j \leq b_j$, while $d_j \leq f(d)_j$ and $f(b)_j \leq b_j$.*

If (d, b) is a down set witness and $d_2 = b_2$, then we call (d, b) a *top-boundary* witness, while if $d_1 = b_1$, then we call (d, b) a *right-boundary* witness.

The following lemma states that if we have a down set witness (d, b), then between d and b we can find either a solution that can be returned by the inner algorithm (cases 1 and 2 of the lemma), or a point that is in the down set of the slice s (case 3 of the lemma).

Informally, the proof for a top-boundary witness (d, b) uses the fact that d and b point towards each other in dimension 1 to argue that there must be a fixed point p (or an order preservation violation) of the one-dimensional slice between d and b. Then, it is shown that either this point is in $\mathrm{Up}(f)$, and so is a solution for the inner algorithm, or it is in $\mathrm{Down}(f_s)$, or that p violates order preservation with d or b.

▶ **Lemma 7.** *If (d, b) is a down set witness, then one of the following exists.*

1. *A point c satisfying $d \preceq c \preceq b$ such that $c \in \mathrm{Up}(f)$.*
2. *Two points x, y satisfying $d \preceq x \preceq y \preceq b$ that witness order preservation violation of f.*
3. *A point c satisfying $d \preceq c \preceq b$ such that $c \in \mathrm{Down}(f_s)$.*

Up set witnesses. An up set witness is simply a down set witness in which all inequalities have been flipped. The second and third diagrams in Figure 3 show the two possible configurations of an up set witness (a, u). Note that for up set witnesses, dimension 3 is now required to weakly decrease.

▶ **Definition 8** (Up set witness). *An up set witness is a pair of points (a, u) with $a, u \in L_s$ such that both of the following are satisfied.*

- *$a_3 \geq f(a)_3$ and $u_3 \geq f(u)_3$.*
- *$\exists\ i, j \in \{1, 2\}$ with $i \neq j$ s.t. $a_i = u_i$ and $u_j \geq a_j$, while $u_j \geq f(u)_j$ and $f(a)_j \geq a_j$.*

We say that an up set witness (a, b) is a *left-boundary* witness if $a_1 = b_1$, while we call it a *bottom-boundary* witness if $a_2 = b_2$.

The following lemma is the analogue of Lemma 7 for up set witnesses. The proof simply flips all inequalities in the proof of Lemma 7.

▶ **Lemma 9.** *If (a, u) is an up set witness, then one of the following exists.*

1. *A point c satisfying $a \preceq c \preceq u$ such that $c \in \mathrm{Down}(f)$.*
2. *Two points x, y satisfying $a \preceq x \preceq y \preceq u$ that witness order preservation violation of f.*
3. *A point c satisfying $a \preceq c \preceq u$ such that $c \in \mathrm{Up}(f_s)$.*

The invariant. At each step of the inner algorithm, we will have a sub-instance $L_{a,b}$ that satisfies the following invariant.

▶ **Definition 10** (Inner algorithm invariant). *The instance $L_{a,b}$ satisfies the invariant if*

- *Either $a \in \mathrm{Up}(f_s)$ or there is a known up set witness (a, u) with $u \preceq b$.*
- *Either $b \in \mathrm{Down}(f_s)$ or there is a known down set witness (d, b) with $a \preceq d$.*

If we have both an up set witness and a down set witness then we also require that $u \preceq d$.

Figure 3 gives four example instances that satisfy the invariant. Note that there are actually nine possible configurations, since the first point of the invariant can be satisfied either by a point in the up set, a left-boundary up set witness, or a bottom-boundary up set witness, and the second point of the invariant likewise has three possible configurations.

The following lemma shows that, if the invariant is satisfied, then the sub-instance $L_{a,b}$ contains a solution that can be returned by the inner algorithm. The proof invokes Lemmas 7 and 9 to either immediately find a solution for the inner algorithm, or find two points $x \preceq y$ in the sub instance where x is in the up set and y is in the down set. The latter case allows us to invoke Lemma 4 to argue that the sub-instance contains a fixed point p of the slice s. If p weakly increases in the third dimension, then $p \in \mathrm{Up}(f)$, while if p decreases in the third dimension then $p \in \mathrm{Down}(f)$.

▶ **Lemma 11.** *If $L_{a,b}$ satisfies the invariant then one of the following exists.*

- *A point $x \in L_{a,b}$ such that $x \in \mathrm{Up}(f)$ or $x \in \mathrm{Down}(f)$.*
- *Two points $x, y \in L_{a,b}$ that witness a violation of the order preservation of f.*

A special case. There is a special case that we will encounter in the inner algorithm that requires more effort to deal with. One example of this case is shown in Case 3.a.ii of Figure 6. Here we have a point p on the right-hand boundary of the instance that satisfies $p_1 < f(p)_1$, meaning that f moves p outside of the instance. If $b \in \mathrm{Down}(f)$, or if there is a top-boundary down set witness, then it is straightforward to show that p and b violate order preservation.

However, if we have a right-boundary down set witness (d, b) with $p \preceq d$ then we need to do further work[2]. Note that the properties of a down set witness ensure that $d_2 \leq f(d)_2$ and $d_3 \leq f(d)_3$. But there are two possibilities for dimension 1. If $d_1 \leq f(d)_1$ then $d \in \mathrm{Up}(f)$ and it can be returned by the inner algorithm. On the other hand, if $d_1 > f(d)_1$, then we can show that p and d violate order preservation. We prove this formally in the following lemma.

▶ **Lemma 12.** *Let $L_{a,b}$ be a sub-instance that satisfies the invariant, and let p be a point satisfying $a \preceq p \preceq b$ that also satisfies one of the following conditions.*

1. *$p_1 = b_1$ and $p_1 < f(p)_1$.*
2. *$p_2 = b_2$ and $p_2 < f(p)_2$.*
3. *$p_1 = a_1$ and $p_1 > f(p)_1$.*
4. *$p_2 = a_2$ and $p_2 > f(p)_2$.*

Suppose further that, if there exists a down-set witness (d, b) then $p \preceq d$, and if there exists an up-set witness (a, u) then $u \preceq p$. Then there exists a solution for the inner algorithm that can be found using constantly many queries.

Initialization. The input to the algorithm is a sub-instance $L_{x,y}$, and recall that we have fixed the principle slice $s = (*, *, s_3)$. The initial values for a and b are determined as follows. For each dimension i we have $a_i = s_3$ if $i = 3$, and $a_i = x_i$ otherwise, and we have $b_i = s_3$ if $i = 3$, and $b_i = y_i$ otherwise. That is, a and b are the projections of x and y onto s.

The following lemma states that either a and b satisfy the invariant, or that we can easily find a violation of order preservation.

▶ **Lemma 13.** *Either $L_{a,b}$ satisfies the invariant, or there is violation of order preservation between a and x, or between b and y.*

The algorithm. Now suppose that we have an instance $L_{a,b}$ that satisfies the invariant. We will describe how to execute one iteration of the algorithm, which will either find a violation of order preservation, or find a new instance whose size is at most half the size of the $L_{a,b}$.

We begin by defining some important points. We define $\mathsf{mid} = \lfloor (a + b)/2 \rfloor$ to be the *midpoint* of the instance, and we define the following points, which are shown in Figure 4:

$$\mathsf{bot} = (\lfloor (a_1 + b_1)/2 \rfloor, a_2), \qquad \mathsf{left} = (a_1, \lfloor (a_2 + b_2)/2 \rfloor),$$
$$\mathsf{top} = (\lfloor (a_1 + b_1)/2 \rfloor, b_2), \qquad \mathsf{right} = (b_1, \lfloor (a_2 + b_2)/2 \rfloor).$$

[2] The case where $p \succ d$ will never occur in our algorithm, so we can ignore it.

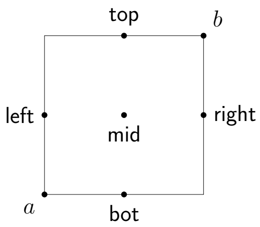


Figure 4 The five points used by the inner algorithm.

Step 1: Fixing the up and down set witnesses. Suppose that $L_{a,b}$ satisfies the invariant with a top-boundary down set witness (d, b), We would like to ensure that $\mathsf{top} \preceq d$, since otherwise if we cut the instance in half in a later step, the witness may no longer be within the sub-instance. For the same reason, we would like to ensure that (d, b) satisfies $\mathsf{right} \preceq d$ for a right-boundary down set witness, that (a, u) satisfies $u \preceq \mathsf{left}$ for a left-boundary up set witness, and that (a, u) satisfies $u \preceq \mathsf{bot}$ for a bottom-boundary up set witness. By the end of Step 1 we will have either found a violation of order preservation, moved into the next iteration with a sub-instance of half the size, or all inequalities above will hold.

Step 1 consists of the following procedure. The procedure should be read alongside Figure 5, which gives a diagram for every case presented below.

1. If (d, b) is a top-boundary down set witness and $\mathsf{top} \preceq d$ then there is no need to do anything. On the other hand, if $d \prec \mathsf{top}$ we use the following procedure.
   a. We first check if $\mathsf{top}_3 > f(\mathsf{top})_3$. If this is the case, then since the invariant ensures that $d_3 \leq f(d)_3$ we have $f(d)_3 \geq d_3 = \mathsf{top}_3 > f(\mathsf{top})_3$ so $d \preceq \mathsf{top}$ but $f(d) \not\preceq f(\mathsf{top})$, and an order preservation violation has been found and the inner algorithm terminates.
   b. We next check the whether $\mathsf{top}_1 > f(\mathsf{top})_1$. In this case, we can use (d, top) as a down set witness for the sub-instance $L_{a,\mathsf{top}}$, where we observe that top satisfies the requirements since $\mathsf{top}_3 \leq f(\mathsf{top})_3$ and $\mathsf{top}_1 > f(\mathsf{top})_1$. Hence, $L_{a,\mathsf{top}}$ satisfies the invariant (if $L_{a,b}$ also has an up set witness (a, u) then note that $u \preceq d$ continues to hold), and so the algorithm moves into the next iteration with the sub-instance $L_{a,\mathsf{top}}$.
   c. In this final case we have $\mathsf{top}_3 \leq f(\mathsf{top})_3$ and $\mathsf{top}_1 \leq f(\mathsf{top})_1$. Therefore (top, b) is a valid down set witness for $L_{a,b}$ (if $L_{a,b}$ also has an up set witness (a, u) then note that $u \preceq d \prec \mathsf{top}$). So we can replace (d, b) with (top, b) and continue, noting that our down set witness now satisfies the required inequality.
2. If (d, b) is a right-boundary down set witness and $\mathsf{right} \preceq d$ then there is no need to do anything. On the other hand, if $d \prec \mathsf{right}$ then we use the same procedure as case 1, where dimensions 1 and 2 are exchanged and the point top is replaced by the point right.
3. If (a, u) is a bottom-boundary up set witness and $u \preceq \mathsf{bot}$ then there is no need to do anything. On the other hand, if $\mathsf{bot} \prec u$ then we use the following procedure, which is the same as the procedure from case 1, where all inequalities have been flipped.
   a. We first check if $\mathsf{bot}_3 < f(\mathsf{bot})_3$. If this is the case, then since the invariant ensures that $u_3 \geq f(u)_3$ we have $f(u)_3 \leq u_3 = \mathsf{bot}_3 < f(\mathsf{bot})_3$ so $u \succeq \mathsf{bot}$ but $f(u) \not\succeq f(\mathsf{bot})$, and an order preservation violation has been found and the inner algorithm terminates.
   b. We next check the whether $\mathsf{bot}_1 < f(\mathsf{bot})_1$. In this case, we can use (bot, u) as an up set witness for the sub-instance $L_{\mathsf{bot},b}$, where we observe that bot satisfies the requirements since $\mathsf{bot}_3 \geq f(\mathsf{bot})_3$ and $\mathsf{bot}_1 < f(\mathsf{bot})_1$. Hence, $L_{\mathsf{bot},b}$ satisfies the invariant (if $L_{a,b}$ also has a down set witness (d, b) then note that $u \preceq d$ continues to hold), and so the algorithm moves into the next iteration with the sub-instance $L_{\mathsf{bot},b}$.

   c. In this final case we have $\mathsf{bot}_3 \geq f(\mathsf{bot})_3$ and $\mathsf{bot}_1 \geq f(\mathsf{bot})_1$. Therefore (a, bot) is a valid up set witness for $L_{a,b}$ (if $L_{a,b}$ also has a down set witness (d, b) then we note that $\mathsf{bot} \preceq u \preceq d$). So we can replace (a, u) with (a, bot), noting that our up set witness now satisfies the required inequality.

4. If (a, u) is a left-boundary up set witness and $u \preceq \mathsf{left}$ then there is no need to do anything. On the other hand, if $u \succ \mathsf{left}$ then we use the same procedure as case 3, where dimensions 1 and 2 are exchanged and the point left is replaced by the point bot.

Step 2: Find a smaller sub-instance. If Step 1 of the algorithm did not already move us into the next iteration of the algorithm with a smaller instance, we apply Step 2. This step performs a case analysis on the point mid. The following procedure should be read in conjunction with Figure 6, which provides a diagram for every case.

1. Check if $\mathsf{mid}_1 \leq f(\mathsf{mid})_1$ and $\mathsf{mid}_2 \leq f(\mathsf{mid})_2$. If this is the case then $\mathsf{mid} \in \mathrm{Up}(f_s)$, and so we can move to the next iteration of the algorithm with the sub-instance $L_{\mathsf{mid},b}$. Note that if $L_{a,b}$ has a down-set witness (d, b), then Step 1 of the algorithm has ensured that $\mathsf{mid} \preceq d$, and so (d, b) is also a valid down-set witness for $L_{\mathsf{mid},b}$.

2. Check if $\mathsf{mid}_1 \geq f(\mathsf{mid})_1$ and $\mathsf{mid}_2 \geq f(\mathsf{mid})_2$. If this is the case then $\mathsf{mid} \in \mathrm{Down}(f_s)$, and so we can move to the next iteration of the algorithm with the sub-instance $L_{a,\mathsf{mid}}$. Note that if $L_{a,b}$ has an up-set witness (a, u), then Step 1 of the algorithm has ensured that $u \preceq \mathsf{mid}$, and so (a, u) is also a valid down-set witness for $L_{a,\mathsf{mid}}$.

3. Check if $\mathsf{mid}_1 \leq f(\mathsf{mid})_1$ and $\mathsf{mid}_2 > f(\mathsf{mid})_2$. If so, we use the following procedure.

   a. Check if $\mathsf{mid}_3 \leq f(\mathsf{mid})_3$. If so, do the following.

      i. Check if $\mathsf{right}_3 > f(\mathsf{right})_3$. If this holds then we have $f(\mathsf{mid})_3 \geq \mathsf{mid}_3 = \mathsf{right}_3 > f(\mathsf{right})_3$, meaning that $\mathsf{mid} \preceq \mathsf{right}$ but $f(\mathsf{mid}) \not\preceq f(\mathsf{right})$. Thus we have found a violation of order preservation and the algorithm terminates.

      ii. Check if $\mathsf{right}_1 < f(\mathsf{right})_1$. If this holds then we use Lemma 12 (with $p := \mathsf{right}$) to find a solution that can be returned by the inner algorithm.

      iii. If we reach this case then we have $\mathsf{mid}_3 \leq f(\mathsf{mid})_3$ and $\mathsf{right}_3 \leq f(\mathsf{right})_3$, while we also have $\mathsf{mid}_1 \leq f(\mathsf{mid})_1$ and $\mathsf{right}_1 \geq f(\mathsf{right})_1$. Thus $(\mathsf{mid}, \mathsf{right})$ is a valid down set witness for the instance $L_{a,\mathsf{right}}$. Note that if $L_{a,b}$ has an up set witness (a, u), then Step 1 ensures that $u \preceq \mathsf{mid}$, and so $L_{a,\mathsf{right}}$ satisfies the invariant. So the algorithm moves to the next iteration with sub-instance $L_{a,\mathsf{right}}$.

   b. In this case we have $\mathsf{mid}_3 > f(\mathsf{mid})_3$. The following three steps are symmetric to those used in Case 3.a, but with all inequalities flipped, dimension 1 substituted for dimension 2, the point bot substituted for right, and the point a substituted for b.

      i. Check if $\mathsf{bot}_3 > f(\mathsf{bot})_3$. If this holds then we have $f(\mathsf{mid})_3 \leq \mathsf{mid}_3 = \mathsf{bot}_3 < f(\mathsf{bot})_3$, meaning that $\mathsf{mid} \succeq \mathsf{bot}$ but $f(\mathsf{mid}) \not\succeq f(\mathsf{bot})$. Thus we have found a violation of order preservation and the algorithm terminates.

      ii. Check if $\mathsf{bot}_2 > f(\mathsf{bot})_2$. If this holds then we can use Lemma 12 (with $p := \mathsf{bot}$) to find a solution that can be returned by the inner algorithm.

      iii. If we reach this case then we have $\mathsf{mid}_3 \geq f(\mathsf{mid})_3$ and $\mathsf{bot}_3 \geq f(\mathsf{bot})_3$, while we also have $\mathsf{mid}_2 \geq f(\mathsf{mid})_2$ and $\mathsf{bot}_2 \leq f(\mathsf{bot})_2$. Thus $(\mathsf{bot}, \mathsf{mid})$ is a valid up set witness for the instance $L_{\mathsf{bot},b}$. Note that if $L_{a,b}$ has a down set witness (d, b), then Step 1 of the algorithm ensures that $d \succeq \mathsf{mid}$, and so $L_{\mathsf{bot},b}$ satisfies the invariant. The algorithm will therefore move to the next iteration with the sub-instance $L_{\mathsf{bot},b}$.

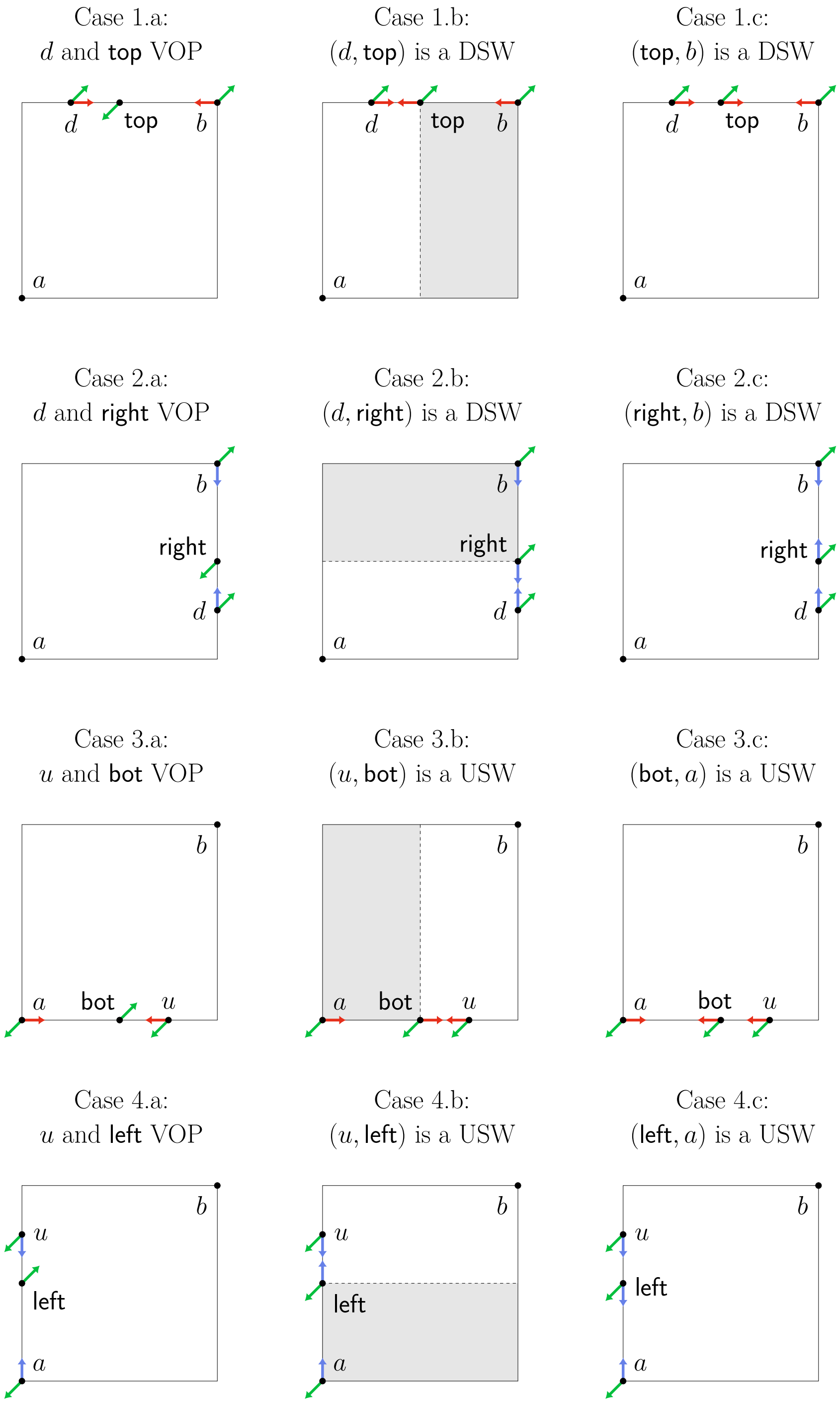


Figure 5 All cases used in Step 1 of the algorithm. In the labels, VOP is short for “violate order preservation”, DSW is short for “down set witness”, and USW is short for “up set witness”.

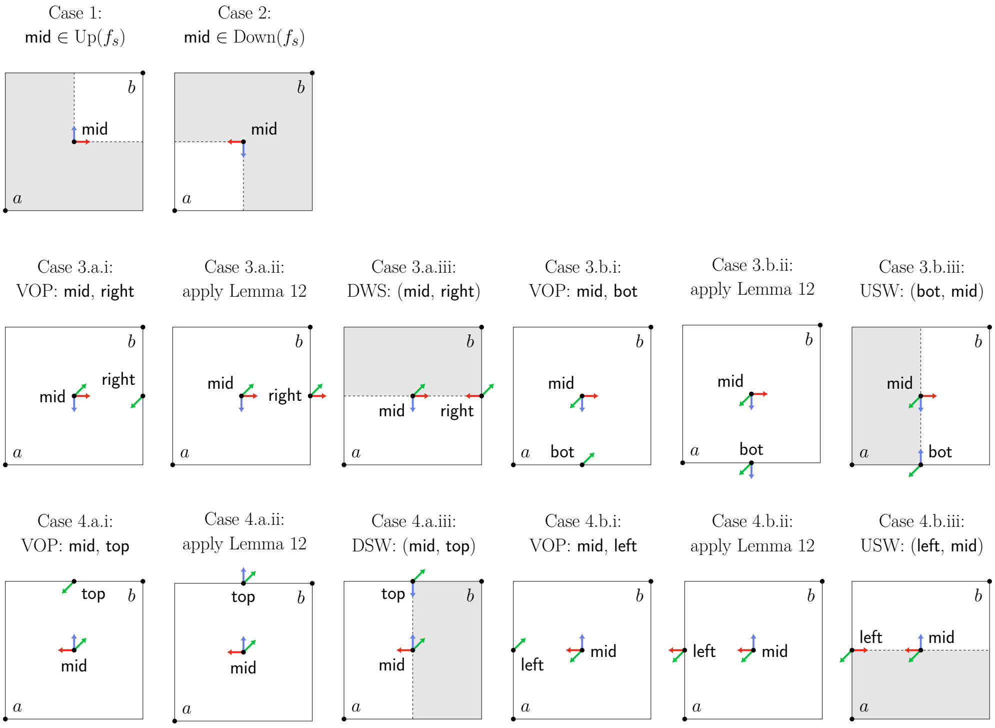


Figure 6 All cases used in Step 2 of the algorithm. In the labels, VOP is short for "violate order preservation", DSW is short for "down set witness", and USW is short for "up set witness".

4. In this final case we have $\mathsf{mid}_1 > f(\mathsf{mid})_1$ and $\mathsf{mid}_2 \leq f(\mathsf{mid})_2$. Here we follow the same procedure as Case 3, but with dimensions 1 and 2 exchanged, every instance of the point right replaced with top, and every instance of bot replaced with left.

The terminal phase of the algorithm. The algorithm can continue so long as $b_1 \geq a_1 + 2$ and $b_2 \geq a_2 + 2$, since this ensures that all cases will cut the width of one of the dimensions in half. The algorithm terminates once we have both $b_1 \leq a_1 + 1$ *and* $b_2 \leq a_2 + 1$. However, once there exists only one dimension i for which $b_i \leq a_i + 1$, we must be careful, since now the midpoint lies on the boundary of the instance, and some of the cases of the algorithm may not rule out anything. We deal with this scenario separately.

There are two distinct cases that we must deal with. The first case is a *width-one instance*, in which $b_i = a_i + 1$ for some index $i \in \{1, 2\}$, and $b_j > a_j + 1$ for the index $j \in \{1, 2\}$ with $j \neq i$, meaning that the width of the shortest dimension is exactly one. These instances are problematic because the midpoint mid will now lie on the boundary of the instance, and due to this, it is possible that the algorithm may be unable to proceed.

We must also deal with *width-zero instances*, in which $b_i = a_i$ for some index $i \in \{1, 2\}$, and $b_j > a_j + 1$ for the index $j \in \{1, 2\}$ with $j \neq i$. These are one-dimensional subinstances, and once again it is possible for the algorithm to be unable to proceed.

We will use special procedures for width-one and width-zero instances, which we outline below.

Width-one instances. In the presentation below we will assume that the index $i = 1$, meaning that $b_1 = a_1 + 1$ (and hence the left-right width of the instance is one). The case for $i = 2$ is symmetric.

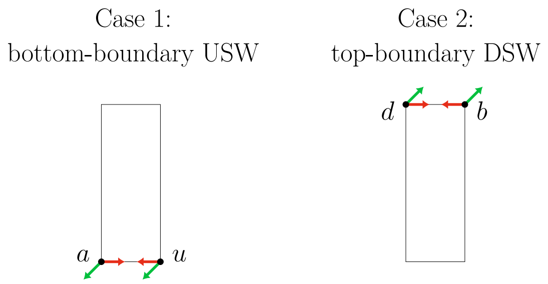


Figure 7 The two cases that trigger the preprocessing step for width-one instances.

When the algorithm is presented with a width-one instance, it first performs some preprocessing to ensure that there is no bottom-boundary up set witnesses, or top-boundary down set witness. The preprocessing considers the following two cases, which are shown in Figure 7.

1. If the instance has a bottom-boundary up set witness (a, u), then note that a and u are directly adjacent in dimension 1, and so Lemma 9 implies that we can either return a or u as a solution for the inner algorithm, or that $a \in \mathrm{Up}(f_s)$, or $u \in \mathrm{Up}(f_s)$.
   a. If $a \in \mathrm{Up}(f_s)$, then the instance $L_{a,b}$ continues to satisfy the invariant if we delete the up set witness.
   b. If $u \in \mathrm{Up}(f_s)$, then the width-zero instance $L_{u,b}$ satisfies the invariant, where we note that if $L_{a,b}$ has a down set witness (d, b), then since $u \preceq d$, we have that (d, b) is also a valid down set witness for $L_{u,b}$.
2. If the instance has a top-boundary down set witness (d, b), then note that d and b are directly adjacent in dimension 1, and so Lemma 7 implies that we can either return d or b as a solution for the inner algorithm, or that $d \in \mathrm{Down}(f_s)$, or $b \in \mathrm{Down}(f_s)$.
   a. If $b \in \mathrm{Down}(f_s)$, then the instance $L_{a,b}$ continues to satisfy the invariant if we delete the down set witness.
   b. If $d \in \mathrm{Down}(f_s)$, then the width-zero instance $L_{a,d}$ satisfies the invariant, where we note that if $L_{a,b}$ has an up set witness (a, u), then since $u \preceq d$, we have that (a, u) is also a valid up set witness for $L_{a,d}$.

With the preprocessing completed, the algorithm uses two separate runs of Steps 1 and 2, which each use a different midpoint. In the first run we use midone $= \lfloor (a + b)/2 \rfloor$ as normal, while in the second run we use midtwo $= \lceil (a + b)/2 \rceil$ as the midpoint, and we also change the definitions of bot, top, left, and right to round up instead of down. If either of the two runs decrease the size of the instance, then we move to the next iteration on the smaller instance, where the reasoning given in Steps 1 and 2 ensures that the instance continues to satisfy the invariant.

However, it could be the case that both runs do not decrease the size of the instance. Due to the preprocessing, if Step 1 attempts to recurse on a smaller sub-instance then it must succeed, since the only problematic cases are Case 1.b and Case 3.b, which both depend on the existence of a top or bottom-boundary witness, and the preprocessing ensures that these cannot exist.

On the other hand, Step 2 can fail to make progress in both runs. For the case where $i = 1$, this can only occur if Case 3.b.iii of Step 2 triggered for the run with midone and Case 4.a.iii triggered for the run with midtwo. But we can argue that in this case a solution to the inner algorithm is easy to find.

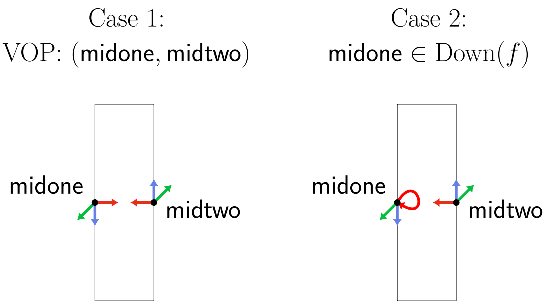


Figure 8 The two cases that are considered for width-one instances, when both runs of the algorithm fail to make progress. In the left instance, $f(\mathsf{midone})$ strictly increases in dimension 1; in the right instance, $f(\mathsf{midone})$ does not move in dimension one, which we indicate with the self loop.

Note that Case 3.b.iii can only trigger for midone when $\mathsf{midone}_1 \leq f(\mathsf{midone})_1$, while Case 4.a.iii can only trigger for midtwo when $f(\mathsf{midtwo})_1 < \mathsf{midtwo}$. Both of the following cases are shown in Figure 8.

1. If $\mathsf{midone}_1 < f(\mathsf{midone})_1$ then we have

$$f(\mathsf{midtwo})_1 \leq \mathsf{midtwo}_1 - 1 = \mathsf{midone}_1 < f(\mathsf{midone})_1,$$

so we have $\mathsf{midone} \preceq \mathsf{midtwo}$ but $f(\mathsf{midone}) \not\preceq f(\mathsf{midtwo})$, meaning that midone and midtwo witness a violation of order preservation.

2. If $\mathsf{midone}_1 = f(\mathsf{midone})_1$, then note that Case 3.b.iii ensures that $\mathsf{midone}_2 \geq f(\mathsf{midone})_2$ and $\mathsf{midone}_3 \geq f(\mathsf{midone})_3$. Therefore midone is in $\text{Down}(f)$, so can be returned by the inner algorithm.

So in both cases a solution to the inner algorithm has been found.

Width-zero instances. We again describe the procedure for the case where $i = 1$, meaning that the instance has width zero in the left-right dimension. The case where $i = 2$ is symmetric.

The algorithm begins by performing a preprocessing step that removes any top-boundary down set witnesses or bottom-boundary up set witnesses. If there is a bottom-boundary up set witness (a, u) then note that $a = u$, and therefore Lemma 9 implies that either a is a solution that can be returned by the inner algorithm, or that $a \in \text{Up}(f_s)$. Likewise, if there is a top-boundary down set witness (d, b), then $d = b$, and Lemma 7 implies that either d can be returned by the inner algorithm, or $d \in \text{Down}(f_s)$. Thus, the preprocessing step can either find a solution for the inner algorithm, or produce an instance that satisfies the invariant that has no top-boundary down set witness and no bottom-boundary up set witness.

Once the preprocessing has taken place, the algorithm proceeds through Step 1 and Step 2 as normal. If those steps make progress, then we continue on the smaller width-zero instance. If they do not make progress, then we will show that the inner algorithm can terminate after making at most $O(\log n)$ further queries.

We first observe that the only cases of Step 1 that would fail to make progress are Case 1.b and Case 3.b, but neither of those cases can trigger because the preprocessing step ensures that there is no bottom-boundary up set witness or top-boundary down set witness.

On the other hand, Case 3.b.iii and Case 4.a.iii of Step 2 can fail to make progress. We show how, in each of these cases, a solution for the inner algorithm can be found by making at most $O(\log n)$ extra queries. All of the following cases are depicted in Figure 9.

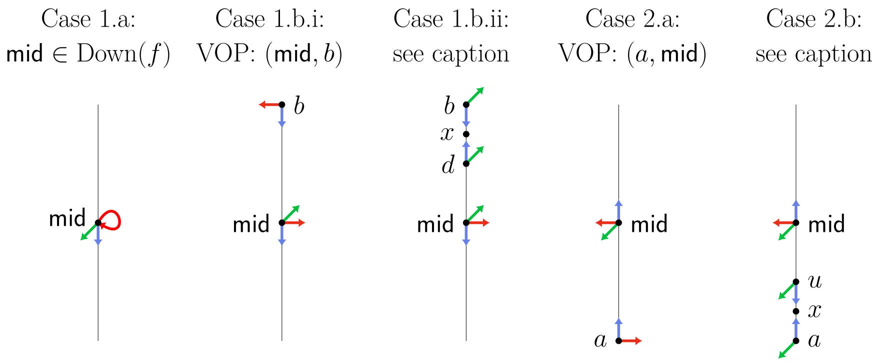


Figure 9 The five cases that can be encountered if the algorithm fails to make progress for a width-zero instance. In Cases 1.b.ii and 2.b we spend $O(\log n)$ queries to find the point x, which then allows us to terminate.

1. If Case 3.b.iii is triggered, then note that $\mathsf{mid}_1 \leq f(\mathsf{mid})_1$. There are two cases to consider.
   a. If $\mathsf{mid}_1 = f(\mathsf{mid})_1$, then since $\mathsf{mid}_2 \geq f(\mathsf{mid})_2$ and $\mathsf{mid}_3 \geq f(\mathsf{mid}_3)$, we have that $\mathsf{mid} \in \mathrm{Down}(f)$, meaning that mid can be returned by the inner algorithm.
   b. If $\mathsf{mid}_1 < f(\mathsf{mid})_1$ then we argue that an order preservation violation can be found using at most $O(\log n)$ queries.
      i. If $b \in \mathrm{Down}(f_s)$, then we have

$$f(b)_1 \leq b_1 = \mathsf{mid}_1 < f(\mathsf{mid})_1,$$

      meaning that $\mathsf{mid} \preceq b$, but $f(\mathsf{mid}) \not\preceq f(b)$, and so mid and b violate order preservation.
      ii. If instead there is a down set witness (d, b), then note that due to the preprocessing, it must be a right-boundary down set witness, and due to Step 1, we must have $\mathsf{mid} \preceq d$. By Lemma 7 there exists a point x satisfying $d \preceq x \preceq b$ that can either be returned by the inner algorithm, or that satisfies $x \in \mathrm{Down}(f_s)$. Furthermore, using we can find this point in $O(\log n)$ queries using binary search. Thus we can spend $O(\log n)$ queries and either immediatly terminate, or in the case where we find a point $x \in \mathrm{Down}(f_s)$, we can repeat the argument above to show that mid and x violate order preservation.
2. If Case 4.a.iii is triggered, then note that $\mathsf{mid}_1 > f(\mathsf{mid})_1$, and we argue that an order preservation violation can be found using at most $O(\log n)$ queries.
   a. If $a \in \mathrm{Up}(f_s)$, then we have

$$f(\mathsf{mid})_1 < \mathsf{mid}_1 = a_1 \leq f(\mathsf{mid})_1,$$

   meaning that $a \preceq \mathsf{mid}$, but $f(a) \not\preceq f(\mathsf{mid})$, and so a and mid violate order preservation.
   b. If instead there is an up set witness (a, u), then note that due to the preprocessing, it must be a left-boundary up set witness, and due to Step 1, we must have $u \preceq \mathsf{mid}$. By Lemma 9 there exists a point x satisfying $a \preceq x \preceq u$ that can either be returned by the inner algorithm, or that satisfies $x \in \mathrm{Up}(f_s)$. Furthermore, we can find this point in $O(\log n)$ queries using binary search. Thus we can spend $O(\log n)$ queries and either immediatly terminate, or in the case where we find a point $x \in \mathrm{Up}(f_s)$, we can repeat the argument above to show that mid and x violate order preservation.

Termination. If the algorithm does not hit any of the cases that return a solution immediately, then it will continue until it finds an instance $L_{a,b}$ with $b_1 \leq a_1 + 1$ and $b_2 \leq a_2 + 1$ that satisfies the invariant. Lemma 11 implies that any sub-instance that satisfies the invariant contains a solution that can be returned by the inner algorithm. Since then $L_{a,b}$ contains at most four points, we can check all of them and then return the solution that must exist.

Query complexity. Observe that each iteration of the algorithm either finds a violation of order preservation, finds a solution after spending $O(\log n)$ further queries, or reduces the size of one of the dimensions by a factor of two. Moreover, each non-terminating iteration of the algorithm queries at most five points. Hence, if the algorithm is run on a sub-instance $L_{a,b}$ with $n_1 = b_1 - a_1$ and $n_2 = b_2 - a_2$, then the algorithm will terminate after making at most $O(\log n_1 + \log n_2 + \log n)$ queries. So the overall query complexity of the algorithm is $O(\log n)$, and we have shown the following theorem.

▶ **Theorem 14.** *There is an $O(\log n)$-query inner algorithm for 3-dimensional* Tarski.

Theorems 5 and 14 imply that 3-dimensional Tarski can be solved using $O(\log^2 n)$ queries, and this can be combined with the $\Omega(\log^2 n)$ lower bound for two-dimensional Tarski [7], to give the following theorem.

▶ **Theorem 15.** *The deterministic query complexity of three-dimensional* Tarski *is $\Theta(\log^2 n)$.*

5 Extension to higher dimensions

We now extend our results to show that k-dimensional Tarski can be solved using $O(\log^{k-1} n)$ queries. The algorithm of Dang et al. [2] solves a k-dimensional Tarski instance by making $O(\log n)$ recursive calls to an algorithm for solving $(k-1)$-dimensional Tarski instances. Our algorithm can be plugged into this recursion as a new base case for $k = 3$.

The following lemma is a consequence of the work of Dang et al. [2]. However, their algorithm deals with the promise version of Tarski in which it is assumed that the input function is order preserving. For this reason, we provide our own proof of the lemma in which we give a variation of the algorithm that either finds a fixed point or explicitly provides a violation of order preservation.

▶ **Lemma 16.** *If $(k-1)$-dimensional* Tarski *can be solved using q queries, then k-dimensional* Tarski *can be solved using $(q+2) \cdot (\log n + 2)$ queries.*

The direct consequence of Lemma 16 and our $O(\log^2 n)$ query algorithm for three-dimensional Tarski is the following theorem.

▶ **Theorem 17.** *k-dimensional* Tarski *can be solved using $O(\log^{k-1} n)$ queries for $k \geq 3$,*

Time complexity. To obtain time complexity results, note that writing down a point in the lattice L already requires $k \cdot \log n$ time. We assume that f is implemented by a Boolean circuit of size that is polynomial in k and $\log n$. With this assumption, our time complexity result can be stated as follows.

▶ **Theorem 18.** *If f is presented as a Boolean circuit of size $\text{poly}(\log n, k)$, then for $k \geq 3$ there is an algorithm for* Tarski *that runs in time $O(\text{poly}(\log n, k) \cdot \log(n)^{k-1})$.*

6 Conclusion

Our $O(\log^{k-1} n)$ query algorithm for k-dimensional TARSKI falsifies prior conjectures that the problem required $\Omega(\log^k n)$ queries [2, 7]. This, of course, raises the question of what is the query complexity of finding a Tarski fixed point? While our upper bound is tight in three dimensions, it seems less likely to be the correct answer in higher dimensions. Indeed, there seems to be a fairly wide range of possibilities. Is it possible to show a $\log^{\Omega(k)} n$ query lower bound for the problem? Or perhaps there exists a fixed parameter tractable algorithm that uses $O(f(k) \cdot \log^2 n)$ queries? Both of those would be consistent with the known upper and lower bounds, and so further research will be needed to close the gap.

References

1 Ching-Lueh Chang, Yuh-Dauh Lyuu, and Yen-Wu Ti. The complexity of Tarski's fixed point theorem. *Theor. Comput. Sci.*, 401(1-3):228–235, 2008.

2 Chuangyin Dang, Qi Qi, and Yinyu Ye. Computations and complexities of Tarski's fixed points and supermodular games. *CoRR*, abs/2005.09836, 2020. Stanford tech report version appeared in 2012. arXiv:2005.09836.

3 Chuangyin Dang and Yinyu Ye. On the complexity of a class of discrete fixed point problems under the lexicographic ordering. Technical report, City University of Hong Kong, 2018. CY2018-3, 17 pages.

4 Chuangyin Dang and Yinyu Ye. On the complexity of an expanded tarski's fixed point problem under the componentwise ordering. *Theor. Comput. Sci.*, 732:26–45, 2018.

5 Chuangyin Dang and Yinyu Ye. Erratum/correction to "on the complexity of an expanded tarski's fixed point problem under the componentwise ordering" [Theor. Comput. Sci. 732 (2018) 26-45]. *Theor. Comput. Sci.*, 817:80, 2020.

6 Federico Echenique. Finding all equilibria in games of strategic complements. *J. Econ. Theory*, 135(1):514–532, 2007.

7 Kousha Etessami, Christos H. Papadimitriou, Aviad Rubinstein, and Mihalis Yannakakis. Tarski's theorem, supermodular games, and the complexity of equilibria. In *Proc. of ITCS*, volume 151, pages 18:1–18:19, 2020.

8 John Fearnley, Paul W. Goldberg, Alexandros Hollender, and Rahul Savani. The complexity of gradient descent: CLS = PPAD ∩ PLS. *CoRR*, abs/2011.01929, 2020. To appear in the Proccedings of the 53nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2021. arXiv:2011.01929.

9 John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique end of potential line. *J. Comput. Syst. Sci.*, 114:1–35, 2020.

10 Paul Milgrom and John Roberts. Rationalizability, learning, and equilibrium in games with strategic complementarities. *Econometrica*, 58(6):1255–1277, 1990.

11 Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.

12 Donald M. Topkis. Equilibrium points in nonzero-sum n-person submodular games. *SIAM J. Control Optim*, 17:773–787, 1979.

13 Donald M. Topkis. *Supermodularity and Complementarity.* Princeton University Press, 1998.

Solving One Variable Word Equations in the Free Group in Cubic Time

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

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

Abstract

A word equation with one variable in a free group is given as $U = V$, where both U and V are words over the alphabet of generators of the free group and X, X^{-1}, for a fixed variable X. An element of the free group is a solution when substituting it for X yields a true equality (interpreted in the free group) of left- and right-hand sides. It is known that the set of all solutions of a given word equation with one variable is a finite union of sets of the form $\{\alpha w^i \beta \,:\, i \in \mathbb{Z}\}$, where α, w, β are reduced words over the alphabet of generators, and a polynomial-time algorithm (of a high degree) computing this set is known. We provide a cubic time algorithm for this problem, which also shows that the set of solutions consists of at most a quadratic number of the above-mentioned sets. The algorithm uses only simple tools of word combinatorics and group theory and is simple to state. Its analysis is involved and focuses on the combinatorics of occurrences of powers of a word within a larger word.

2012 ACM Subject Classification Mathematics of computing → Combinatorics on words; Theory of computation → Formalisms; Computing methodologies → Equation and inequality solving algorithms

Keywords and phrases Word equations, free group, one-variable equations

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.30

Related Version *Full Version*: http://arxiv.org/abs/2101.06201

Funding This work was supported under National Science Centre (NCN), Poland project number 2017/26/E/ST6/00191.

1 Introduction

Word equations in the free group. A word equation is a formal equation $U = V$ in which both U, V contain letters from a fixed set (called alphabet) Σ and variables; a solution is a substitution of variables by words over Σ such that this formal equation is turned into an equality. We consider such equations in a free group, so the aforementioned equality is interpreted as the equality in the free group generated by Σ; naturally, we allow the usage of inverses of variables and generators in the equations. The satisfiability problem (of word equation over the free group) is to decide, whether the input equation has a solution. By solving the equation we mean to return an (explicit or effective) representation of all solutions.

The first algorithm for the satisfiability problem was given by Makanin [27] and it is an involved generalization of Makanin's algorithm for the satisfiability of word equation in the free monoid [26]; Razborov generalized the algorithm so that it solves word equations in the free group [32]; the description is infinite and is known as Makanin-Razborov diagrams. Makanin's algorithm is very involved and known to be not primitively recursive [21], the same applies to Razborov's generalisation, which was the first step of solving Tarski's conjectures (on elementary equivalence and decidability of the theory of free groups) [20, 33]. A different approach based on Plandowski's algorithm for the free monoid case [31] was later

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 30; pp. 30:1–30:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

proposed [8], and an even simpler approach, which gives also a finite description of the solution set, was given by Diekert, Plandowski and Jeż [9], it extends Jeż's algorithm for the free monoid case [16].

The problem of word equations in the free group was first investigated by Lyndon [25], who considered the restricted variant of one-variable equations. He showed that the solution set is a finite union of sets of the form

$$\{w_0 w_1^{i_1} w_2 w_3^{i_2} \cdots w_{2k-1}^{i_k} w_{2k} \;:\; i_1, \ldots, i_k \in \mathbb{Z}\} \ , \tag{1}$$

where $w_0, \ldots, w_{2k}$ are words over the generators of the free group, we call such sets k-parametric. In fact, it was first shown using combinatorial arguments that a superset of all solutions is of this form, and using algebraic methods the superset of all solutions is transformed into the actual set of all solutions. As a result, k depends on the equation and is a by-product of the algorithm rather than an explicitly given number. By using a more refined, though purely combinatorial, argument Appel [1] showed that there exists a superset of solutions that is a finite union of 1-parametric sets and that one can test for which values such words are indeed solutions. In principle, the proof can be readily used as an algorithm, but no reasonable bounds can be derived from it. Unfortunately, Appel's proof contains an error (see [5] for a discussion). A similar characterization was announced by Lorentz [23], but the proof was not supplied. Chiswell and Remeslennikov [5] used a different approach, based on geometric group theory, to show that the solution is a finite union of 1-parametric sets. However, their argument does not give any algorithm for solving an equation. Gilman and Myasnikov [12] gave a proof that the solution set is 4-parametric; their proof is based on formal language theory and is considerably simpler and shorter than the other known ones, however, it yields no algorithm.

A polynomial-time algorithm solving the one-variable word equations (in the free group) was given by Bormotov, Gilman and Myasnikov [3]. In principle, their argument is similar to Appel, though simpler (and without errors), and extra care is taken to guarantee that testing takes polynomial time. The running time is high, though little effort was made to lower the exponent, we believe that simple improvements and better analysis should yield $\mathcal{O}(n^5)$ running time of their algorithm.

It is known that already two-variable word equations (in the free group) do not always have a parametrizable solution set [2], here a parametrizable set is a generalization of parametric sets (1) in which the exponents using integer parameters can be nested and one exponent may depend on different parameters. Moreover, no polynomial-time algorithm for two-variable equations is known. Other restricted cases were also investigated, say the famous Lyndon-Schützenberger Theorem was originally shown for the free group [24] and satisfiability of quadratic word equations is known to be NP-complete [19] in the case of free group.

Our results and proof outline. We present an $\mathcal{O}(n^2 m)$ algorithm for solving equations with one variable in a free group, where n is the length of the equation and m the number of occurrences of the variable in it.

▶ **Theorem 1.** *Given a word equation with one variable in a free group, with length n and m occurrences of the variable, we can compute the set of all its solutions in time $\mathcal{O}(n^2 m)$. The set of solutions is a union of $\mathcal{O}(n^2)$ sets of the form $\{\alpha w^k \beta \;:\; k \in \mathbb{Z}\}$, where α, w, β are words over the generators of the given free group.*

The running time is achieved in the RAM model, more specifically we require that operations on $\log n$-bits long integers (and byte-arrays) can be performed in $\mathcal{O}(1)$ time. If this is not the case, then the running time increases by a multiplicative $\mathcal{O}(\log n)$ factor. Note that in Theorem 1 we allow $w = \varepsilon$, i.e. the set $\{\alpha w^k \beta \ : \ k \in \mathbb{Z}\}$ from Theorem 1 may consist of a single string.

The $\mathcal{O}(n^2 m)$ running time seems hard to improve: all known characterization of solution set include $\Omega(n^2)$ individual words that should be tested as solutions and natural testing of a single solution is done in $\Theta(m)$ time, note that this does not take into account the 1-parametric sets that do depend on the parameter, which seem to be harder to be tested.

We use a previous characterization of the solution superset [3], from which it follows that the main task is to compute, given words α, u, v, β, for which $i, j \in \mathbb{Z}$ the word $\alpha u^i v^j \beta$ is a solution. Roughly speaking, the previous approaches [1, 3] argued that if $\alpha u^i v^j \beta$ is a solution for a "large enough" i then $\alpha u^{i'} v^j \beta$ is a solution for each $i' \in \mathbb{Z}$; thus one has to check some "small" is and one "large enough"; for each fixed i we substitute its value and similarly argue that if j is "large enough" then each j' yields a solution (the actual argument is more subtle and symmetric in terms of i and j). We refine this approach: previously the tested values of i and j did not depend on the actual equation, but only on its length. We identify a small set of candidate pairs (i, j) based on the actual equation. To this end, we substitute $\alpha u^I v^J \beta$ to the equation, where I, J are integer variables, and intend to verify, for which values (i, j) of variables (I, J) it is a solution. Such parametric candidates cannot be tested as solutions (in particular because it could be that only for some values of I and J they indeed are solutions), however, some operations can be performed on u^I (or v^J), regardless of the actual value substituted for I: say $u^I u^I u^{-1} u^{-I}$ is equal to u^{I-1} (in a free group). After performing all such possible operations we obtain a word with "parametric powers" of u, v, i.e. powers, whose exponents depend on parameters I, J, note that the parameters are the same for all powers in the parametric word, but the actual exponents in different powers may be different. If there are only powers of u (or only powers of v) then using known tools one can show that one of those exponents is (almost) 0. This yields a linear set of possible is that should be tested. Ideally, we would like to say that a similar claim holds also when parametric powers of both u and v are present. However, those powers can interact and such an approach does not work directly. Instead, if $I = i, J = j$ yields a solution, then substituting $I = i$ (as a mental experiment) either reduces the whole word to ε, in which case each $J = j$ yields a solution, or leaves only powers of v, in which case we can reiterate the same approach, this time for powers of v. The former case gives a set of candidates for I, the latter for J, technically those depend on the substituted i, but this dependency can be removed by further analysis. A similar analysis can be made for substitution $J = j$, together yielding a superset of all possible solutions, which are then individually tested.

Additional analysis is needed to bound the number of candidates that is obtained in this way. To this end, we analyze the set of possible exponents of powers of u and v. In particular, we show that initially all such exponents are of the form $\pm I + c$ and $\pm J + c$, which allows for much better estimations: for the candidate solution to be different, the constants in those expressions need to be different and to have a factor u^{I+c} some $c|u|$ letters from the equation are "consumed" and easy calculations show that there are only $\mathcal{O}(\sqrt{n})$ different possible constants, which leads to $\mathcal{O}(\sqrt{n})$ different candidates. One has to take special care of α, β, as their introduction can yield a quadratic-size equation. To avoid this, we analyze how powers of u in concatenations of words can be obtained.

In most cases, we reduce the problem in the free group to the problem in the free monoid (with involution) and use standard tools of word combinatorics. However, this requires some additional properties of words α, u, v, β. Those cannot be inferred from known characterizations, and so known proofs are reproved and the additional claims are shown.

Connection to word equation in the free monoid. The connection between word equations in the free group and free monoid is not perfectly clear. On one hand, the satisfiability of the former can be reduced to the satisfiability of word equations over the free monoid (with involution), this was implicitly done by Makanin [27] and explicitly by Diekert et al. [8] and so generalizations of algorithms for the monoid case are used for the group case. However, there is an intuition that the additional group structure should make the equations somehow easier. This manifests for instance for quadratic equations (so the case when each variable is used at most twice), for which an NP algorithm was given for the free-group case [19] and no such result is known for the free monoid case. Furthermore, the whole first-order theory of equations over the free group is decidable [20], while already one alternation of the quantifiers make a similar theory for monoid undecidable (see [6] for an in-depth discussion of undecidable and decidable fragments).

On the other hand, such general reductions increase the number of variables and so are not suitable in the bounded number of variables case. In particular, a polynomial time algorithm for the satisfiability of two-variable equations for the free monoid is known [10], in contrast to the case of the free group (the set of solutions is still not parametrisable [13], as in the case of the free group.).

Word equations in free monoid with restricted number of variables. Word equations in the free monoid with restricted number variables were also considered. For one variable a cubic-time algorithm is trivial and can be easily improved to quadratic-running time [7]. Eyono Obono, Goralcik and Maksimenko gave a first non-trivial algorithm running in time $\mathcal{O}(n \log n)$ [30]. This was improved by Dąbrowski and Plandowski [11] to $\mathcal{O}(n + m \log n)$, where m is the number of occurrences of the variable in the equation, and to $\mathcal{O}(n)$ by Jeż [15]; the last two algorithms work in the RAM model, i.e. they assume that operations on the $\log n$-bits long numbers can be performed in constant time. The properties of the solution set were also investigated: all above algorithms essentially use the fact that the solution set consists of at most one 1-parametric set and $\mathcal{O}(\log n)$ other solutions [30]. Plandowski and Laine showed that the solution set is either exactly a 1-parametric set or of size $\mathcal{O}(\log m)$ [22] and conjectured that in the latter case there are at most 3 solutions. This conjecture was recently proved by Saarela and Nowotka [29] using novel techniques.

Word equations in the free monoid with two variables were also investigated. it was shown by Hmelevskiĭ [13] that there are equations whose solution set is not parametrizable. The first polynomial-time algorithm (of a rather high degree) for satisfiability of such equations was given by Charatonik and Pacholski [4], this was improved to $\mathcal{O}(n^6)$ by Ille and Plandowski [14] and later to $\mathcal{O}(n^5)$ by Dąbrowski and Plandowski [10], the latter algorithm also returns a description of all solutions. The computational complexity of word equations with three variables is unknown, similarly, the computational complexity of satisfiability in the general case of word equations in the free monoid remains unknown (it is NP-hard and in PSPACE).

2 Definitions and preliminaries

2.1 Notions

Monoids, monoids with involution. By Σ we denote an alphabet, which is endowed with involution $\bar{\cdot} : \Sigma \to \Sigma$, i.e. a function such that $\overline{a} \neq a = \overline{\overline{a}}$. The free monoid Σ^* with involution consists of all finite words over Σ and the involution uniquely extended from Σ to Σ^* by requiring that $\overline{(uv)} = \overline{v}\,\overline{u}$, i.e. we think of it as of inverse in a group. We denote the empty word by ε. Given a word uvw: u is its prefix, w suffix and v its subword; for a word w often w' and w'' will denote the prefix and suffix of w, this will be always written explicitly. A word $w = a_1 \cdots a_k$, where $a_1, \ldots, a_k \in \Sigma$, has length $|w| = k$ and $w[i\,..\,j]$ denotes a subword $a_i \cdots a_j$. For $k \geq 0$ a word u^k is a k-th *power* of u (or simply u-power), by convention u^{-k} denotes $\overline{u}^k$. A u-power prefix (suffix) of v is the longest u-power that is prefix (suffix, respectively) of v, note that this may b a positive or negative power, or ε. A single-step reduction replaces $wa\overline{a}v$ with wv, a reduction is a sequence of single-step reductions. A word in a free monoid Σ^* with involution is *reduced* if no reduction can be performed on it. It is folklore knowledge that for w there exists exactly one reduced v such that w reduces to v; we call such a v the *normal form* of w and denote it by $\mathrm{nf}(w)$; we write $w \approx v$ when $\mathrm{nf}(w) = \mathrm{nf}(v)$. We write $u \sim v$ to denote that $u = u'v'$ and $v = v'u'$ or $\overline{v} = v'u'$ for some u', v'. A reduced word w is *cyclically reduced* if it is not of the form $w = av\overline{a}$ for any $a \in \Sigma$ and w is *primitive* if there is no word v, such that $w = v^k$ for some natural number $k > 1$.

Free group. Formally, the free group (over generators Σ) consists of all reduced words over Σ with the operation $w \cdot v = \mathrm{nf}(wv)$. We use all elements of Σ^* to denote elements of the free group, with w simply denoting $\mathrm{nf}(w)$. Note that in such a setting $\approx$ corresponds to equality in free group. Note that the inverse w^{-1} of w is $\overline{w}$ and we will use this notation, as most of the arguments are given for the monoid and not the free group.

Any equation in the free group is equivalent to an equation in which the right-hand side is ε, as $u \approx v$ is equivalent to $uv^{-1} \approx \varepsilon$, thus in the following we consider only equations in such a form. Moreover, $uv \approx \varepsilon$ is equivalent to $vu \approx \varepsilon$, which can be seen by multiplying by v from the left and v^{-1} from the right; hence we can assume that the equation begins with a variable. Let us fix the equation

$$X^{p_1} u_1 X^{p_2} u_2 \cdots u_{m-1} X^{p_m} u_m \approx \varepsilon \tag{2}$$

for the rest of the paper, each u_i is a reduced word in Σ^*, every p_i is 1 or -1 and there are no expressions $X\varepsilon\overline{X}$ nor $\overline{X}\varepsilon X$ in the equation. Clearly, m is the number of occurrences of the variable X in the equation, let $n = m + \sum_{i=1}^{m} |u_i|$ be the length of the equation. A reduced word $x \in \Sigma^*$ is a solution when $x^{p_1} u_1 x^{p_2} \cdots u_{m-1} x^{p_m} u_m \approx \varepsilon$.

Integer expressions, parametric words. Let us fix two integer variables I, J for the remainder of the paper. An integer expression is of the form $n_I I + n_J J + n_c$, where $n_I, n_J, n_c \in \mathbb{Z}$ are integers; an expression is *constant* when $n_I = n_J = 0$ and *non-constant* otherwise. We denote integer expressions with letters ϕ, ψ, note that all expressions that we consider are in the same two variables I, J. A value $\phi(i, j)$ is defined in a natural way; we also use this notation for substitutions of variables, say $\phi(I, k - I)$, which is defined in a natural way. The integer expression ϕ depends on the variable I (J) if $n_I \neq 0$ ($n_J \neq 0$) and it depends on $I + J$ if $n_I = n_J \neq 0$. If ϕ depends on exactly one variable then we write $\phi(i)$ to denote its value.

An s-parametric power is of the form s^ϕ, where ϕ is an integer expression and s a word; then $s(i, j)$ denotes $s^{\phi(i,j)}$, this can be interpreted both as an element in the monoid and in the free group. Unless explicitly stated, we consider only non-constant expressions ϕ as exponents in parametric powers, this should remove the ambiguity that an s-power is also an s-parametric power. A parametric word is of the form $w = t_0 s_1^{\phi_1} t_1 \cdots t_{k-1} s_k^{\phi_k} t_k$ (all arithmetic expressions $\phi_1, \ldots, \phi_k$ are in the same two variables I, J) and $w(i, j)$ denotes $t_0 s_1^{\phi_1(i,j)} t_1 \cdots t_{k-1} s_k^{\phi_k(i,j)} t_k$. In most cases, we consider very simple parametric words, where

$k \leq 2$ and both expressions depend on one variable only. We sometimes talk about equality of parametric words (in a free group), formally $w \approx w'$ if for each $(i, j) \in \mathbb{Z}^2$ it holds that $w(i, j) \approx w'(i, j)$. We will use those only in very simple cases, say $u^{I+1}u^{-I+1} \approx u^2$.

As we process sets of integer expressions (as well as parametric powers), we will often represent them as sorted lists (with duplicates removed): we can use any linear order, say for integer expressions the lexicographic order on triples (n_I, n_J, n_c) and for parametric powers the lexicographic order on tuples (s, n_I, n_J, n_c), where tuple (s, n_I, n_J, n_c) corresponds to a parametric power $u^{n_I I + n_J J + n_c}$.

2.2 Data structure

Words appearing naturally in our proofs and algorithms are concatenations of a constant number of subwords (or their involutions) of the input equation. We say that a word w is *k-represented*, if w is given as $w = (U\overline{U})[b_1 \mathinner{..} e_1] \cdots (U\overline{U})[b_k \mathinner{..} e_k]$, where $U = u_1 \cdots u_m$ is the concatenation of all words from the equation (2). A parametric word $s_0 t_1^{\phi_1} s_1 \cdots s_{\ell-1} t_\ell^{\phi_\ell} s_\ell$ is k-represented, when $s_0, t_1, s_1, \ldots, t_\ell, s_\ell$ are $k_0, \ldots, k_{2\ell}$ represented and $k = \sum_{i=0}^{2\ell} k_i$.

We use standard data structures, like suffix arrays [17] and structures for answering longest common prefix queries on them [18]. As a result, we can answer all basic queries (like normal form, longest common prefix, power prefix, etc.) about words in the equation in $\mathcal{O}(1)$ time; note that this is the place in which we essentially use that we can perform operations on $\mathcal{O}(\log n)$-size numbers in $\mathcal{O}(1)$ time. As an example of usage, we can test whether a word is a solution in $\mathcal{O}(m)$ time:

▶ **Lemma 2.** *Given a word $\alpha u^i v^j \beta$, where α, β, u, v are $\mathcal{O}(1)$-represented, α, β are reduced and u, v are cyclically reduced and primitive and i, j are a pair of integer numbers, we can test whether $\alpha u^i v^j \beta$ is a solution of (2) in $\mathcal{O}(m)$ time.*

2.3 Superset of solutions

The previous characterization [3] essentially showed that a solution is either a $\mathcal{O}(1)$-represented word or of the form $u^i u' v'' v^j$ for some $i, j \in \mathbb{Z}$ and u' is a prefix of u and v'' is a suffix of v for some well defined u, v. As we intend to analyze those solutions using word combinatorics, it is useful to assume that u, v are cyclically reduced and primitive. Unfortunately, this cannot be extracted directly from the previous characterization, so we repeat the previous arguments taking some extra care.

▶ **Lemma 3** (cf. [12, Lemma 15]). *For a given equation (2), in $\mathcal{O}(n^2)$ time one can compute a superset of solutions of the form*

$$S \cup \bigcup_{(\alpha u^I v^J \beta) \in W} \; \bigcup_{i,j \in \mathbb{Z}} \{\alpha u^{I(i)} v^{J(j)} \beta\}$$

where S is a set of $\mathcal{O}(1)$-represented words with $|S| = \mathcal{O}(n^2)$ and for each $0 \leq i \leq m-1$ there are numbers $\ell_i, \ell'_i \leq |u_i| + |u_{i+1}|$ such that W contains exactly $\ell_i \cdot \ell'_i$ parametric words satisfying

- *α, β, are $\mathcal{O}(1)$-represented, reduced and $|\alpha|, |\beta| \leq |u_i| + |u_{i+1}|$;*
- *u, v are 2-represented, cyclically reduced, primitive and $|u| = \ell_i$ and $|v| = \ell'_i$.*

2.4 Maximal powers

We say that a word s^p is a *maximal power* in a word t, if it is a subword of t and there is no s nor $\overline{s}$ to its left and right in t; note that t need not to be reduced. For instance a^3, a^2 and $(ab)^2$ are maximal powers in $aaababaa$. To streamline the analysis, we assume that s^0 (called the *trivial power*) is a maximal power in any word t, even the empty one.

If s^p is a maximal power in a normal form of concatenation of several words $\mathrm{nf}(w_1 \cdots w_\ell)$, then clearly s^p can be partitioned into ℓ subwords such that the i-th of them comes from w_i. However, we show more: we can identify such a maximal power in each w_i, that s^p is (almost) the normal form of concatenation of those maximal powers. This is beneficial: the number of different maximal powers in a word is much smaller than the number of different powers that are subwords.

▶ **Lemma 4.** *Let $w_1, w_2, \ldots, w_\ell$ be reduced and s be cyclically reduced. If s^k is a maximal power in $\mathrm{nf}(w_1 \cdots w_\ell)$ then for each $1 \leq h \leq \ell$ there exists such a maximal power s^{k_h} in w_h that $|\sum_{h=1}^{\ell} k_h - k| < \ell$. Moreover, if s^k is the s-power prefix (suffix) of $\mathrm{nf}(w_1 \cdots w_\ell)$ then we can choose s^{k_1} as the s-power prefix of w_1 or a trivial power (s^{k_ℓ} as the s-power suffix of w_ℓ or a trivial power, respectively); if $s^k = \mathrm{nf}(w_1 \cdots w_\ell)$ then both conditions hold simultaneously.*

The proof of Lemma 4 in case of $\ell \leq 2$ is a simple case distinction. For larger ℓ, we let $w_{1,2} = \mathrm{nf}(w_1 w_2)$ and apply the induction assumption to $w_{1,2} w_3, \ldots, w_\ell$, the proof again follows by simple combinatorics on words.

There cannot be too many different maximal powers of the same word s in a given word w: different maximal powers $s^{k_1}, \ldots, s^{k_p}$ use together $|s|k_1 + \cdots + |s|k_p$ letters in w and when $k_1, \ldots, k_p$ are pairwise different then this sum is $\Omega(p^2|s|)$ and so $p = \mathcal{O}(\sqrt{|w|/|s|})$; this can be naturally generalized to a set of words W instead of a single word w.

▶ **Lemma 5.** *Let s be cyclically reduced word. Let W be a set of words and $k = \sum_{w \in W} |w|$. Suppose that $s^{k_1}, \ldots, s^{k_p}$ are pairwise disjoint subwords of words in W and that $k_1, \ldots, k_p$ are pairwise different integers. Then $p \leq \sqrt{4k/|s| + 1}$ and if additionally $k \geq |s|$ then $p \leq \sqrt{5k/|s|}$.*

3 Restricting the superset of solutions

By Lemma 3, we know the form of possible solutions, and by Lemma 2 we can test a single candidate solution in $\mathcal{O}(m)$ time. In particular, all solutions from the set S in Lemma 3 can be tested in $\mathcal{O}(n^2 m)$ time, as desired. The other solutions are instances of parametric words the form $\alpha u^I v^J \beta$ for well-defined α, u, v, β. The next step is to bound, for fixed α, u, v, β, the set of values (i, j) such that $\alpha u^I v^J \beta(i, j)$ could be a solution; this is the main result of the paper.

Idea. Suppose we want to find out which words of the form u^i are a solution of (2). We substitute u^I to the equation and treat its left-hand side as a parametric word w depending on I. If substituting $I = i$ leads to a trivial word, then it is known that some u-power cancels within the neighboring u-powers (actually, a variant of this fact was used to characterize the superset of solutions [25, 1, 3], and it is attributed already to Nielsen [28]), more formally:

▶ **Lemma 6** (cf. [3, Lemma 3]). *Let $\varepsilon \approx s_0 u_1 s_1 u_2 \cdots s_{k-1} u_k s_k$. Then there is u_i which reduces within $u_{i-1} s_{i-1} u_i s_i u_{i+1}$.*

We want to use Lemma 6 to claim that some u-parametric powers need to reduce, however, as there can be powers of u as constants, this makes the analysis problematic: as an example, consider an equation $a u^I u^\ell \overline{a} \approx \varepsilon$, if $I = i$ is a solution and we set $s_0 = a, u_1 = u^i, s_1 = u^\ell \overline{a}$ (so that u_1 corresponds to u^I) then Lemma 6 guarantees that u^i cancels within u^ℓ, i.e. $0 \geq i \geq -\ell$, even though $I = -\ell$ is the only solution. This is caused by u-powers next to

u-parametric power, which makes our application of the Lemma 6 nearly useless. To fix this, in $au^iu^\ell\overline{a}$ we set $s_0 = a, u_1 = u^{i+\ell}, s_1 = \overline{a}$, and then Lemma 6 yields $i = -\ell$. On the level of the parametric word this corresponds to considering $au^{I+\ell}\overline{a} \approx au^I u^\ell \overline{a}$, i.e. we include u-powers into the u-parametric power next to them.

This is formalized as follows: A parametric word w is *u-reduced* when u is cyclically reduced, primitive and w does not have a subword of the form:

- u^ϕ for a constant integer expression ϕ;
- $a\overline{a}$ for some letter a (so w is reduced);
- $u^\phi u^\psi$ for some (non-constant) integer expressions ϕ, ψ;
- $uu^\phi, \overline{u}u^\phi, u^\phi u, u^\phi\overline{u}$ for some (non-constant) integer expression ϕ.

Note that we do not forbid subwords that are powers of u, we forbid parametric subwords that are in fact subwords, i.e. have constant exponents.

Given a parametric word w we can u-reduce it to obtain a parametric word that is equal (in the free group) and u-reduced by a simple greedy procedure, i.e. replacing a parametric power with a constant integer expression as exponent with a power or reduction or joining two u-powers into one (the running time for specific applications is analyzed separately at appropriate places). When we replace, say uu^ϕ with $u^{\phi+1}$, then we say that letters in u were u-reduced to $u^{\phi+1}$. Note that there are different u-reduced equivalent parametric words, so the output of u-reduction is not unique, this has no effect on the algorithm, though.

If a parametric word w (with all exponents depending on one variable) is u-reduced then from Lemma 6 we infer that $w(i) \approx \varepsilon$ implies $|\phi(i)| \leq 3$ for some parametric power u^ϕ in w:

► **Lemma 7.** *Let $w = w_0 u^{\phi_1} w_1 \cdots u^{\phi_k} w_k$ be a u-reduced parametric word, where $w_0, \dots, w_k$ are words and $\phi_1, \dots, \phi_k$ are integer expressions, all depending on exactly one and same variable. If $w(i) \approx \varepsilon$ then there is ϕ_ℓ such that $|\phi_\ell(i)| \leq 3$. In particular, $w(i) \approx \varepsilon$ for each i if and only if $w = \varepsilon$.*

As ϕ_ℓ in Lemma 7 is a non-constant integer expression then there are at most 7 values of i such that $|\phi_\ell(i)| \leq 3$. Hence it is enough to find appropriate i values. Clearly, there are at most m integer expressions in w (as this is the number of variables). We can give better estimations, though: if the expression is not of the form kI then it "used" at least $|u|$ letters from the equation. So there are $n/|u|$ different expressions and the ones of the form kI; as $|ki| \leq 3$ implies $|i| \leq 3$, there are $7(1 + n/|u|)$ candidates for i in total. Lastly, when the solution depends on two variables, it can be shown that all obtained parametric powers have coefficient ±1, which allow even better estimations: a parametric power $I + c$ uses at least $c|u|$ letters from the equation and so it can be shown that at most $\mathcal{O}(\sqrt{n/|u|})$ different integer expressions can be formed in such a case.

The actual solution is of the form $\alpha u^I v^J \beta$. Firstly, the presence of α, β make estimations harder, as their letters can also be used in the u- and v-reductions. Secondly, there are two parameters, which makes a simple usage of Lemma 7 impossible. However, if $w(i, j) \approx \varepsilon$ then $w(I, j) \approx \varepsilon$ depends on one variable, so Lemma 7 is applicable to it. The analysis yields that we can restrict the possible value of i or j or (i, j); note that this is non-obvious, as there are infinitely many $w(I, j)$s. A similar analysis can be made for $w(i, J)$, and combining those two yields a set of pairs to be tested as well as $\mathcal{O}(1)$ individual is and js that should be tested separately. But for a fixed i (j) we can substitute it to the equation and use Lemma 7 for J (I, respectively).

3.1 Restricting the set of (i, j)

Fix some $0 \leq i_0 \leq m-1$ and the corresponding u_{i_0}, u_{i_0+1} in the equation (2). Using Lemma 3 we construct a parametric word $\alpha u^I v^J \beta$, with α, u, v, β depending on u_{i_0}, u_{i_0+1} as well as exponents $p_{i_0}, p_{i_0+1}, p_{i_0+2}$. We substitute $X = \alpha u^I v^J \beta$ to the equation (2), obtaining a parametric word on the left-hand side. We are to find values $(i, j) \in \mathbb{Z}^2$ for which the value of the obtained parametric word is equivalent to ε, thus we call such an (i, j) a solution. We want to find a suitable set of pairs (i, j) and test each one individually, using Lemma 2.

The analysis depends on the relation between u and v: i.e. whether $u \in \{v, \overline{v}\}$, $u \not\sim v$ or $u \sim v$. We analyze particulate cases in Sections 3.1.1–3.1.3. The idea is the same in each case, but technical details differ.

3.1.1 $u \not\sim v$

Due to symmetry, we consider the case when $|v| \geq |u|$, note that it could be that $|u| = |v|$. We rotate the left-hand side of the equation so that it begins and ends with a parametric power: we rotate $\alpha u^I v^J \beta w = \varepsilon$ to $v^J \beta w \alpha u^I = \varepsilon$ or $\overline{\beta} \overline{v}^J \overline{u}^I \overline{\alpha} w = \varepsilon$ to $\overline{u}^I \overline{\alpha} w \overline{\beta} \overline{v}^J = \varepsilon$, depending on the form of the equation. The equation after the rotation is equisatisfiable to the previous one.

We call each parametric word beginning with v^J or $\overline{u}^I$ and ending with u^I or $\overline{v}^J$ and no parametric power inside a *fragment*. The parametric word after the rotation is a concatenation of m fragments. We use the name h-th fragment to refer to the one corresponding to u_h (so h-th from the left); let f_h denote the word that is left from h-th fragment after removing the leading and ending parametric power; note that f_h is of one of the forms $\beta u_h \alpha$, $\beta u_h \overline{\beta}$, $\overline{\alpha} u_h \alpha$, $\overline{\alpha} u_h \overline{\beta}$. For u^I we call the preceding α the associated word, the same name is used to β succeeding v^J, $\overline{\alpha}$ succeeding $\overline{u}^I$ and $\overline{\beta}$ preceding $\overline{v}^J$. To simplify, we will call it a word associated with the parametric power.

We now preprocess the equation, by replacing the left-hand side with an equivalent parametric word (i.e. equal according to $\approx$). As a first step, we replace each f_h with $\text{nf}(f_h)$. Next, observe that if w is the power of u then $\overline{u}^I w u^I \approx w$ and similarly $v^J w' \overline{v}^J \approx w'$ for w' being a power of v. In the second step we check each fragment separately, and if possible, replace it as described above. For fragments that remained unchanged in the second step, we use previous names, i.e. if h-th fragment $v^J \,\text{nf}(f_h) u^I$ was not replaced then we still write it as $v^J \,\text{nf}(f_h) u^I$ and call it h-th fragment. A *trivial fragment* is a maximal subword obtained as concatenations of words obtained due to replacements in the second step.

We now perform the u-reduction (note that the v^J is not touched) and afterwards the v-reduction. Let the obtained equation be of the form

$$W \approx \varepsilon \ , \tag{3}$$

where W is a parametric word.

▶ **Lemma 8.** *For $u \not\sim v$ we can perform the u-reduction and v-reduction after the preprocessing in $\mathcal{O}(m)$ time; the obtained parametric word is u-reduced. No two parametric powers are replaced by one during the u-reduction and v-reduction, in particular, for a given parametric power u^ϕ (v^ψ) in (3) the ϕ (ψ) has a coefficient of the variable equal to ± 1 and the only letters that are u-reduced (v-reduced) to this power come either from the associated fragment of u^I or $\overline{u}^I$ (v^J or $\overline{v}^J$) and the letters from the adjacent trivial fragment (assuming that there is an adjacent trivial fragment).*

Note that the claim that no two parametric powers are replaced by one is not obvious – in principle, it could be that after the preprocessing a trivial fragment is a power of u (or v) and then it is wholly u-reduced, which can lead to two adjacent parametric powers of u, which are then replaced with one. However, this cannot happen, as such a trivial fragment is of the form $u^{k_1}v^{k_2}\cdots$ for some $0 < |k_1|, |k_2|, \ldots$ and such a word cannot be a power of u nor v when $u \not\sim v$, as the subgroup generated by u, v is a free group.

We now estimate, how many different u-parametric expressions are there after the reductions. When we want to distinguish between occurrences of parametric powers with the same exponent (say, two occurrences of u^{I+1} counted separately) then we write about parametric powers and when we want to treat it as one, then we talk about exponents. We provide two estimations, one focuses on parametric powers and the other on exponents.

▶ **Lemma 9.** *There is a set S of $\mathcal{O}(1)$ size of integer expressions such that there are $\mathcal{O}(n/|u|)$ occurrences of u-parametric powers in W from* (3) *whose exponents are not in S and $\mathcal{O}(n/|v|)$ occurrences of v-parametric powers whose exponents are not in S. The set S can be computed and the parametric powers identified in $\mathcal{O}(m + n/|u|)$ time.*

The Lemma considers, whether the parametric power used some letters from the trivial fragment or its associated fragment had u_h of length at least $|u|$. If so, then it is in the $\mathcal{O}(n/|u|)$ parametric powers, as one such power uses at least $|u|$ letters of the input equation (this requires some argument for the trivial fragments) and otherwise is can be shown that there are only $\mathcal{O}(1)$ possible exponents: say, when we consider the longest suffix of $\mathrm{nf}(\beta u_h \alpha)$ that is a u-power, where $|u_h| < |u|$, then there is a constant number of possibilities how this suffix is formed (fully within α, within $\mathrm{nf}(u_h\alpha)$, uses some letters of β) and in each case the fact that $|u_h| < |u|$ means that there are only $\mathcal{O}(1)$ different u_hs that can be used; note that we need the primitivity of u here. Concerning the algorithm, note that we can distinguish between these two cases during the preprocessing and mark the appropriate powers.

The next lemma provides a better estimation for the number of different exponents, it essentially uses the fact that all exponents have coefficients at variables ±1: as there are only two possible coefficients, we can focus on the constants. Now, to have a constant $|c|$, we have to use a power u^c from W and to have k *different* constants one has to use k different powers and so from Lemma 5 we conclude that $k = \mathcal{O}(|W|/|u|)$. In general, W can be of quadratic length, as we introduce m copies of α and β into it; the resulting bound is too weak for our purposes. To improve the bound, we employ Lemma 4: consider that when the u-power suffix of, say, $\beta u_h \alpha$, is u^k then by Lemma 4 there are k_α, k_u, k_β such that $|k - k_\alpha - k_u - k_\beta| \leq 2$ and u^{k_u}, u^{k_β} are maximal u-powers in u_h, β and u^{k_α} is the u-power suffix of α. Using Lemma 5, this yields that there are $\mathcal{O}(\sqrt{n/|u|})$ different possible values of k_u (over all u_h), $\mathcal{O}(\sqrt{|\beta|/|u|}) = \mathcal{O}(\sqrt{|u_{i_0}u_{i_0+1}|/|u|})$ of k_β and k_α is fixed, so there are at most $\mathcal{O}(\sqrt{n/|u|} \cdot \sqrt{|u_{i_0}u_{i_0+1}|/|u|}) = \mathcal{O}(\sqrt{n|u_{i_0}u_{i_0+1}|}/|u|)$ possible values of k.

The actual argument is more involved, as it is also possible that the u-parametric power includes letters from the trivial fragments, which requires some extra arguments, nevertheless the general approach is similar.

▶ **Lemma 10.** *After the u-reduction and v-reduction there are $\mathcal{O}(\sqrt{n|u_{i_0}u_{i_0+1}|}/|u|)$ different integer expressions as exponents in parametric powers of u and $\mathcal{O}(\sqrt{n|u_{i_0}u_{i_0+1}|}/|v|)$ of v in the equation. The (sorted) lists of such expressions can be computed in $\mathcal{O}(m + n/|u|)$ and $\mathcal{O}(m + n/|v|)$ time, respectively.*

Concerning the algorithm and its running time, it is enough to list all exponents, remove duplicates, and sort them.

We can use Lemma 6 together with bounds on the number of different exponents in parametric powers from Lemma 10 to limit the possible candidates (i, j) for a solution. However, these bounds are either on i or on j. And as soon as we fix, say, $J = j$ and substitute it to W, the obtained parametric word $W(I, j)$ (or $W(i, J)$) is more complex than W, in particular, we do not have the bounds of Lemma 10 for it, so the set of possible candidates for i for a given $W(I, j)$ is linear, which is too much for the desired running time.

Instead, we analyze (as a mental experiment) $W(I, j)$: Fix $j \in \mathbb{Z}$ such that $W(i, j) \approx \varepsilon$ for some i. Compute $W(I, j)$, u-reduce it, call the resulting parametric word $W_{J=j}$. If $W_{J=j} = \varepsilon$, then clearly for each i the (i, j) is a solution of (3) (and vice-versa, see Lemma 7). It can be shown that in this case for some v^ψ in $W_{J=j}$ it holds that $|\psi(j)| < 6$: at least some two u-parametric powers in W should be merged in $W_{J=j}$, in W they are separated by a v-parametric power, say v^ψ. All letters of $v^{\psi(j)}$ are u-reduced, then standard arguments using periodicity show that $|\psi(j)| < 6$ so we can compute all candidates for such js and test for each one whether indeed $W_{J=j} = \varepsilon$, this is formally stated in Lemma 12.

If $W_{J=j}$ depends on I then from Lemma 7 for some of the (new) u-parametric powers u^ϕ it holds that $|\phi(i)| < 6$. Consider, how this ϕ was created. It could be that it is (almost) unaffected by the second u-reduction and so it is (almost) one of the u-parametric powers in W, see Lemma 13 for precise formulation and sketch of proof, in which case we can use Lemma 10. Intuitively, u^ϕ is affected if the whole two parametric powers in W were used to create u^ϕ. Then it can be shown that some v-parametric power v^ψ from W turned into v-power $v^{\psi(j)}$ satisfies $|\psi(j)| < 6$ and is u-reduced to u^ϕ, the argument is as before, when $W_{J=j} \approx \varepsilon$. Moreover, this occurrence of v^ψ also determines u^ϕ; hence the choice of ψ determines $\mathcal{O}(1)$ candidates for j, uniquely identifies ϕ and i satisfies $|\phi(i)| < 6$, i.e. there are $\mathcal{O}(1)$ candidates for (i, j). Then Lemma 9 is applied to this v^ψ: if it is one of $n/|v|$ occurrences of v-parametric powers then we get $\mathcal{O}(1)$ candidates for (i, j) (for this ψ), so $\mathcal{O}(n/|v|)$ in total, over all choices of such ψ. Otherwise, ψ it is one of $\mathcal{O}(1)$ integer expressions (Lemma 9) and so j is from $\mathcal{O}(1)$-size set and we can compute and consider $W_{J=j}$ for each one of them separately.

A similar analysis applies also to $i \in \mathbb{Z}$ substituted for I. The results are formalized in the Lemma 11 below, its proof is spread across a couple of Lemmata.

▶ **Lemma 11.** *Given equation (3) we can compute in $\mathcal{O}(mn/|u|)$ time sets $S_I, S_J, S_{\mathbb{Z},J} \subseteq \mathbb{Z}$ and $S_{I,J} \subseteq \mathbb{Z}^2$, where $|S_I| = \mathcal{O}(\sqrt{n|u_{i_0}u_{i_0+1}|}/|u|)$, $|S_J| = \mathcal{O}(1)$, $|S_{\mathbb{Z},J}|, |S_{I,J}| = \mathcal{O}(n/|u|)$, such that: if (i, j) is a solution of (3) then at least one of the following holds:*

- *$i \in S_I$ or*
- *$j \in S_J$ or*
- *$j \in S_{\mathbb{Z},J}$ and for each i' the (i', j) is a solution or*
- *$(i, j) \in S_{I,J}$.*

Similarly, given equation (3) we can compute in $\mathcal{O}(mn/|v|)$ time sets $S'_I, S'_J, S'_{I,\mathbb{Z}} \subseteq \mathbb{Z}$ and $S'_{I,J} \subseteq \mathbb{Z}^2$, where $|S'_I| = \mathcal{O}(1)$, $|S'_J| = \mathcal{O}(\sqrt{n|u_{i_0}u_{i_0+1}|}/|v|)$ $|S'_{I,\mathbb{Z}}|, |S'_{I,J}| = \mathcal{O}(n/|v|)$ such that at if (i, j) is a solution of (3) then least one of the following holds:

- *$i \in S'_I$ or;*
- *$i \in S'_{I,\mathbb{Z}}$ and for each $j' \in \mathbb{Z}$ the (i, j') is a solution or;*
- *$j \in S'_J$ or;*
- *$(i, j) \in S'_{I,J}$.*

As noted above, the main distinction is whether the u^ϕ in $W_{J=j}$ was “affected” or not during the second u-reduction. Let us formalize this. Given an occurrence of a parametric power u^ϕ in $W_{J=j}$ consider the largest subword w of W such that each letter in $w(I, j)$ is either reduced or u-reduced to this u^{ϕ}; note that this may depend on the order of reductions, we fix an arbitrary order. We say that parametric powers in w are *merged* to u^{ϕ}. We extend this notion also to the case when $W_{J=j} = \varepsilon$, in which case $W = w$ and every parametric power is merged to the same parametric power u^0. A similar notion is defined also for parametric powers of v. Note that a parametric power is not merged to two different parametric powers u^{ϕ} and $u^{\phi'}$.

We say that a u-parametric power u^{ϕ} in $W_{J=j}$ *was affected* by substitution $J = j$ if

- more than one parametric power was merged to u^{ϕ} *or*
- for the unique u-parametric power $u^{\phi'}$ merged to u^{ϕ} there is a v-parametric power $v^{\psi'}$ such that $|\psi'(j)| < 6$ and there is no u-parametric power between $u^{\phi'}$ and $v^{\psi'}$.

The intuition behind the first condition is that when we merge two u-powers then we create a completely new parametric power, for the second condition, when $|\psi'(j)| < 6$ then $v^{\psi'(j)}$ no longer behaves like $v^{\psi'}$ and can either be wholly merged to a u-power or be canceled by a trivial fragment, which can also lead to a large modification of the neighbouring u-parametric power. Note that the second condition could be made more restrictive, but the current formulation is good enough for our purposes.

We first investigate the case, when the parametric power was affected by a substitution.

▶ **Lemma 12.** *In $\mathcal{O}(mn/|v|)$ time we can compute and sort sets $S_J, S_{E,J}$, where $|S_J| = \mathcal{O}(1)$ and $|S_{E,J}| = \mathcal{O}(n/|v|)$, such that for each occurrence of a u-parametric power u^{ϕ} in $W_{J=j}$ affected by the substitution $J = j$ either $j \in S_J$ or $(\phi, j) \in S_{E,J}$.*

Similarly, in time $\mathcal{O}(mn/|u|)$ we can compute and sort sets $S'_I, S_{I,E}$, where $|S'_I| = \mathcal{O}(1)$ and $|S_{I,E}| = \mathcal{O}(n/|u|)$, such that for each occurrence of a v-parametric power v^{ψ} in $W_{I=i}$ affected by the substitution $I = i$ either $i \in S'_I$ or $(i, \psi) \in S_{I,E}$.

The sketch of the argument was given above Lemma 11. Concerning the running time, the appropriate exponents are identified during the u-reduction and v-reduction, which are performed in given times using the data structure.

We now consider the case when u^{ϕ} was not affected. Essentially, we claim that u^{ϕ} is almost the same as some $u^{\phi'}$ in W. The difference is that it can u-reduce letters from v-parametric powers that become v-powers. However, as such v-power is not wholly merged (as it is not affected), only its proper suffix or prefix can be u-reduced and by primitivity and by case assumption $u \not\sim v$ and $|v| \geq |u|$, this suffix is of length at most $|v| + |u|$. Thus, while in principle there are infinitely many possibilities for $v^{\psi}(j)$ when $j \in \mathbb{Z}$, it is enough to consider a constant number of different candidates (roughly: $\overline{v}^2, \overline{v}, \varepsilon, v, v^2$) and we can procure all of them so that an analysis similar to the one in Lemma 10 can be carried out: essentially we replace a fragment $v^J f_h u^I$ with 5 "fragments" $v^c f_h u^I$ for $c \in \{-2, -1, 0, 1, 2\}$. In this argument, we used the assumption that $|v| \geq |u|$ (the u-reduction is of length at most $|v| + |u| \leq 2|v|$), but it turns out that in the case v-parametric powers the argument is even simpler: the v-reduced prefix of u-parametric power is of length at most $2|v|$, so the v-parametric power is modified by an additive $\mathcal{O}(1)$ summand.

▶ **Lemma 13.** *We can compute and sort in $\mathcal{O}(m + n/|u|)$ time a set of $\mathcal{O}(\sqrt{n|u_{i_0}u_{i_0+1}|}/|u|)$ integer expressions E such that for every j if u^{ϕ} is a parametric power in $W_{J=j}$ not affected by substitution $J = j$ then $\phi \in E$.*

A similar set of $\mathcal{O}(\sqrt{n|u_{i_0}u_{i_0+1}|}/|v|)$ integer expressions can be computed for the not affected v-parametric powers after the second v-reduction in $\mathcal{O}(m + n/|v|)$ time.

The algorithm works by simple grouping of the parametric powers after the u- and v-reductions, the running times are obtained by appropriate usage of the data structure.

Lemmata 9, 10, 12 and 13 are enough to prove Lemma 11, by a simple case distinction, as described in text preceding Lemma 9.

What is left to show is how to compute candidate solutions, when one of I, J, say J, is already fixed, as in the claim of Lemma 11. The analysis is similar as in the case of two parameters, however, we cannot guarantee that after the u-reduction the coefficient at the u-parametric powers are ± 1. On the positive side, as there is only one integer variable, we can apply Lemma 7 directly. The additional logarithmic in the running time is due to sorting, which now cannot be done using counting sort, as the involved numbers may be large.

▶ **Lemma 14.** *For any given j in $\mathcal{O}(m)$ time we can decide, whether for each $i \in \mathbb{Z}$ the $\alpha u^i v^j \beta$ is a solution of* (2) *and if not then in $\mathcal{O}(m + n \log m/|u|)$ time compute a superset (of size $\mathcal{O}(n/|u|)$) of is such that $\alpha u^i v^j \beta$ is a solution.*

A similar claim holds for any fixed i (with superset size $\mathcal{O}(n/|v|)$ and running time $\mathcal{O}(m + n \log m/|v|)$).

3.1.2 $u \in \{v, \overline{v}\}$

When $u \in \{v, \overline{v}\}$ then $u^I v^J \approx u^{I+J}$ or $u^I v^J \approx u^{I-J}$ and we can replace the parameter $I + J$ (or $I - J$) with a single I. This case is subsumed by the case when we fix one of the parameters (i.e. I or J), see Lemma 14.

3.1.3 $u \sim v$

In this case either $u = u'u''$ and $v = u''u'$ or $v = \overline{u'}\,\overline{u''}$, for some u', u''. By substituting $v = \overline{v}$ we reduce the latter case to the former. We consider the parametric solution $\alpha u^I v^J \beta$, note that $v \approx u'' u \overline{u''}$ and so $\alpha u^I v^J \beta \approx \alpha u^I u'' u^J \overline{u''} \beta$. From now on the approach is similar as when $u \not\sim v$. Most of the arguments are simpler, however, the extra technicality is that after the u-reduction we can have u-parametric power of the form $u^{\pm(I+J)+c}$. As a result, we consider not only substitutions $I = i$ and $J = j$, but also $I + J = k$, i.e. we substitute ϕ with $\phi(I, k - I)$, which depends only on I. This requires some additional cases to consider and makes some formulations longer, but everything follows in a similar way.

Overall, the main characterization is

▶ **Lemma 15** (cf. Lemma 11). *Given the equation* (3) *we can compute in $\mathcal{O}(mn/|u|)$ time sets $S_I, S_J, S_{I+J}, S_{I+J,\mathbb{Z}} \subseteq \mathbb{Z}$ and $S_{I,J} \subseteq \mathbb{Z}^2$, where $|S_I|, |S_J| = \mathcal{O}(\sqrt{n|u_{i_0} u_{i_0+1}|}/|u|)$, $|S_{I+J}| = \mathcal{O}(1)$, $|S_{I+J,\mathbb{Z}}|, |S_{I,J}| = \mathcal{O}(n/|u|)$ such that if (i, j) is a solution of* (3) *then at least one of the following holds:*

- *$i \in S_I$ or*
- *$j \in S_J$ or*
- *$i + j \in S_{I+J}$ or*
- *$i + j \in S_{I+J,\mathbb{Z}}$ and for each i' the $(i', (i + j) - i')$ is a solution or*
- *$(i, j) \in S_{I,J}$.*

Similar sets corresponding to substitutions $I = i$ and $J = j$ can be computed in the same time bounds.

The fourth possibility in Lemma 15 means that $W(I, k - I) \approx \varepsilon$, which would yield an infinite family of solutions $\{\alpha u^i v^{k-i} \beta : i \in \mathbb{Z}\}$. Additional combinatorial analysis yields that this cannot happen (and we know this from the earlier characterisation of the solution set).

▶ **Lemma 16.** *Consider a parametric word $\alpha u^I v^J \beta$ for $u \sim v$ and the corresponding $W \neq \varepsilon$ obtained after the substitution of $X = \alpha u^I v^J \beta$, as in* (3)*. Then for every k it holds that $W(I, k - I) \not\approx \varepsilon$.*

3.2 Algorithm and running time

By Lemma 2 one solution of the form $\alpha u^i v^j \beta$, for fixed $\alpha, u, i, v, j, \beta$ which are $\mathcal{O}(1)$-represented, can be tested in $\mathcal{O}(m)$ time. So it is enough to show that there are at most $\mathcal{O}(n^2)$ different candidates tested (we estimate other computation times as well). Lemma 3 yields that there are $\mathcal{O}(n^2)$ candidate solutions (from the set S). Other solutions are obtained in the following way: for two consecutive words u_{i_0}, u_{i_0+1} from the equation we have a family of $\ell_{i_0} \cdot \ell'_{i_0}$ candidates of the form $\alpha u^I v^J \beta$, see Lemma 3, where $\ell_{i_0} = |u|, \ell'_{i_0} = |v|$ and $\ell_{i_0}, \ell'_{i_0} \leq |u_{i_0}| + |u_{i_0+1}|$; by Lemma 3 the total time, over all i_0, spent on computing words α, β, u, v is $\mathcal{O}(n^2)$ time. We will often use the estimation (a similar one hold for ℓ'_{i_0}):

$$\sum_{i_0=1}^{m} \ell_{i_0} \leq \sum_{i_0=1}^{m} |u_{i_0}| + |u_{i_0+1}| \leq 2n \ . \tag{4}$$

Suppose first that $u \not\sim v$, then by Lemma 11 we can compute in time $\mathcal{O}(mn/\ell_{i_0})$ sets S_I, S_J, $S_{J,\mathbb{Z}}$, $S_{I,J}$, where $|S_I| = \mathcal{O}\left(\sqrt{n|u_{i_0}u_{i_0+1}|/l_{i_0}}\right)$, $|S_J| = \mathcal{O}(1)$, $|S_{\mathbb{Z},J}|, |S_{I,J}| = \mathcal{O}(n/\ell_{i_0})$, such that for each solution (i,j) at least one of the following holds:

i1. $i \in S_I$ or

i2. $j \in S_J$ or

i3. $j \in S_{\mathbb{Z},J}$ and (i',j) is a solution for each i' or

i4. $(i,j) \in S_{I,J}$.

and in time $\mathcal{O}(mn/\ell'_{i_0})$ sets S'_J, S'_I, $S'_{I,\mathbb{Z}}$, $S'_{I,J}$, where $|S'_I| = \mathcal{O}(1)$, $|S'_J| = \mathcal{O}(\sqrt{n|u_{i_0}u_{i_0+1}|}/\ell'_{i_0})$ $|S'_{I,\mathbb{Z}}|, |S'_{I,J}| = \mathcal{O}(n/\ell'_{i_0})$, such that for each solution (i,j) at least one of the following holds:

j1. $j \in S'_J$ or

j2. $i \in S'_I$ or

j3. $i \in S_{I,\mathbb{Z}}$ and (i,j') is a solution for each j' or

j4. $(i,j) \in S'_{I,J}$.

As both of those characterization hold, we should describe how do we treat each of the 16 cases. Fortunately, for most of the cases the further action and analysis depends on one of the cases alone.

If we are in the case i4 or j4 then we test each pair $(i,j) \in S_{I,J} \cup S'_{I,J}$ separately. There are (over all $0 \leq i_0 \leq m-1$) at most (note that some of those solutions have $u \sim v$, we will estimate their running time separately, so now we overestimate the running time)

$$\sum_{i_0=0}^{m-1} \ell_{i_0}\ell'_{i_0}\left(\frac{n}{\ell_{i_0}} + \frac{n}{\ell'_{i_0}}\right) = n \sum_{i_0=0}^{m-1} \ell'_{i_0} + \ell_{i_0} \leq 4n^2 \qquad \text{by (4)} \tag{5}$$

such solutions.

Concerning the time of establishing those sets, the largest is from Lemma 12 and it is $\mathcal{O}(mn/\ell_{i_0})$ (for $S_{I,J}$) or $\mathcal{O}(mn/\ell'_{i_0})$ (for $S'_{I,J}$). So up to a constant it is:

$$\sum_{i_0=0}^{m-1} \ell_{i_0}\ell'_{i_0}\left(\frac{mn}{\ell_{i_0}} + \frac{mn}{\ell'_{i_0}}\right) = mn \sum_{i_0=0}^{m-1} (\ell_{i_0} + \ell'_{i_0}) \leq 2mn^2 \qquad \text{by (4)} \ .$$

If we are in the case i3 then for each $j \in S_{J,\mathbb{Z}}$ we substitute $J = j$ and test, whether $W(I,j) \approx \varepsilon$; by Lemma 7 this is equivalent to (i',j) being a solution for each $i' \in \mathbb{Z}$. Each such j yields a family of solutions of the required form $\{\underbrace{\alpha}_{\text{fixed}} u^i \underbrace{v^j\beta}_{\text{fixed}} : i \in \mathbb{Z}\}$ and there are at most $|S_{J,\mathbb{Z}}| = \mathcal{O}(n/\ell_{i_0})$ such families. Over all $0 \leq i_0 \leq m-1$ this yields at most (up to a constant)

$$\sum_{i_0=0}^{m-1} \ell_{i_0}\ell'_{i_0}\frac{n}{\ell'_{i_0}} = n \sum_{i_0=0}^{m-1} \ell_{i_0} \leq 2n^2 \qquad \text{by (4)} \ .$$

Concerning the running time, note that testing whether $W(I,j) \approx \varepsilon$ takes $\mathcal{O}(m)$ time, see Lemma 14, so it is enough to show that we test $\mathcal{O}(n^2)$ such js. As $|S_{\mathbb{Z},J}| = \mathcal{O}(n/\ell_{i_0})$, the calculations are as in (5). A similar analysis applies to $S_{I,\mathbb{Z}}$, i.e. case j3.

If we are in case i2 then for each $j \in S_J$ we can compute, by Lemma 14, in time $\mathcal{O}(m)$ whether each (i',j) is a solution, note that the set is of the required form, as in the case of $j \in S_{\mathbb{Z},J}$, moreover the estimation on the number of such solution sets is not larger than in the case of $j \in S_{\mathbb{Z},J}$, as $|S_J| = \mathcal{O}(1)$ and $|S_{\mathbb{Z},J}| = \mathcal{O}(n/\ell_{i_0})$. Otherwise, again by Lemma 14, we compute in time $\mathcal{O}(m + n/\ell_{i_0} \log m))$ a set S of size $|S| = \mathcal{O}(n/\ell_{i_0})$ such that if (i,j) is a solution then $i \in S$. This yields $|S_J| \times |S| = \mathcal{O}(n/\ell_{i_0})$ candidate pairs, which are individually tested, so the running time is $\mathcal{O}(mn/\ell_i)$, note that this dominates $\mathcal{O}(m + n \log m/\ell_{i_0})$ from Lemma 14. The estimation in (5) yields that there are at most $\mathcal{O}(n^2)$ such candidate pairs and the whole running time is $\mathcal{O}(mn^2)$. A similar analysis applies to $i \in S'_I$.

The only remaining option is that we are simultaneously in case i1 and j1, i.e. $i \in S_I$ and $j \in S'_J$. As $|S_I| = \mathcal{O}\left(\sqrt{n|u_{i_0}u_{i_0+1}|}/\ell_i\right)$ and $|S'_J| = \mathcal{O}\left(\sqrt{n|u_{i_0}u_{i_0+1}|}/\ell'_i\right)$ there are (over all i_0 and up to a constant) at most

$$\sum_{i_0=1}^{m} \ell_{i_0}\ell'_{i_0} \frac{\sqrt{n|u_{i_0}u_{i_0+1}|}}{\ell_{i_0}} \cdot \frac{\sqrt{n|u_{i_0}u_{i_0+1}|}}{\ell'_{i_0}} = \sum_{i_0=1}^{m} n|u_{i_0}u_{i_0+1}| \leq 2n^2$$

such solutions tested.

The cases of $u = v$ or $u = \overline{v}$ are done using Lemma 14, the bounds are the same as in case of $u \not\sim v$.

The case of $u \sim v$ is a bit more involved, let $\ell_{i_0} = |u|$. By Lemma 15 we can compute in time $\mathcal{O}(mn/|u|)$ sets S_I, S_J, S_{I+J}, $S_{I+J,\mathbb{Z}}$, $S_{I,J}$ and such that for each solution (i,j) either

1. $i \in S_I$ or
2. $j \in S_J$ or
3. $i + j \in S_{I+J}$ or
4. $i + j \in S_{I+J,\mathbb{Z}}$ and for each i' the $(i', (i+j) - i')$ is a solution or
5. $(i,j) \in S_{I,J}$.

The third case is dealt with as previously (for each $i+j$ we make a substitution $I + J = i + j$, check, whether the obtained equation is trivial and solve the corresponding equation), similarly fourth (for each $i + j$ we substitute $I + J = i + j$ and check whether the obtained word is ε; note that it can be shown that this never holds, see Lemma 16) and fifth (we substitute $I = i, J = j$ and test). So we are left only with the first two cases. Moreover, Lemma 15 also gives us a similar characterization resulting from a substitution $I = i$, again there are 5 cases and the last three of them are dealt with similarly, the first two give that there are sets S'_J, S'_{I+J} such that

1. $j \in S'_J$ or
2. $i + j \in S'_{I+J}$

and applied to substitution $J = j$ again gives 5 cases, the last three of which are dealt with and the first two yield that there are sets S''_I, S''_{I+J} such that

1. $i \in S''_I$ or
2. $i + j \in S''_{I+J}$.

There are in total 8 cases (we choose one of two options for three substitutions), in each such a case from the three choices some two (though not each two) allow to give $\mathcal{O}\left(n|u_{i_0}u_{i_0+1}|/\ell^2_{i_0}\right)$ candidates for (i,j) : say if $i \in S_I$, $j \in S'_J$ and $i + j \in S''_{I+J}$ then any two determine (i,j) and when $j \in S_J$, $j \in S'_J$ and $i + j \in S''_{I+J}$ then $j \in S_J \cap S'_J$ and $i + j \in S''_{I+J}$. The rest of the calculations is the same as in the case of $u \not\sim v$.

References

1 Kenneth I. Appel. One-variable equations in free groups. *Proceedings of the American Mathematical Society*, 19:912–918, 1968.

2 Kenneth I. Appel. On two variable equations in free groups. *Proceedings of the American Mathematical Society*, 21:179–184, 1969.

3 Dimitri Bormotov, Robert Gilman, and Alexei Myasnikov. Solving one-variable equations in free groups. *Journal of Group Theory*, 12:317–330, 2009. doi:10.1515/JGT.2008.080.

4 Witold Charatonik and Leszek Pacholski. Word equations with two variables. In *IWWERT*, pages 43–56, 1991. doi:10.1007/3-540-56730-5_30.

5 Ian M. Chiswell and Vladimir N. Remeslennikov. Equations in free groups with one variable. I. *Journal of Group Theory*, 3(4), 2000. doi:10.1515/jgth.2000.035.

6 Joel D. Day, Vijay Ganesh, Paul He, Florin Manea, and Dirk Nowotka. The satisfiability of word equations: Decidable and undecidable theories. In Igor Potapov and Pierre-Alain Reynier, editors, *Reachability Problems - 12th International Conference, RP 2018, Marseille, France, September 24-26, 2018, Proceedings*, volume 11123 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2018. doi:10.1007/978-3-030-00250-3_2.

7 Volker Diekert. Makanin's algorithm. In M. Lothaire, editor, *Algebraic Combinatorics on Words*, chapter 12, pages 342–390. Cambridge University Press, 2002.

8 Volker Diekert, Claudio Gutiérrez, and Christian Hagenah. The existential theory of equations with rational constraints in free groups is PSPACE-complete. *Inf. Comput.*, 202(2):105–140, 2005. doi:10.1016/j.ic.2005.04.002.

9 Volker Diekert, Artur Jeż, and Wojciech Plandowski. Finding all solutions of equations in free groups and monoids with involution. *Inf. Comput.*, 251:263–286, 2016. doi:10.1016/j.ic.2016.09.009.

10 Robert Dąbrowski and Wojciech Plandowski. Solving two-variable word equations. In *ICALP*, pages 408–419, 2004. doi:10.1007/978-3-540-27836-8_36.

11 Robert Dąbrowski and Wojciech Plandowski. On word equations in one variable. *Algorithmica*, 60(4):819–828, 2011. doi:10.1007/s00453-009-9375-3.

12 Robert H. Gilman and Alexei G. Myasnikov. One variable equations in free groups via context free languages. *Contemporary Mathematics*, 349:83–88, 2004.

13 Yu. I. Hmelevskiĭ. *Equations in Free Semigroups.* Number 107 in Proceedings Steklov Institute of Mathematics. American Mathematical Society, 1976. Translated from the Russian original: Trudy Mat. Inst. Steklov. 107, 1971.

14 Lucian Ilie and Wojciech Plandowski. Two-variable word equations. *RAIRO Theor. Informatics Appl.*, 34(6):467–501, 2000. doi:10.1051/ita:2000126.

15 Artur Jeż. One-variable word equations in linear time. *Algorithmica*, 74:1–48, 2016. doi:10.1007/s00453-014-9931-3.

16 Artur Jeż. Recompression: a simple and powerful technique for word equations. *J. ACM*, 63(1):4:1–4:51, March 2016. doi:10.1145/2743014.

17 Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *J. ACM*, 53(6):918–936, 2006. doi:10.1145/1217856.1217858.

18 Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *CPM*, pages 181–192, 2001. doi:10.1007/3-540-48194-X_17.

19 Olga Kharlampovich, Igor G. Lysënok, Alexei G. Myasnikov, and Nicholas W. M. Touikan. The solvability problem for quadratic equations over free groups is NP-complete. *Theory of Computing Systems*, 47(1):250–258, 2010. doi:10.1007/s00224-008-9153-7.

20 Olga Kharlampovich and Alexei Myasnikov. Elementary theory of free non-abelian groups. *Journal of Algebra*, 302:451–552, 2006.

21 Antoni Kościelski and Leszek Pacholski. Makanin's algorithm is not primitive recursive. *Theor. Comput. Sci.*, 191(1-2):145–156, 1998. doi:10.1016/S0304-3975(96)00321-0.

22 Markku Laine and Wojciech Plandowski. Word equations with one unknown. *Int. J. Found. Comput. Sci.*, 22(2):345–375, 2011. doi:10.1142/S0129054111008088.

23 A. A. Lorents. Representations of sets of solutions of systems of equations with one unknown in a free group. *Dokl. Akad. Nauk. SSSR*, 178:290–292, 1968.

24 Roger C. Lyndon and Marcel-Paul Schützenberger. The equation $a^M = b^N c^P$ in a free group. *Michigan Mathematical Journal*, 9(4):289–298, 1962.

25 Roger Conant Lyndon. Equations in free groups. *Transansaction of American Mathematical Society*, 96:445–457, 1960. doi:10.1090/S0002-9947-1960-0151503-8.

26 Gennadií Makanin. The problem of solvability of equations in a free semigroup. *Matematicheskii Sbornik*, 2(103):147–236, 1977. (in Russian).

27 Gennadií Makanin. Equations in a free group. *Izv. Akad. Nauk SSR*, Ser. Math. 46:1199–1273, 1983. English transl. in Math. USSR Izv. 21 (1983).

28 Jakob Nielsen. Über die Isomorphismen unendlicher Gruppen ohne Relation. *Mathematische Annalen*, 79:269–272, 1918.

29 Dirk Nowotka and Aleksi Saarela. An optimal bound on the solution sets of one-variable word equations and its consequences. 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 136:1–136:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.136.

30 Seraphin D. Eyono Obono, Pavel Goralčik, and Marianne Maksimenko. Efficient solving of the word equations in one variable. In *MFCS*, pages 336–341, 1994. doi:10.1007/3-540-58338-6_80.

31 Wojciech Plandowski. Satisfiability of word equations with constants is in PSPACE. *J. ACM*, 51(3):483–496, 2004. doi:10.1145/990308.990312.

32 Alexander A. Razborov. *On Systems of Equations in Free Groups*. PhD thesis, Steklov Institute of Mathematics, 1987. In Russian.

33 Zlil Sela. Diophantine geometry over groups VI: the elementary theory of a free group. *Geometric & Functional Analysis*, 16:707–730, 2006. doi:10.1007/s00039-006-0565-8.

Diverse Collections in Matroids and Graphs

Fedor V. Fomin
University of Bergen, Norway

Petr A. Golovach
University of Bergen, Norway

Fahad Panolan
Department of Computer Science and Engineering, IIT Hyderabad, India

Geevarghese Philip
Chennai Mathematical Institute, India
UMI ReLaX, Chennai, India

Saket Saurabh
Institute of Mathematical Sciences, Chennai, India
University of Bergen, Norway

Abstract

We investigate the parameterized complexity of finding diverse sets of solutions to three fundamental combinatorial problems, two from the theory of matroids and the third from graph theory. The input to the Weighted Diverse Bases problem consists of a matroid M, a weight function $\omega : E(M) \to \mathbb{N}$, and integers $k \geq 1, d \geq 0$. The task is to decide if there is a collection of k *bases* $B_1, \ldots, B_k$ of M such that the weight of the symmetric difference of any pair of these bases is at least d. This is a diverse variant of the classical matroid base packing problem. The input to the Weighted Diverse Common Independent Sets problem consists of two matroids M_1, M_2 defined on the same ground set E, a weight function $\omega : E \to \mathbb{N}$, and integers $k \geq 1, d \geq 0$. The task is to decide if there is a collection of k *common independent sets* $I_1, \ldots, I_k$ of M_1 and M_2 such that the weight of the symmetric difference of any pair of these sets is at least d. This is motivated by the classical weighted matroid intersection problem. The input to the Diverse Perfect Matchings problem consists of a graph G and integers $k \geq 1, d \geq 0$. The task is to decide if G contains k *perfect matchings* $M_1, \ldots, M_k$ such that the symmetric difference of any two of these matchings is at least d.

The underlying problem of finding *one* solution (basis, common independent set, or perfect matching) is known to be doable in polynomial time for each of these problems, and Diverse Perfect Matchings is known to be NP-hard for $k = 2$. We show that Weighted Diverse Bases and Weighted Diverse Common Independent Sets are both NP-hard. We show also that Diverse Perfect Matchings cannot be solved in polynomial time (unless P = NP) even for the case $d = 1$. We derive fixed-parameter tractable (FPT) algorithms for all three problems with (k, d) as the parameter.

The above results on matroids are derived under the assumption that the input matroids are given as *independence oracles*. For Weighted Diverse Bases we present a polynomial-time algorithm that takes a representation of the input matroid over a finite field and computes a *poly*(k, d)-sized kernel for the problem.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Matroids, Diverse solutions, Fixed-parameter tractable algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.31

Related Version *Full Version*: `https://arxiv.org/abs/2101.04633`

Funding *Fedor V. Fomin*: Supported by the Research Council of Norway via the project "MULTIVAL" (grant no. 263317).
Petr A. Golovach: Supported by the Research Council of Norway via the project "MULTIVAL" (grant no. 263317).

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 31; pp. 31:1–31:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Fahad Panolan: Seed grant, IIT Hyderabad (SG/IITH/F224/2020-21/SG-79).
Saket Saurabh: European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant no. 819416), and Swarnajayanti Fellowship grant DST/SJF/MSA-01/2017-18.

1 Introduction

In this work we study the parameterized complexity of finding *diverse collections of solutions* to three basic algorithmic problems. Two of these problems arise in the theory of matroids. The third problem belongs to the domain of graph theory, and its restriction to bipartite graphs can be rephrased as a question about matroids. Each of these is a fundamental algorithmic problem in its respective domain.

Diverse FPT Algorithms

Nearly every existing approach to solving algorithmic problems focuses on finding *one solution of good quality* for a given input. For algorithmic problems which are – eventually – motivated by problems from the real world, finding "one good solution" may not be of much use for practitioners of the real-world discipline from which the problem was originally drawn. This is primarily because the process of abstracting out a "nice" algorithmic problem from a "messy" real-world problem invariably involves throwing out a lot of "side information" which is very relevant to the real-world problem, but is inconvenient, difficult, or even impossible to model mathematically.

The other extreme of enumerating *all* (or even all minimal or maximal) solutions to an input instance is also usually not a viable solution. A third approach is to look for *a few solutions of good quality* which are "far away" from one another according to an appropriate notion of distance. The intuition is that given such a collection of "diverse" solutions, an end-user can choose one of the solutions by factoring in the "side information" which is absent from the algorithmic model.

These and other considerations led Fellows to propose *the Diverse X Paradigm* [9]. Here "X" is a placeholder for an optimization problem, and the goal is to study the fixed-parameter tractability of finding a diverse collection of good-quality solutions for X. Recall that the *Hamming distance* of two sets is the size of their symmetric difference. A natural measure of diversity for problems whose solutions are subsets of some kind is the *minimum* Hamming distance of any pair of solutions. In this work we study the parameterized complexity of finding diverse collections of solutions for three fundamental problems with this diversity measure and its weighted variant.

Our problems

Let M be a matroid on ground set $E(M)$ and with rank function $r()$. The departure point of our work is the classical theorem of Edmonds from 1965 [6] about matroid partition. This theorem states that a matroid M has k *pairwise disjoint* bases if and only if, for every subset X of $E(M)$,

$$k \cdot r(X) + |E(M) - X| \geq k \cdot r(M).$$

An important algorithmic consequence of this result is that given access to an independence oracle for a matroid M, one can find a maximum number of *pairwise disjoint bases* of M in polynomial time (See, e.g., [18, Theorem 42.5]). This in turn implies, for instance, that the maximum number of pairwise edge-disjoint spanning trees of a connected graph can be found in polynomial time.

We take a fresh look at this fundamental result of Edmonds: what happens if we don't insist that the bases be pairwise disjoint, and instead allow them to have some pairwise intersection? We work in the weighted setting where each element e of the ground set $E(M)$ has a positive integral weight $\omega(e)$ associated with it, and the weight of a subset X of $E(M)$ is the sum of the weights of the elements in X. The relaxed version of the pairwise disjoint bases problem is then: Given an independence oracle for a matroid M and integers k, d as input, find if M has k bases $B_1, \ldots, B_k$ such that for every pair of bases B_i, B_j ; $i \neq j$ the weight $\omega(B_i \triangle B_j)$ of their symmetric difference is at least d. We call this the Weighted Diverse Bases problem:

Weighted Diverse Bases

Input: A matroid M, a weight function $\omega\colon E(M) \to \mathbb{N}$, and integers $k \geq 1$ and $d \geq 0$.

Task: Decide whether there are bases $B_1, \ldots, B_k$ of M such that $\omega(B_i \triangle B_j) \geq d$ holds for all distinct $i, j \in \{1, \ldots, k\}$.

Due to the expressive power of matroids Weighted Diverse Bases captures many interesting computational problems. We list a few examples; in each case the weight function assigns positive integral weights, $k \geq 1$ and $d \geq 0$ are integers, and we say that a collection of objects is *diverse* if the weight of the symmetric difference of each pair of objects in the collection is at least d. When M is a graphic matroid Weighted Diverse Bases corresponds to finding diverse *spanning trees* in an edge-weighted graph. When M is a vector matroid then this is the problem of finding diverse *column (or row) bases* of a matrix with column (or row) weights. And when M is a transversal matroid on a weighted ground set then this problem corresponds to finding diverse *systems of distinct representatives.*

Another celebrated result of Edmonds is the *Matroid Intersection Theorem* [7] which states that if M_1, M_2 are matroids on a common ground set E and with rank functions r_1, r_2, respectively, then the size of a largest subset of E which is independent in both M_1 and M_2 (a *common independent set*) is given by

$$\min_{T \subseteq E}(r_1(T) + r_2(E - T)).$$

Edmonds showed that given access to independence oracles for M_1 and M_2, a maximum-size common independent set of M_1 and M_2 can be found in polynomial time [7]. This is called the Matroid Intersection problem. Frank [12] found a polynomial-time algorithm for the more general Weighted Matroid Intersection problem where the input has an additional weight function $\omega : E \to \mathbb{N}$ and the goal is to find a common independent set of the maximum *weight.* The second problem that we address in this work is a "diverse" take on Weighted Matroid Intersection where we replace the maximality requirement on individual sets with a lower bound on the weight of their symmetric difference. Given M_1, M_2, ω as above and integers k, d, we ask if there are k common independent sets whose pairwise symmetric differences have weight at least d each; this is the Weighted Diverse Common Independent Sets problem.

Weighted Diverse Common Independent Sets

Input: Matroids M_1 and M_2 with a common ground set E, a weight function $\omega\colon E \to \mathbb{N}$, and integers $k \geq 1$ and $d \geq 0$.

Task: Decide whether there are sets $I_1, \ldots, I_k \subseteq E$ such that I_i is independent in both M_1 and M_2 for every $i \in \{1, \ldots, k\}$ and $\omega(I_i \triangle I_j) \geq d$ for all distinct $i, j \in \{1, \ldots, k\}$.

Weighted Diverse Common Independent Sets also captures many interesting algorithmic problems. We give a few examples (*cf.* [18, Section 41.1a]). We use "diverse" here in the sense defined above. Given a bipartite graph G with edge weights, Weighted Diverse Common Independent Sets can be used to ask if there is a diverse collection of k *matchings* in G. A *partial orientation* of an undirected graph G is a directed graph obtained by (i) assigning directions to some subset of edges of G and (ii) deleting the remaining edges. Given an undirected graph $G = (V, E)$ with edge weights and a function $\iota : V \to \mathbb{N}$, we say that a partial orientation $\mathcal{O}$ of G *respects* ι if the in-degree of every vertex v in $\mathcal{O}$ is at most $\iota(v)$. We can use Weighted Diverse Common Independent Sets to ask if there is a diverse collection of k partial orientations of G, all of which respect ι. For a third example, let $G = (V, E)$ be an undirected graph with edge weights, in which each edge is assigned a – not necessarily distinct – *color*. A *colorful forest* in G is any subgraph of G which is a forest in which no two edges have the same color. We can use Weighted Diverse Common Independent Sets to ask if there is a diverse collection of k colorful forests in G.

Finding whether a bipartite graph has a perfect matching or not is a well-known application of Matroid Intersection ([18, Section 41.1a]). The third problem that we study in this work is a diverse version of the former problem, extended to general graphs. Note that there is no known interpretation of the problem of finding perfect matchings in (general) undirected graphs in terms of Matroid Intersection.

Diverse Perfect Matchings

Input: An undirected graph G on n vertices, and integers $k \geq 1$ and $d \geq 0$.

Task: Decide whether there are perfect matchings $M_1, \ldots, M_k$ of G such that $|M_i \bigtriangleup M_j| \geq d$ for all distinct $i, j \in \{1, \ldots, k\}$.

Our results

We assume throughout that matroids in the input are given in terms of an *independence oracle*. Recall that with this assumption, we can find *one* basis of the largest weight and *one* common independent set (of two matroids) of the largest weight, both in polynomial time. In contrast, we show that the diverse versions Weighted Diverse Bases and Weighted Diverse Common Independent Sets are both NP-hard, even when the weights are expressed in unary[1].

▶ **Theorem 1.** *Both Weighted Diverse Bases and Weighted Diverse Common Independent Sets are strongly* NP*-complete, even on the uniform matroids* U_n^3.

Given this hardness, we analyze the parameterized complexity of these problems with d, k as the parameters. Our first result is that Weighted Diverse Bases is fixed-parameter tractable (FPT) under this parameterization:

▶ **Theorem 2.** *Weighted Diverse Bases can be solved in* $2^{\mathcal{O}(dk^2(\log k + \log d))} \cdot |E(M)|^{\mathcal{O}(1)}$ *time.*

We have a stronger result if the input matroid is given as a representation over a finite field (and not just as a "black box" independence oracle): in this case we show that Weighted Diverse Bases admits a *polynomial kernel* with this parameterization.

[1] See Theorem 7 for an alternative hardness result for Weighted Diverse Bases.

▶ **Theorem 3.** *Given a representation of the matroid M over a finite field GF(q) as input, we can compute a kernel of* Weighted Diverse Bases *of size $\mathcal{O}(k^6 d^4 \log q)$.*

We then show that our second matroid-related diverse problem is also FPT under the same parameterization.

▶ **Theorem 4.** Weighted Diverse Common Independent Sets *can be solved in time $2^{\mathcal{O}(k^3 d^2 \log(kd))} \cdot |E|^{\mathcal{O}(1)}$.*

We now turn to the problem of finding diverse perfect matchings. Diverse Perfect Matchings is known to be NP-hard already when $k = 2$ and G is a 3-regular graph [16, 10]. Since all perfect matchings of a graph have the same size the symmetric difference of two distinct perfect matchings is at least 2. Setting $d = 1$ in Diverse Perfect Matchings is thus equivalent to asking whether G has at least k distinct perfect matchings. Since a bipartite graph on n vertices has at most $\frac{n}{2}!$ perfect matchings and since $\log(\frac{n}{2}!) = \mathcal{O}(n \log n)$ we get – using binary search – that there is a polynomial-time Turing reduction from the problem of *counting* the number of perfect matchings in a bipartite graph to Diverse Perfect Matchings instances with $d = 1$. Since the former problem is #P-complete [19] we get

▶ **Theorem 5.** Diverse Perfect Matchings *with $d = 1$ cannot be solved in time polynomial in $n = |V(G)|$ even when graph G is bipartite, unless* P = NP*.*

Thus we get that Diverse Perfect Matchings is unlikely to have a polynomial-time algorithm even if *one* of the two numbers k, d is a small constant. We show that the problem *does* have a (randomized) polynomial-time algorithm when *both* these parameters are bounded; Diverse Perfect Matchings is (randomized) FPT with k and d as parameters:

▶ **Theorem 6.** *There is an algorithm that given an instance of* Diverse Perfect Matchings*, runs in time $2^{2^{\mathcal{O}(kd)}} n^{\mathcal{O}(1)}$ and outputs the following: If the input is a* No*-instance then the algorithm outputs* No*. Otherwise the algorithm outputs* Yes *with probability at least $1 - \frac{1}{e}$.*

Note that Theorem 6 implies, in particular, that Diverse Perfect Matchings can be solved in (randomized) polynomial time when $kd \leq c_1 + \frac{\log \log n}{c_2}$ holds for some constants c_1, c_2 which depend on the constant hidden by the $\mathcal{O}()$ notation.

Our methods

We prove the NP-hardness results (Theorem 1) by reduction from the 3-Partition problem. To show that Weighted Diverse Bases is FPT (Theorem 2) we observe first that if the input matroid M contains a set of size $\Omega(kd)$ which is *both* independent *and* co-independent in M then the input is a Yes instance of Weighted Diverse Bases (Lemma 14). We can check for the existence of such a set in time polynomial in $|E(M)|$, so we assume without loss of generality that no such set exists. We then show that starting with an arbitrary basis of M and repeatedly applying the greedy algorithm (Proposition 9) *poly*(k, d)-many times we can find, in time polynomial in $(|E(M)| + k + d)$, (i) a subset $S^* \subseteq E(M)$ of size *poly*(k, d) and (ii) a matroid $\widetilde{M}$ on the ground set S^* such that $(\widetilde{M}, \omega, k, d)$ is *equivalent* to the input instance (M, ω, k, d) (Lemma 15). We also show how to compute a useful partition of $E(\widetilde{M}) = S^*$ which speeds up the subsequent FPT-time search for a diverse set of bases in $\widetilde{M}$. The kernelization result for Weighted Diverse Bases (Theorem 3) follows directly from Lemma 15. This “compression lemma” is thus the main technical component of our algorithms for Weighted Diverse Bases.

To show that Weighted Diverse Common Independent Sets is FPT (Theorem 4) we observe first that if the two input matroids M_1, M_2 have a *common* independent set of size $\Omega(kd)$ then the input is a Yes instance of Weighted Diverse Common Independent Sets (Lemma 16). So we assume that this is not the case, and then show (Lemma 17) that we can construct, in $f(k,d)$ time, a collection $\mathcal{F}$ of common independent sets of M_1 and M_2 of size $g(k,d)$ such that if the input is a Yes-instance then it has a solution $I_1, \ldots, I_k$ with $I_i \in \mathcal{F}$ for $i \in \{1, \ldots, k\}$. The FPT algorithm for Weighted Diverse Common Independent Sets follows by a simple search in the collection $\mathcal{F}$.

Our algorithm for Diverse Perfect Matchings is based on two procedures.

P1 Given an undirected graph G on n vertices, perfect matchings $M_1, \ldots, M_r$ of G, and a non-negative integer s as input, this procedure (Lemma 18) runs in time $2^{\mathcal{O}(rs)}n^{\mathcal{O}(1)}$ and outputs a perfect matching M of G such that $|M \triangle M_i| \geq 2s$ holds for all $i \in \{1, \ldots, r\}$ (if such a matching exists), with probability at least $\frac{2}{3}e^{-rs}$.

P2 Given an undirected graph G on n vertices, a perfect matching M of G, and non-negative integers r, d, s, this procedure(Lemma 19) runs in time $2^{\mathcal{O}(r^2 s)}n^{\mathcal{O}(1)}$, and outputs r perfect matchings $M_1^\star, \ldots, M_r^\star$ of G such that $|M \triangle M_i^\star| \leq s$ holds for all $i \in \{1, \ldots, r\}$ and $|M_i^\star \triangle M_j^\star| \geq d$ holds for all distinct $i, j \in [r]$ (if such matchings exist), with probability at least e^{-rs}. If no such perfect matchings exist, then the algorithm outputs No.

Let (G, k, d) be the input instance of Diverse Perfect Matchings. We use procedure **P1** to greedily compute a collection of matchings which are "far apart": We start with an arbitrary perfect matching M_1. In step i, we have a collection of perfect matchings $M_1, \ldots, M_{i-1}$ such that $|M_j \triangle M_{j'}| \geq 2^{k-i}d$ holds for any two distinct $j, j' \in \{1, \ldots, i-1\}$. We now run procedure **P1** with $r = i-1$ and $s = 2^{k-i}d$ to find – if it exists – a matching M_i such that $|M_i \triangle M_j| \geq 2^{k-i+1}d$ holds for all $j \in \{1, \ldots, i\}$. By exhaustively applying **P1** we get a collection of perfect matchings $M_1, \ldots, M_q$ such that

(a) for any two distinct integers $i, j \in \{1, \ldots, q\}$, $|M_i \triangle M_j| \geq 2^{k-q+1}d$, and
(b) for any other perfect matching $M \notin \{M_1, \ldots, M_q\}$, $|M \triangle M_j| \leq 2^{k-q}d$.

Thus, if $k \leq q$, then clearly $\{M_1, \ldots, M_k\}$ is a solution. Otherwise, let $\mathcal{M} = \{M_1^\star, \ldots, M_k^\star\}$ be a hypothetical solution. Then for each $M_i^\star$ there is a *unique* matching M_j in $\{M_1, \ldots, M_q\}$ such that $|M_j \triangle M_i^\star| < 2^{(k-q)}d$ holds (Claim 20). For each $i \in \{1, \ldots, q\}$ we guess the number r_i of perfect matchings from $\mathcal{M}$ that are *close* to M_i, and use procedure **P2** to compute a set of r_i diverse perfect matchings that are close to M_i. The union of all the matchings computed for all $i \in \{1, \ldots, q\}$ form a solution.

We use algebraic methods and color coding to design procedure **P1**. The Tutte matrix $\mathbf{A}$ of an undirected graph G over the field $\mathbb{F}_2[X]$ is defined as follows, where $\mathbb{F}_2$ is the Galois field on $\{0, 1\}$ and $X = \{x_e : e \in E(G)\}$. The rows and columns of $\mathbf{A}$ are labeled with $V(G)$ and for each $e = \{u, v\} \in E(G)$, $\mathbf{A}[u, v] = \mathbf{A}[v, u] = x_e$. All other entries in $\mathbf{A}$ are zeros. There is a bijective correspondence between the set of monomials of $det(\mathbf{A})$ and the set of perfect matchings of G. Procedure **P1** extracts the required matching from $det(\mathbf{A})$ using color coding. Procedure **P2** is realized using color coding and dynamic programming.

Related work

Recall that all bases of a matroid have the same size, and that the number of bases of a matroid on ground set E is at most $2^{|E|}$. So using the same argument as for Theorem 5 we get that Weighted Diverse Bases generalizes – via Turing reductions – the problem of *counting the number of bases* of a matroid. Each of these reduced Weighted Diverse

Bases instances will have $d = 1$, and a weight function which assigns the weight 1 to each element in the ground set. Counting the number of bases of a matroid is known to be #P-complete even for restricted classes of matroids such as transversal [3], bircircular [14], and binary matroids [20]. Hence we have the following alternative[2] hardness result for Weighted Diverse Bases

▶ **Theorem 7.** *Weighted Diverse Bases cannot be solved in time polynomial in $|E(M)|$ unless* P = NP*, even when $d = 1$ and every element of the ground set $E(M)$ has weight 1.*

The study of the parameterized complexity of finding diverse sets of solutions is a very recent development, and only a handful of results are currently known. In the work which introduced this notion Baste et al. [1] showed that diverse variants of a large class of graph problems which are FPT when parameterized by the *treewidth* of the input graph, are also FPT when parameterized by the treewidth and the number of solutions in the collection. In a second article [2] the authors show that for each fixed positive integer d, two diverse variants – one with the *minimum* Hamming distance of any pair of solutions, and the other with the *sum* of all pairwise Hamming distances of solutions – of the d-Hitting Set problem are FPT when parameterized by the size of the hitting set and the number of solutions. In a recent manuscript on diverse FPT algorithms [10] the authors show that the problem of finding *two* maximum-sized matchings in an undirected graph such that their symmetric difference is at least d, is FPT when parameterized by d. Note that our result on Diverse Perfect Matchings generalizes this to $k \geq 2$ matchings, *provided* the input graph has a perfect matching.

In a very recent manuscript Hanaka et al. [15] propose a number of results about finding diverse solutions. We briefly summarize their results which are germane to our work. For a collection of sets $X_1, \ldots, X_k$ let $d_{sum}(X_1, \ldots, X_k)$ denote the sum of all pairwise Hamming distances of these sets and let $d_{min}(X_1, \ldots, X_k)$ denote the smallest Hamming distance of any pair of sets in the collection. Hanaka et al. show that there is an algorithm which takes an independence oracle for a matroid M and an integer k as input, runs in time polynomial in $(|E(M)| + k)$, and finds a collection $B_1, B_2, \ldots, B_k$ of k bases of M which maximizes $d_{sum}(B_1, B_2, \ldots, B_k)$. This result differs from our work on Weighted Diverse Bases in two key aspects. They deal with the unweighted (counting) case, and their diversity measure is the *sum* of the pairwise symmetric differences, whereas we look at the *minimum* (weight of the) symmetric difference. These two measures are, in general, not comparable.

Hanaka et al. also look at the complexity of finding k matchings $M_1, \ldots, M_k$ in a graph G where each M_i is of size t. They show that such collections of matchings maximizing $d_{min}(M_1, \ldots, M_k)$ and $d_{sum}(M_1, \ldots, M_k)$ can be found in time $2^{\mathcal{O}(kt \log(kt))} \cdot |V(G)|^{\mathcal{O}(1)}$. The key difference with our work is that their algorithm looks for matchings of a specified size t whereas ours looks for perfect matchings, of size $t = \frac{|V(G)|}{2}$; note that this t does not appear in the exponential part of the running time of our algorithm (Theorem 6). The manuscript [15] has a variety of other interesting results on diverse FPT algorithms as well.

Organization of the rest of the paper

In the next section we collect together some preliminary results. In Section 3 we prove that Weighted Diverse Bases and Weighted Diverse Common Independent Sets are strongly NP-hard. In Section 4 we derive our FPT and kernelization algorithms for

[2] Compare with Theorem 1.

Weighted Diverse Bases, and in Section 5 we show that Weighted Diverse Common Independent Sets is FPT. We derive our results for Diverse Perfect Matchings in Section 6. We conclude in Section 7.

All the proofs omitted in this Extended Abstract can be found in the full version on arXiv at `https://arxiv.org/abs/2101.04633`.

2 Preliminaries

We use $X \triangle Y$ to denote the *symmetric difference* $(X \setminus Y) \cup (Y \setminus X)$ of sets X and Y. We use $\mathbb{N}$ to denote the set of positive integers.

A parameterized problem Π is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. We say that a parameterized problem Π is *fixed parameter tractable (*FPT*)*, if there is an algorithm that given an instance (x, k) of Π as input, solves in time $f(k)|x|^{\mathcal{O}(1)}$, where f is an arbitrary function and $|x|$ is the length of x. A kernelization algorithm for a parameterized problem Π is a polynomial time algorithm (computable function) $\mathcal{A} : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ such that $(x, k) \in \Pi$ if and only if $(x', k') = \mathcal{A}((x, k)) \in \Pi$ and $|x'| + k' \leq g(k)$ for some computable function g. When g is a polynomial function, we say that Π admits a polynomial kernel. For a detailed overview about parameterized complexity we refer to the monographs [5, 4, 11]

A pair $M = (E, \mathcal{I})$, where E is a finite *ground* set and $\mathcal{I}$ is a family of subsets of the ground set, called *independent sets* of E, is a *matroid* if it satisfies the following conditions, called *independence axioms*: **(I1)** $\emptyset \in \mathcal{I}$; **(I2)** If $A \subseteq B \subseteq E(M)$ and $B \in \mathcal{I}$ then $A \in \mathcal{I}$, and **(I3)** If $A, B \in \mathcal{I}$ and $|A| < |B|$, then there is $e \in B \setminus A$ such that $A \cup \{e\} \in \mathcal{I}$. We use $E(M)$ and $\mathcal{I}(M)$ to denote the ground set and the set of independent sets, respectively. As is standard for matroid problems, we assume that each matroid M that appears in the input is given by an *independence oracle*, that is, an oracle that in constant (or polynomial) time replies whether a given $A \subseteq E(M)$ is independent in M or not. An inclusion-wise maximal independent set B is called a *basis* of M. All the bases of M have the same size that is called the *rank* of M, denoted $\mathsf{rank}(M)$. The *rank* of a subset $A \subseteq E(M)$, denoted $\mathsf{rank}(A)$, is the maximum size of an independent set $X \subseteq A$; the function $\mathsf{rank}\colon 2^{E(M)} \to \mathbb{Z}$ is the *rank* function of M.

The *dual* of a matroid $M = (E, \mathcal{I})$, denoted M^*, is the matroid whose ground set is E and whose set of bases is $\mathcal{B}^* = \{\overline{B} \mid B \in \mathcal{B}(M)\}$. That is, the bases of M^* are exactly the complements of the bases of M. A basis (independent set, rank, respectively) of M^* is a *cobasis* (*coindependent set*, *corank*, respectively) of M. Given an independence oracle for M we can construct – with an overhead which is polynomial in $|E|$ – a *rank* oracle for M, and thence *corank* and *coindependence* oracles for M.

Let M be a matroid and let $\mathbb{F}$ be a field. An $n \times m$-matrix $\mathbf{A}$ over $\mathbb{F}$ is a *representation of M over* $\mathbb{F}$ if there is one-to-one correspondence f between $E(M)$ and the set of columns of $\mathbf{A}$ such that for any $X \subseteq E(M)$, $X \in \mathcal{I}(M)$ if and only if the columns $f(X)$ are linearly independent (as vectors of $\mathbb{F}^n$); if M has such a representation, then it is said that M has a *representation over* $\mathbb{F}$. In other words, $\mathbf{A}$ is a representation of M if M is isomorphic to the *linear matroid* of $\mathbf{A}$, i.e., the matroid whose ground set is the set of columns of $\mathbf{A}$ and a set of columns is independent if and only if these columns are linearly independent.

Let $1 \leq r \leq n$ be integers. We use U_n^r to denote the *uniform* matroid, that is, the matroid with the ground set of size n such that the bases are all r-element subsets of the ground set.

We use the classical results of Edmonds [7] and Frank [12] about the Weighed Matroid Intersection problem. The task of this problem is, given two matroids M_1 and M_2 with the same ground set E and a weight function $\omega\colon E \to \mathbb{N}$, find a set X of maximum weight such that X is independent in both matroids. Edmonds [7] proved that the problem can be solved in polynomial time for the unweighted case (that is, the task is to find a common independent set of maximum size; we refer to this variant as Matroid Intersection) and the result was generalized for the variant with the weights by Frank in [12].

▶ **Proposition 8** ([7, 12]). *Weighted Matroid Intersection can be solved in polynomial time.*

We also need another classical result of Edmonds [8] that a basis of maximum weight can be found by the greedy algorithm. Recall that, given a matroid M with a weight function $\omega\colon E(M) \to \mathbb{N}$, the greedy algorithm finds a basis B of maximum weight as follows. Initially, $B := \emptyset$. Then at each iteration, the algorithm finds an element of $x \in E(M) \setminus B$ of maximum weight such that $B \cup \{x\}$ is independent and sets $B := B \cup \{x\}$. The algorithms stops when there is no element that can be added to B.

▶ **Proposition 9** ([8]). *The greedy algorithm finds a basis of maximum weight of a weighted matroid in polynomial time.*

We need the following observation(See [17, Lemma 2.1.10]).

▶ **Observation 10.** *Let X and Y be disjoint sets such that X is independent and Y is coindependent in a matroid M. Then there is a basis B of M such that $X \subseteq B$ and $Y \cap B = \emptyset$.*

Observe that for any sets X and Y that are subsets of the same universe, $X \triangle Y = \overline{X} \triangle \overline{Y}$. This implies the following.

▶ **Observation 11.** *For every matroid M, every weight function $\omega\colon E(M) \to \mathbb{N}$, and all integers $k \geq 1$ and $d \geq 0$, the instances (M, ω, k, d) and (M^*, ω, k, d) of Weighted Diverse Bases are equivalent.*

We need the following simple observations about the symmetric differences of perfect matchings.

▶ **Observation 12.** *The cardinality of symmetric differences of perfect matchings in a graph obeys the triangle inequality. That is, for a graph G and perfect matchings M_1, M_2, M_3 in G, $|M_1 \triangle M_2| + |M_2 \triangle M_3| \geq |M_1 \triangle M_3|$.*

▶ **Observation 13.** *Let G be a graph and M_1 and M_2 be two perfect matchings in G. Then $|M_1 \triangle M_2| = 2 \cdot |M_1 \setminus M_2| = 2 \cdot |M_2 \setminus M_1|$.*

3 Hardness of Weighted Diverse Bases and Weighted Diverse Common Independent Sets

Both Weighted Diverse Bases and Weighted Diverse Common Independent Sets are NP-complete in the strong sense even for uniform matroids. Both reductions are from the 3-Partition problem which is known to be NP-complete in the strong sense, i.e., it is NP-complete even if the various integers in the input are encoded in unary [13, SP15].

▶ **Theorem 1.** *Both Weighted Diverse Bases and Weighted Diverse Common Independent Sets are strongly NP-complete, even on the uniform matroids U_n^3.*

4 An FPT algorithm and kernelization for Weighted Diverse Bases

In this section we show that Weighted Diverse Bases is FPT when parameterized by k and d. Moreover, if the input matroid is representable over a finite field and is given by such a representation, then Weighted Diverse Bases admits a polynomial kernel. We start with the observation that if the input matroid has a sufficiently big set that is simultaneously independent and coindependent, then diverse bases always exist.

▶ **Lemma 14.** *Let M be a matroid, and let $k \geq 1$ and $d \geq 0$ be integers. If there is $X \subseteq E(M)$ of size at least $k\lceil \frac{d}{2} \rceil$ such that X is simultaneously independent and coindependent, then (M, ω, k, d) is a yes-instance of Weighted Diverse Bases for any weight function ω.*

Proof. Let $X \subseteq E(M)$ be a set of size at least $k\lceil \frac{d}{2} \rceil$ such that X is simultaneously independent and coindependent. Then there is a partition $X_1, \ldots, X_k$ of X such that $|X_i| \geq \lceil \frac{d}{2} \rceil$ for every $i \in \{1, \ldots, k\}$. Let $i \in \{1, \ldots, k\}$. Since X is independent, X_i is independent, and since X is coindependent, then $X \setminus X_i$ is coindependent. Then by Observation 10, there is a basis B_i of M such that $X_i \subseteq B_i$ and $B_i \cap (X \setminus X_i) = \emptyset$. The latter property means that $B_i \cap X_j = \emptyset$ for every $j \in \{1, \ldots, k\}$ such that $j \neq i$. We consider the bases B_i defined in this manner for all $i \in \{1, \ldots, k\}$. Then for every distinct $i, j \in \{1, \ldots, k\}$, $X_i \cup X_j \subseteq B_i \triangle B_j$. Therefore, $\omega(B_i \triangle B_j) \geq \omega(X_i \cup X_j) \geq |X_i \cup X_j| = |X_i| + |X_j| \geq 2\lceil \frac{d}{2} \rceil \geq d$ for any $\omega \colon E(M) \to \mathbb{N}$. Hence, (M, ω, k, d) is a yes-instance of Weighted Diverse Bases. ◀

Using Proposition 8 we can check, in polynomial time, whether the conditions of Lemma 14 are satisfied. If they are, then we correctly return Yes. We show that if there is no such large set X as in Lemma 14 then there is a way to repeatedly apply the greedy algorithm of Proposition 9 to obtain an equivalent small instance of the problem, as stated in the following "compression" lemma.

▶ **Lemma 15.** *There is an algorithm that, given an instance (M, ω, k, d) of Weighted Diverse Bases, runs in time polynomial in $(|E(M)| + k + d)$ and either correctly decides that (M, ω, k, d) is a yes-instance or outputs an equivalent instance $(\widetilde{M}, \omega, k, d)$ of Weighted Diverse Bases such that $E(\widetilde{M}) \subseteq E(M)$ and $|E(\widetilde{M})| \leq 2\lceil \frac{d}{2} \rceil^2 k^3$. In the latter case, the algorithm also computes a partition (L, L^*) of $E(\widetilde{M})$ with the property that for every basis B of $\widetilde{M}$, $|B \cap L| \leq \lceil \frac{d}{2} \rceil k$ and $|L^* \setminus B| \leq \lceil \frac{d}{2} \rceil k$, and the algorithm outputs an independence oracle for $\widetilde{M}$ that answers queries for $\widetilde{M}$ in time polynomial in $|E(M)|$. Moreover, if M is representable over a finite field $\mathbb{F}$ and is given by such a representation, then the algorithm outputs a representation of $\widetilde{M}$ over $\mathbb{F}$.*

Given Lemma 15 it is easy to show that Weighted Diverse Bases is FPT when parameterized by k and d.

▶ **Theorem 2.** *Weighted Diverse Bases can be solved in $2^{\mathcal{O}(dk^2(\log k + \log d))} \cdot |E(M)|^{\mathcal{O}(1)}$ time.*

Proof. Let (M, ω, k, d) be an instance of Weighted Diverse Bases. We run the algorithm from Lemma 15. If the algorithm solves the problem, then we are done. Otherwise, the algorithm outputs an equivalent instance $(\widetilde{M}, \omega, k, d)$ of Weighted Diverse Bases such that $E(\widetilde{M}) \subseteq E(M)$ and $|E(\widetilde{M})| \leq 2\lceil \frac{d}{2} \rceil^2 k^3$. Moreover, the algorithm computes the partition (L, L^*) of $E(\widetilde{M})$ with the property that for every basis B of $\widetilde{M}$, $|B \cap L| \leq \lceil \frac{d}{2} \rceil k$ and $|L^* \setminus B| \leq \lceil \frac{d}{2} \rceil k$. Then we check all possible k-tuples of bases by brute force and verify whether there are k bases forming a solution. By the properties of L and L^*, $\widetilde{M}$ has

$(d^2k^3)^{\mathcal{O}(dk)}$ distinct bases. Therefore, we check at most $(d^2k^3)^{\mathcal{O}(dk^2)}$ k-tuples of bases. We conclude that this checking can be done in $2^{\mathcal{O}(dk^2(\log k+\log d))} \cdot |E(M)|^{\mathcal{O}(1)}$ time, and the claim follows. ◀

If the input matroid is given by a representation over a finite field, then Weighted Diverse Bases also admits a polynomial kernel when parameterized by k and d.

▶ **Theorem 3.** *Given a representation of the matroid M over a finite field GF(q) as input, we can compute a kernel of Weighted Diverse Bases of size $\mathcal{O}(k^6d^4\log q)$.*

Proof. Let (M, ω, k, d) be an instance of Weighted Diverse Bases. Let also $\mathbf{A}$ be its representation over GF(q). We run the algorithm from Lemma 15. If the algorithm solves the problem and reports that (M, ω, k, d) is a yes-instance, we return a trivial yes-instance of the problem. Otherwise, the algorithm outputs an equivalent instance $(\widetilde{M}, \omega, k, d)$ of Weighted Diverse Bases such that $E(\widetilde{M}) \subseteq E(M)$ and $|E(\widetilde{M})| \leq 2\lceil\frac{d}{2}\rceil^2k^3$. Moreover, the algorithm computes a representation $\tilde{\mathbf{A}}$ of $\widetilde{M}$ over GF(q). Clearly, it can be assumed that the number of rows of the matrix $\tilde{\mathbf{A}}$ equals $\mathsf{rank}(\widetilde{M})$. Since $\mathsf{rank}(\widetilde{M}) \leq |E(\widetilde{M})|$, the matrix $\tilde{\mathbf{A}}$ has $\mathcal{O}(k^6d^4)$ elements. Because $\tilde{\mathbf{A}}$ is a matrix over GF(q), it can be encoded by $\mathcal{O}(k^6d^4\log q)$ bits. Finally, note that the weights of the elements can be truncated by d, that is, we can set $\omega(e) := \min\{\omega(e), d\}$ for every $e \in E(\widetilde{M})$. Then the weights can be encoded using $\mathcal{O}(d^2k^3\log d)$ bits. This concludes the construction of our kernel. ◀

5 An FPT algorithm for Weighted Diverse Common Independent Sets

In this section we show that Weighted Diverse Common Independent Sets is FPT when parameterized by k and d. We use a similar win-win approach as for Weighted Diverse Bases and observe that if the two matroids from an instance of Weighted Diverse Common Independent Sets have a sufficiently big common independent set, then we have a yes-instance of Weighted Diverse Common Independent Sets.

▶ **Lemma 16.** *Let M_1 and M_2 be matroids with a common ground set E, and let $k \geq 1$ and $d \geq 0$ be integers. If there is an $X \subseteq E$ of size at least $k\lceil\frac{d}{2}\rceil$ such that X is a common independent set of M_1 and M_2, then (M_1, M_2, ω, k, d) is a yes-instance of Weighted Diverse Common Independent Sets for any weight function $\omega : E \to \mathbb{N}$.*

Lemma 16 implies that we can assume that the maximum size of a common independent set of the input matroids is bounded. We use this to prove the following crucial lemma.

▶ **Lemma 17.** *Let (M_1, M_2, ω, k, d) be an instance of Weighted Diverse Common Independent Sets such that the maximum size of a common independent set of M_1 and M_2 is at most s. Then there is a set $\mathcal{F}$ of common independent sets of M_1 and M_2, of size $|\mathcal{F}| = 2^{\mathcal{O}(s^2\log(ks))} \cdot d$, such that if (M_1, M_2, ω, k, d) is a yes-instance of Weighted Diverse Common Independent Sets then the instance has a solution $I_1, \ldots, I_k$ with $I_i \in \mathcal{F}$ for $i \in \{1, \ldots, k\}$. Moreover, $\mathcal{F}$ can be constructed in $2^{\mathcal{O}(s^2\log(ks))} \cdot d \cdot |E|^{\mathcal{O}(1)}$ time where E is the (common) ground set of M_1 and M_2.*

Combining Lemma 16 and Lemma 17, we obtain the main result of this section.

▶ **Theorem 4.** *Weighted Diverse Common Independent Sets can be solved in time $2^{\mathcal{O}(k^3d^2\log(kd))} \cdot |E|^{\mathcal{O}(1)}$.*

Proof. Let (M_1, M_2, ω, k, d) be an instance of Weighted Diverse Common Independent Sets. First, we use Proposition 8 to solve Matroid Intersection for M_1 and M_2 and find a common independent set X of maximum size. If $|X| \geq k\lceil\frac{d}{2}\rceil$, then by Lemma 16, we conclude that (M_1, M_2, ω, k, d) is a yes-instance. Assume that this is not the case. Then the maximum size of a common independent set of M_1 and M_2 is $s < k\lceil\frac{d}{2}\rceil$. We apply Lemma 17 and construct the set $\mathcal{F}$ of size $2^{\mathcal{O}((kd)^2\log(kd))}$ in $2^{\mathcal{O}((kd)^2\log(kd))} \cdot |E|^{\mathcal{O}(1)}$ time. By this lemma, if (M_1, M_2, ω, k, d) is a yes-instance, it has a solution $I_1, \ldots, I_k$ such that $I_i \in \mathcal{F}$ for $i \in \{1, \ldots, k\}$. Hence, to solve the problem we go over all k-tuples of the elements of $\mathcal{F}$, and for each k-tuple, we verify whether these common independent sets of M_1 and M_2 give a solution. Clearly, we have to consider $2^{\mathcal{O}(k^3d^2\log(kd))}$ tuples. Hence, the total running time is $2^{\mathcal{O}(k^3d^2\log(kd))} \cdot |E|^{\mathcal{O}(1)}$. ◀

6 Perfect Matchings

In this section we prove that Diverse Perfect Matchings is fixed parameter tractable when parameterized by k and d. There are two main ingredients to our algorithm. The first ingredient is an algorithm that helps us greedily compute a collection of matchings which are “far apart”.

▶ **Lemma 18.** *There is an algorithm that given an undirected graph G, perfect matchings $M_1, \ldots, M_r$, and a non-negative integer s, runs in time $2^{\mathcal{O}(rs)}n^{\mathcal{O}(1)}$, and outputs a perfect matching M such that $|M \setminus M_i| \geq s$ for all $i \in \{1, \ldots, r\}$ (if such a matching exists) with probability at least $\frac{2}{3}e^{-rs}$.*

The second ingredient is an algorithm which lets us compute matchings which are “close” to each matching in the “spread-out” collection computed using Lemma 18.

▶ **Lemma 19.** *There is an algorithm that given an undirected graph G, a perfect matching M, and non-negative integers r, d, s, runs in time $2^{\mathcal{O}(r^2s)}n^{\mathcal{O}(1)}$, and outputs r perfect matchings $M_1^\star, \ldots, M_r^\star$ such that $|M \triangle M_i^\star| \leq s$ for all $i \in \{1, \ldots, r\}$ and $|M_i^\star \triangle M_j^\star| \geq d$ for all distinct $i, j \in [r]$ (if such matchings exist) with probability at least e^{-rs}. If no such perfect matchings exist, then the algorithm outputs No*

Putting these two ingredients together we get

▶ **Theorem 6.** *There is an algorithm that given an instance of Diverse Perfect Matchings, runs in time $2^{2^{\mathcal{O}(kd)}}n^{\mathcal{O}(1)}$ and outputs the following: If the input is a No-instance then the algorithm outputs No. Otherwise the algorithm outputs Yes with probability at least $1 - \frac{1}{e}$.*

Proof. Let (G, k, d) be the input instance. Our algorithm $\mathcal{A}$ has two steps. In the first step of $\mathcal{A}$ we compute a collection of matchings greedily such that they are far apart using Lemma 18. Towards that first we run an algorithm to compute a maximum matching in G and let M_1 be the output. If M_1 is not a perfect matching we output No and stop. Next we iteratively apply Lemma 18 to compute a collection of perfect matchings that are far apart. Formally, at the beginning of step i, where $\leq 1 \leq i < k$, we have perfect matchings $M_1, \ldots, M_i$ such that $|M_j \setminus M_{j'}| \geq 2^{k-i}d$ for any two distinct $j, j' \in \{1, \ldots, i\}$. Now, we apply Lemma 18 with $r = i$ and $s = 2^{k-i-1}d$ and it will either output a matching M_{i+1} such that $|M_{i+1} \setminus M_j| \geq 2^{k-i-1}d$ for all $j \in \{1, \ldots, i\}$, or not. If no such matching exists, then the first step of the algorithm $\mathcal{A}$ is complete. So at the end of the first step of the algorithm $\mathcal{A}$, we have perfect matchings $M_1, \ldots, M_q$, where $q \in \{1, \ldots, k\}$ such that

(i) for any two distinct integers $i, j \in \{1, \dots, q\}$, $|M_i \setminus M_j| \geq 2^{k-q}d$, and
(ii) if $q \neq k$, then for any other perfect matching $M \notin \{M_1, \dots, M_q\}$, $|M \setminus M_j| \leq 2^{k-q-1}d$.

If $q = k$, then $\{M_1, \dots, M_k\}$ is a solution to the instance (G, k, d), and hence our algorithm $\mathcal{A}$ outputs Yes. Now on, we assume that $q \in \{1, \dots, k-1\}$. Statements (i) and (ii), and Observation 13 imply that

(iii) for any two distinct integers $i, j \in \{1, \dots, q\}$, $|M_i \triangle M_j| \geq 2^{k-q+1}d$, and
(iv) for any perfect matching $M \notin \{M_1, \dots, M_q\}$, $|M \triangle M_j| < 2^{k-q}d$.

Statements (ii) and (iv), and Observation 12 imply the following claim.

▷ Claim 20. For any perfect matching M, there exists a unique $i \in \{1, \dots, q\}$ such that $|M \triangle M_i| < 2^{k-q}d$.

Let $\mathcal{M} = \{M_1^\star, \dots, M_k^\star\}$ is a solution to the instance (G, k, d). Then, by Claim 20, there is a partition of $\mathcal{M}$ into $\mathcal{M}_1 \uplus \dots \uplus \mathcal{M}_q$ (with some blocks possibly being empty) such that for each $i \in \{1, \dots, q\}$, and each $M \in \mathcal{M}_i$, $|M \triangle M_i| \leq 2^{k-q}d$. Thus, in the second step of our algorithm $\mathcal{A}$, we guess $r_1 = |\mathcal{M}_1|, \dots, r_q = |\mathcal{M}_q|$ and apply Lemma 19. That is, for each $i \in \{1, \dots, q\}$ such that $r_i \neq 0$, we apply Lemma 19 with $M = M_i$, $r = r_i$, and $s = 2^{k-q}d$. Then for each $i \in 1, \dots, q$, let the output of Lemma 19 be $N_{i,1}, \dots, N_{r_i}$. Clearly $|N_{i,j} \triangle N_{i,j'}| \geq d$ for any two distinct $j, j' \in \{1, \dots, r_i\}$. Observation 12 and statement (iii) implies that for any two distinct $i, j \in \{1, \dots, q\}$, the cardinality of the symmetric difference between a matching in $\{N_{i,1}, \dots, N_{i,r_i}\}$ and a matching in $\{N_{j,1,\dots,N_{j,r_j}}\}$ is at least d.

If algorithm $\mathcal{A}$ computes a solution in any of the guesses for $r_1, \dots, r_d$, then we output Yes. Otherwise we output No. As the number of choices for $r_1, \dots r_k$ is upper bounded by $k^{\mathcal{O}(k)}$, from Lemmas 18 and 19 we get that the running time of $\mathcal{A}$ is $2^{2^{\mathcal{O}(kd)}} n^{\mathcal{O}(1)}$ and the success probability is at least $2^{-2^{ckd}}$ for some constant c. To get success probability $1 - 1/e$, we do $2^{2^{ckd}}$ many executions of $\mathcal{A}$ and output Yes if we succeed in at least one of the iterations and output No otherwise. Thus, running time of the overall algorithm is $2^{2^{\mathcal{O}(kd)}} n^{\mathcal{O}(1)}$. ◀

7 Conclusion

We took up weighted diverse variants of two classical matroid problems and the unweighted diverse variant of a classical graph problem. We showed that the two diverse matroid problems are NP-hard, and that the diverse graph problem cannot be solved in polynomial time even for the smallest sensible measure of diversity. We then showed that all three problems are FPT with the combined parameter (k, d) where k is the number of solutions and d is the diversity measure.

We conclude with a list of open questions:

- We showed that the unweighted, counting variant of Weighted Diverse Bases does not have a polynomial-time algorithm unless P = NP (Theorem 7). This is the case when all the weights are 1 and $d = 1$ or $d = 2$. Both the weighted and unweighted variants can be solved in polynomial time when $k = 1$ (the greedy algorithm) and $k = 2$ ((weighted) matroid intersection). What happens for larger, constant values of d and/or k? Till what values of d, k does the problem remain solvable in polynomial time? These questions are interesting also for special types of matroids. For instance, is there a polynomial-time algorithm that checks if an input graph has *three* spanning trees whose edge sets have pairwise symmetric difference at least d, or is this already NP-hard?

- A potentially easier question along the same vein would be: we know from Theorem 7 that Weighted Diverse Bases is unlikely to have an FPT algorithm parameterized by d alone. Is Weighted Diverse Bases FPT parameterized by k alone?
- Unlike for the other two problems, we don't have hardness results for Weighted Diverse Common Independent Sets for small values of k or d. Is Weighted Diverse Common Independent Sets FPT when parameterized by either d or k? Is this problem in P when all the weights are 1?

References

1 Julien Baste, Michael R. Fellows, Lars Jaffke, Tomáš Masařík, Mateus de Oliveira Oliveira, Geevarghese Philip, and Frances A. Rosamond. Diversity of solutions: An exploration through the lens of fixed-parameter tractability theory, 2019. To appear at IJCAI 2020, arXiv:1903.07410.

2 Julien Baste, Lars Jaffke, Tomáš Masařík, Geevarghese Philip, and Günter Rote. FPT algorithms for diverse collections of hitting sets. *Algorithms*, 12(12):254, 2019.

3 Charles J. Colbourn, J. Scott Provan, and Dirk Vertigan. The complexity of computing the Tutte polynomial on transversal matroids. *Combinatorica*, 15(1):1–10, 1995. doi:10.1007/BF01294456.

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

5 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.

6 Jack Edmonds. Lehman's switching game and a theorem of tutte and nash-williams. *J. Res. Nat. Bur. Standards Sect. B*, 69:73–77, 1965.

7 Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In *Combinatorial Structures and their Applications (Proc. Calgary Internat. Conf., Calgary, Alta., 1969)*, pages 69–87. Gordon and Breach, New York, 1970.

8 Jack Edmonds. Matroids and the greedy algorithm. *Math. Program.*, 1(1):127–136, 1971. doi:10.1007/BF01584082.

9 Michael Ralph Fellows. The diverse X paradigm. Manuscript, November 2018.

10 Fedor V. Fomin, Petr A. Golovach, Lars Jaffke, Geevarghese Philip, and Danil Sagunov. Diverse pairs of matchings. *CoRR*, abs/2009.04567, 2020. arXiv:2009.04567.

11 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing.* Cambridge University Press, 2019. doi:10.1017/9781107415157.

12 András Frank. A weighted matroid intersection algorithm. *J. Algorithms*, 2(4):328–336, 1981. doi:10.1016/0196-6774(81)90032-8.

13 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

14 Omer Giménez and Marc Noy. On the complexity of computing the Tutte polynomial of bicircular matroids. *Combin. Probab. Comput.*, 15(3):385–395, 2006. doi:10.1017/S0963548305007327.

15 Tesshu Hanaka, Yasuaki Kobayashi, Kazuhiro Kurita, and Yota Otachi. Finding diverse trees, paths, and more, 2020. arXiv preprint. arXiv:2009.03687.

16 Ian Holyer. The NP-completeness of edge-coloring. *SIAM Journal on computing*, 10(4):718–720, 1981.

17 James G. Oxley. *Matroid theory.* Oxford University Press, 1992.

18 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

19 Leslie G Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979.

20 Dirk Vertigan. Bicycle dimension and special points of the Tutte polynomial. *J. Combin. Theory Ser. B*, 74(2):378–396, 1998. doi:10.1006/jctb.1998.1860.

Rice-Like Theorems for Automata Networks

Guilhem Gamard
Aix-Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France

Pierre Guillon
Aix-Marseille Université, CNRS, I2M, Marseille, France

Kevin Perrot
Aix-Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France
Université Côte d'Azur, CNRS, I3S, Sophia Antipolis, France

Guillaume Theyssier
Aix-Marseille Université, CNRS, I2M, Marseille, France

Abstract

We prove general complexity lower bounds on automata networks, in the style of Rice's theorem, but in the computable world. Our main result is that testing any fixed first-order property on the dynamics of an automata network is either trivial, or NP-hard, or coNP-hard. Moreover, there exist such properties that are arbitrarily high in the polynomial-time hierarchy. We also prove that testing a first-order property given as input on an automata network (also part of the input) is PSPACE-hard. Besides, we show that, under a natural effectiveness condition, any nontrivial property of the limit set of a nondeterministic network is PSPACE-hard. We also show that it is PSPACE-hard to separate deterministic networks with a very high and a very low number of limit configurations; however, the problem of deciding whether the number of limit configurations is maximal up to a polynomial quantity belongs to the polynomial-time hierarchy.

2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Complexity classes

Keywords and phrases Automata networks, Rice theorem, complexity classes, polynomial hierarchy, hardness

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.32

Funding This work has been partially supported by the French ANR project FANs ANR-18-CE40-0002.

Acknowledgements We would like to thank the colleagues that were involved in some discussions on the topic at the early stages of this work and somehow convinced us to pursue and finally settle the main result.

1 Introduction

An automata network is a digraph where each node holds a state (among a finite set) that evolves in function of the states of its inbound neighbors. All the nodes evolve at the same time, in parallel. In other terms, the main difference between an automata network and a cellular automaton is that the "grid" may be an arbitrary finite digraph, and that different cells (nodes) may have different local functions. Since this definition is very general, any finite dynamical system may be encoded into an automata network in a reasonable fashion.

Initially, *Boolean* automata networks, where the set of states is required to be $\{0, 1\}$ for all nodes, were introduced in the 1940's as a formal model of neural networks [16]. Subsequently, *linear* automata networks, where the evolution function of each node is a linear combination of its inputs, were investigated [3, 6, 9, 15], still motivated by neural networks. General automata networks were then introduced in theoretical biology, in order to study the

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 32; pp. 32:1–32:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

dynamics of gene expression and inhibition [14, 24]. They have since been further considered, mostly from the standpoint of applications [7, 13, 17, 23], although theoretical results also appeared [1, 4, 8, 19, 20].

In the literature, many questions about automata networks deal with the *dynamics* of the system, i.e., the global function that it computes. For instance: does a given network have a fixed point (i.e. a stable configuration)? How many of them does it have? Does it have a cycle of exactly two configurations evolving one to the other? Does it have a configuration with at least three predecessors? As one may suspect, such questions are computationally hard to solve in general. The reason why one may have this intuition is that automata networks can be viewed as a model of computation, so they probably are subject to some kind of theorem in the flavor of Rice's [18]:

▶ **Theorem 1.1.** *Any nontrivial property of the function computed by a Turing machine is undecidable.*

One may object that automata networks are strictly less powerful than Turing machines, for they lack unbounded memory. Any question about the function computed by an automata network may be answered by exhaustive search, i.e., by enumerating all possible configurations of the network (among finitely many), and testing each of them for the desired property. That objection stands but the brute-force approach is not practical, for the number of configurations is exponential in the size of network. On the other hand, most applications of automata networks amount to answering questions about the functions that they compute. We therefore endeavor to prove results along the lines of [2]:

▶ **Metatheorem 1.2.** *Any nontrivial property of the function computed by an automata network has high computational complexity.*

Typically, "high computational complexity" means something like "NP-hard", "co-NP-hard" or even "PSPACE-hard". As a consequence, there is probably no approach significantly faster than brute-force for those questions, which makes them out of reach for our current computational power. Thus, our results show that any application of automata networks requiring fast testing of some dynamical property will have to rely on specific aspects of the practical situation under consideration.

In order to make the statement of the metatheorem precise, we need to specify the concepts of "property" and "nontriviality". We obtained several different results that fit the pattern of Metatheorem 1.2, with various tradeoffs on "property" and "nontriviality", as explained in the *contributions and organization of the paper* paragraph below.

Let F denote an automata network and X its set of configurations. The *dynamics* of F may refer to two equivalent objects: either the function $f : X \to X$ given by the action of F; or the *transition digraph* (X, E) where E is the set $\{(x, f(x)) \mid x \in X\}$.

Specifying an automata network – say, by giving a Boolean circuit for the local function of each node – is a way to specify its transition digraph in a concise way. Thus, some of our results may be interpreted as statements about succinct graphs. However, in this paper we mostly consider *deterministic* automata networks, i.e., networks whose transition digraph has out-degree one. This restriction is not common in graph theory nor in finite model theory. Still, some generalizations of our results might be of interest for those communities.

As usual with dynamical systems, the *long-term behavior* of an automata network is of special interest. For pumping reasons, a deterministic automata network is always ultimately periodic. Many practical questions can be asked about both the transient and the periodic regimes of the system; therefore, it is interesting to know that such questions are computationally hard. The periodic regime of an automata network is called its *limit dynamics.* It is the dynamics spanned by the configurations that are always visited infinitely many times whenever they are visited once. We have two kinds of results: some are about the limit dynamics of a given automata network, and some are about the full dynamics.

Contributions and organization of the paper

- In Section 2, we set up the formalism: definitions, first remarks, etc.
- In Section 3, we prove that a large class of properties over the set of limit configurations of networks are PSPACE-complete. This echoes a result from [12] on cellular automata.
- In Section 4, we show that it is PSPACE-complete to distinguish automata networks with a very small and a very high number of limit configurations. However, we show that the problem of deciding whether this number is maximal up to a polynomial quantity (in the number of states and the number of nodes) belongs to the arithmetical hierarchy.
- In Section 5, we prove that if a property on the dynamics of automata networks is expressible by a first-order formula over a simple signature, then its complexity is either bounded, NP-hard, or co-NP-hard. In this setting, the formula is considered fixed, not part of the input. We also observe that this result still holds when restricted to bijective automata networks, or to limit dynamics instead of full dynamics.
- In Section 6, we show that if the first-order formula is considered as part of the input, then the previous problem becomes PSPACE-complete.

2 Definitions and terminology

Let $\{A_i\}_{i\in I}$ denote a finite family of finite sets and $A = \prod_{i\in I} A_i$. An *automata network* (AN) is a function f from A to itself. We think of it as a system of finite automata linked to each other, where the input of one automaton is the current state of the other automata (thus there is no external input word). More precisely, A_i is the set of states of the i^{th} automaton (or node); an element of A is a *configuration* of the system (it assigns one state to each automaton); and $f : A \to A$ gives the *evolution* of the system after one step of time.

We can split f into a family of *local functions* $\{f_i\}_{i\in I}$, where f_i goes from A to A_i and returns the state of the i^{th} automaton at the next step. In other terms, if $I = \{1, \dots, n\}$ and $a = (a_1, \dots, a_n)$ is an element of A, then we have $f(a) = (f_1(a), f_2(a), \dots, f_n(a))$. For a given i, it might happen that $f_i(a)$ does not depend on all the components of a. For instance, $f_i(a)$ might depend only on a_i and a_{i-1} (where a_0 stands for a_n). The *interaction digraph* G_f of f is the graph $(I, \mathcal{I})$ where $\mathcal{I}$ is the set of pairs (i, j) such that, for some a, b with $a_k = b_k$ for every $k \neq i$, we have $f_j(a) \neq f_j(b)$. A configuration of an automata network may be viewed as a labeling of the interaction digraph. The label of each node evolves under f, but the new label depends only on the labels of the inbound neighbours of the node.

The *dynamics*, or *transition digraph*, of a network f, denoted by $\mathcal{G}_f$, is the graph of the function f: it is given by $(A, \mathcal{F})$, where $\mathcal{F}$ is the set of pairs $(a, f(a))$, for a ranging over A. The *limit set* of an automata network f is denoted Ω_f and defined as $\Omega_f = \bigcap_{n\in\mathbb{N}} f^n(A)$. Its elements are the *limit configurations*, which are those that are met infinitely often in at least one execution of the system. The *limit dynamics* or the *limit digraph* of f is denoted $\mathcal{G}_f^\omega$ and is the subgraph of $\mathcal{G}_f$ induced by Ω_f. Figure 1 illustrates the definitions so far.

When all the A_i's are equal, we say that f is an *automata network with uniform alphabet,* or ANU for short.

All these definitions generalize immediately if f is a relation instead of a function; the f_i's are then *local relations.* Such an object is called a *nondeterministic* automata network. Unless explicitly mentioned, automata networks are supposed to be deterministic.

$$f_1(x) = \begin{cases} x_1 + 1 & \text{if } x_1 \neq 2 \\ x_1 - 1 & \text{otherwise} \end{cases}$$

$$f_2(x) = \begin{cases} x_2 + 1 & \text{if } x_1 \neq 2 \\ x_2 - 1 & \text{otherwise} \end{cases}$$

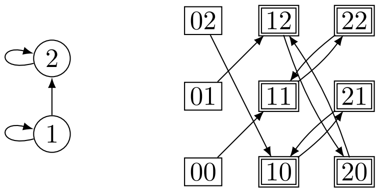


Figure 1 Example of automata network f for $A_1 = A_2 = \{0, 1, 2\}$ (left), its interaction digraph G_f (middle), and its transition digraph $\mathcal{G}_f$ (right). $\Omega_f = \{10, 11, 12, 20, 21, 22\}$.

When we need to give an automata network as input for an algorithm, we provide the interaction digraph and one Boolean circuit for each local function (or relation). Circuit sizes are assumed to be at most $|A_i|^{|A|}$ (or $2^{|A_i| \cdot |A|}$), because those are the sizes of the corresponding truth tables. The nodes of the interaction digraph are assumed to be numbered 1, …, n and each A_i is assumed to be of the form $\{0, \ldots, |A_i| - 1\}$.

If P is a property that automata networks may or may not satisfy, and f is an automata network, then we write $f \models P$ if f satisfies P, and $f \not\models P$ otherwise. This is an abuse of notation, and its precise meaning depends on the exact nature of P.

Unless otherwise stated, our reductions are polynomial-time many-one, and $\leq^p_{tt}$ denotes polynomial-time truth-table reduction. For every integer k, the symbol Σ^p_k denotes the level Σ_k of the polynomial hierarchy. For a decision problem P where an ANU is given as input, it is natural to consider the Q-variant where the inputs are restricted to ANU having alphabet Q. In the following, we will say that P is hard *with fixed alphabet* if there exists some Q such that the Q-variant of the problem is hard. It will be the case of most of our hardness results on ANU.

3 Abstract properties of limit sets

In this section, we focus on properties of the *limit set* and establish a Rice-like theorem similar to the well-known result for limit sets of cellular automata [12].

A property P is a *limit set property* if, whenever two AN f and g have the same limit sets ($\Omega_f = \Omega_g$), the following holds: $f \models P \iff g \models P$. The simplest possible limit set is a singleton, and the following lemma already shows that separating between singleton limit sets and exponentially large ones can depend on arbitrary linear space Turing computations.

▶ **Lemma 3.1.** *For any Turing machine M, any $k \in \mathbb{N}$, any $n \in \mathbb{N}$ and any input u for M of size at most n, there is an alphabet Q depending only on M and k and a deterministic ANU $f_{M,k,n,u} : Q^n \to Q^n$ and $q_0 \in Q$ such that:*

- *if M accepts input u in at most $k^n - 1$ steps, using space at most n, then $\Omega_{f_{M,k,n,u}} = \{q_0^n\}$;*
- *otherwise, there is a configuration $c \in \Omega_{f_{M,k,n,u}}$ belonging to a cyclic orbit of length k^n where state q_0 never appears: $\forall t \in \mathbb{N}, \forall i \in \{1, \ldots, n\} : f^t_{M,k,n,u}(c)_i \neq q_0$.*

Moreover, circuits for the local functions of $f_{M,k,n,u}$ can be computed in time $\mathsf{poly}(M, k, n, u)$.

Proof. Let Σ be the alphabet, S the state set of M, and $\bot$, q_0 fresh symbols. Define $H = S \sqcup \{\bot\}$ and $Q = (\Sigma \times H \times \{0, \ldots, k-1\}) \sqcup \{q_0\}$. Any configuration $c \in Q^n$ not containing state q_0 (*i.e.* $c_i \neq q_0$ for all i) can be seen as a triple (c^Σ, c^H, c^k) of configurations in Σ^n, H^n and $\{1, \ldots, k\}^n$ respectively. We say that configuration c is *valid* if it does not contain state q_0 and there is exactly one position i such that $c^H_i \in S$. If c is valid, c^Σ encodes the content of the tape (limited to n cells), c^H encodes the position and the state of the

Turing head, and c^k encodes a counter between 0 and $k^n - 1$ in base k. We call c_0 the valid configuration where c^Σ represents the tape containing input u starting on the leftmost position of the (finite) tape, c^H represents the head in the initial state on the leftmost position of the tape, and c^k represents the number 0. The behavior of $f_{M,k,n,u}$ is as follows:

1. send any invalid configuration to q_0^n;
2. send any valid configuration c such that c^k represents value $k^n - 1$ to configuration c_0;
3. send any valid c such that c^H represents a head in an accepting state to q_0^n;
4. for any valid configuration c from which one step of M does not make the head move outside the n-cell tape, $f_{M,k,n,u}$ performs this step and increments the value in c^k;
5. for any other valid configuration c, leave c^Σ and c^H unchanged but increment c^k.

It should be clear enough from the above description that circuits computing local maps of $f_{M,k,n,u}$ can be produced in polynomial time given M, k, n and u.

Suppose first that M halts on input u in at most $k^n - 1$ steps and using space at most n and suppose for the sake of contradiction that there is a configuration $c \in \Omega_{f_{M,k,n,u}}$ which is not q_0^n. Then q_0^n cannot be in the orbit of c, so only cases 2, 4 and 5 are used in the orbit of c. The counter is always incremented, until reaching $k^n - 1$, so that c_0 must appear in the (periodic) orbit of c, and therefore c is in the orbit of c_0. We get a contradiction because M halts on input u in at most $k^n - 1$ steps and using space at most n, so that case 3 must be triggered at the corresponding step in the orbit of c_0.

Suppose now that M does not halt within time $k^n - 1$ and space n starting from input u. We claim that c_0 belongs to a cyclic orbit of length k^n and that state q_0 cannot appear in this orbit. Indeed, validity of configurations is preserved under iteration of $f_{M,k,n,u}$ except in case 3, which is discarded by hypothesis, and after $k^n - 1$ applications of case 4 or 5, during which the counter component c^k is constantly incremented, we reach case 2 and the orbits cycles back to c_0. ◀

▶ **Corollary 3.2.** *The following problem is* PSPACE*-complete, even with fixed alphabet:*

Nilpotency
Input: a deterministic ANU $f : \{0, \dots, q-1\}^n \to \{0, \dots, q-1\}^n$.
Question: does $|\Omega_f| = 1$*?*

Proof. First, the problem is in PSPACE because checking that an AN on $\{q\}^n$ has a singleton limit set can be done by checking that $f^{q^n}(x)$ is the same configuration for all $x \in \{q\}^n$. Second, we can make a reduction from quantified Boolean satisfiability (QBF) problem [22] as follows: let M be any Turing machine that solves the QBF problem in linear space by a brute force algorithm and let k be large enough so that M works in less than k^n time steps on instances of QBF of size at most n. By Lemma 3.1, given an instance u of size at most n of QBF to be solved by M, the AN $f_{M,k,n,u}$ can be produced in polynomial time and has a singleton limit set if and only if u is true. The alphabet of $f_{M,k,n,u}$ only depends on machine M, so we have a reduction working with fixed alphabet ANU. ◀

Next, we present another theorem, whose proof is inspired by [12]. Intuitively, the firing squad from [12] is replaced by nondeterminism, and the nilpotency problem is replaced by the problem of having an orbit completely avoiding a given state (whose hardness is established by Lemma 3.1).

Given a collection of AN (deterministic or not, ANU or not, etc), we say that a property is *effectively nontrivial* in the collection if there is a polynomial-time algorithm that, given n in unary, produces two AN with n nodes belonging in this collection, one that satisfies the property and another one that does not. This condition of effectiveness is natural since, if one wants to make some reduction to prove that a property is hard, then the reduction usually induces an algorithm to produce models and counter-models of the property.

▶ **Theorem 3.3.** *Effectively nontrivial limit set properties of nondeterministic AN are the same as effectively nontrivial limit set properties of nondeterministic ANU. If $\mathcal{P}$ is an effectively nontrivial limit set property of nondeterministic ANU, then the following problem is* PSPACE*-hard for the $\leq_{tt}^{p}$ reduction:*

$\boldsymbol{\mathcal{P}}$-limit-set
Input: a nondeterministic ANU $f : \{0, \ldots, q-1\}^n \to \{0, \ldots, q-1\}^n$.
Question: does $f \models \mathcal{P}$?

Proof. Every effectively nontrivial limit set property for nondeterministic ANU is also effectively nontrivial for nondeterministic AN. Conversely, if P is effectively nontrivial for nondeterministic AN, then there is a polynomial-time algorithm which, given n, produces a model f_1 and a counter-model f_2 of P that may have nonuniform alphabets, but we can extend them to larger alphabets while preserving the limit set by sending (deterministically) any extra configuration to a fixed one that uses only the original alphabet. This transformation is effective (a description by circuits of the new rule can be computed in polynomial time from the description of the original rule). This proves the first claim of the theorem and allows us to focus on ANU.

Given two (possibly nondeterministic) ANU f_1 and f_2 both acting on Q_f^n, and a deterministic one h on Q_h^n with some distinguished state $q_0 \in Q_h$, we define two nondeterministic ANU g_1 and g_2 both acting on $(Q_f \cup (Q_f \times Q_h))^n$ as follows. We fix some $q \in Q_f$. Intuitively, g_i mimics f_i on Q_f^n, and either mimics h on $(Q_f \times Q_h)^n$ or projects onto the Q_f component provided state q_0 is not present in the Q_h component of states. In any other case, the behavior is go to configuration q^n deterministically. To simplify notation we see any configuration $x \in (Q_f \times Q_h)^n$ as a pair $x = (x^f, x^h) \in Q_f^n \times Q_h^n$. Then g_i is defined as follows:

$$g_i(x)_v = \begin{cases} f_i(x)_v & \text{if } x \in Q_f^n, \\ \{(x_v^f, h(x^h)_v), x_v^f\} & \text{if } x = (x^f, x^h) \in Q_f^n \times Q_h^n \text{ and } x_j^h \neq q_0 \text{ for all } j \in [n], \\ q & \text{otherwise,} \end{cases}$$

for $v \in \{n\}$ and $i = 1, 2$. It is straightforward to check that if state q_0 appears in all orbits of h, then $\Omega_{g_i} = \Omega_{f_i}$ because, in this case, any orbit of g_i must end up in Q_f^n. In particular, in this case, $\Omega_{g_1} \neq \Omega_{g_2}$ because f_1 and f_2 are respectively a model and a counter-model of the limit-set property P. On the other hand, if h has some orbit that completely avoids state q_0, then for any $x \in Q_f^n$ we have $x \in \Omega_{g_i}$ because x can be reached arbitrarily late from (x, y) in the dynamics of g_i where y is any configuration of the considered orbit of h. Moreover in this case $\Omega_{g_1} = \Omega_{g_2}$ holds because, by definition, for any $y \notin Q_f^n$ we have $y \in \Omega_{g_1} \Leftrightarrow y \in \Omega_{g_2}$. Thus we have a $\leq_{tt}^{p}$ reduction from the problem of deciding whether h has an orbit completely avoiding state q_0 to the property P: the former can be decided by checking whether $\Omega_{g_1} = \Omega_{g_2}$.

To conclude the proof, it is sufficient to invoke Lemma 3.1 and use an argument similar to the proof of Corollary 3.2, in order to show that deciding whether a given AN h has an orbit completely avoiding state q_0 is PSPACE-hard. ◀

4 Size of limit sets

In this section, we are interested in problems about the size of limit sets. First, if we take the settings of nondeterministic AN as in the previous section, Theorem 3.3 already tells us that any effectively nontrivial problem about the size of the limit set will be PSPACE-hard, as a particular limit set property. We now focus on deterministic ANU, and the following canonical problems on the size of limit sets.

Given a map $\lambda : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that $\lambda(q,n) \leq q^n$, we define the problem P_λ as follows:

Problem P_λ
Input: a deterministic ANU $f : \{0,\dots,q-1\}^n \to \{0,\dots,q-1\}^n$.
Question: does $\left|\Omega_f\right| \geq \lambda(q,n)$?

The goal of this section is to show that problem P_λ jumps from PSPACE-hardness down to the polynomial-time hierarchy depending on λ. First, when λ stays far enough from the total number of configurations, we already have the tools to conclude PSPACE-hardness.

Using Lemma 3.1 as in the proof of Corollary 3.2 we obtain the following theorem.

▶ **Theorem 4.1.** *Let $\lambda : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be a map such that for some $k > 1$ and for any $n \in \mathbb{N}$ it holds $2 \leq \lambda(q,n) \leq k^n$. Then the problem P_λ is* PSPACE*-hard, even with fixed alphabet.*

However, the problem whether the size of the limit set is maximal up to a polynomial quantity belongs in the polynomial-time hierarchy. The intuition is that if the limit set is close to maximal, then it is reached quickly under iterations of the AN.

▶ **Proposition 4.2.** *Let $\delta : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be a polynomial map, and define $\lambda(n,q) = q^n - \delta(n,q)$. Then problem P_λ is Σ_3^p and* co-NP*-hard, even with fixed alphabet.*

Proof. Let us denote $[q] = \{0,\dots,q-1\}$. Consider any deterministic ANU $f : [q]^n \to [q]^n$. We claim that if $\left|\Omega_f\right| \geq \lambda(q,n)$ then $\Omega_f = f^{\delta(q,n)}([q]^n) = f^{\delta(q,n)+1}([q]^n)$. Indeed, by induction, we have that $f^k([q]^n) = f^{k+1}([q]^n)$ for some k implies that $f^k([q]^n) = \Omega_f$. Therefore $\Omega_f \subsetneq f^{\delta(q,n)}([q]^n)$ would imply $\left|\Omega_f\right| < q^n - \delta(q,n)$. The claim follows. Conversely, the same argument shows that $f^{\delta(q,n)}([q]^n) \neq f^{\delta(q,n)+1}([q]^n)$ implies $\left|\Omega_f\right| < \lambda(q,n)$.

We deduce that the problem P_λ is equivalent to: "$f^{\delta(q,n)}([q]^n) = f^{\delta(q,n)+1}([q]^n)$ and $\left|[q]^n \setminus f^{\delta(q,n)}([q]^n)\right| \leq \delta(q,n)$." This can be rephrased as the conjunction:

- $\forall x \in [q]^n, \exists y \in [q]^n$ such that $f^{\delta(q,n)}(x) = f^{\delta(q,n)+1}(y)$, and
- there is a set $L \subseteq [q]^n$ of $\delta(q,n)$ distinct configurations such that $\forall x \in [q]^n, f^{\delta(q,n)}(x) \notin L$ and $\forall x \in [q]^n, x \notin L \Rightarrow \exists y \in [q]^n, f^{\delta(q,n)}(y) = x$

This shows that the problem P_λ is Σ_3^p.

To show co-NP-hardness of problem P_λ we make a reduction from UNSAT. First note that $\delta(4,n) < 2^n - 1$ for large enough n because $\delta(4,n)$ is a polynomial in n. Then, given any instance ϕ of UNSAT with n variables, build the ANU $f : [4]^n \to [4]^n$ as follows:

$$f(x) = \begin{cases} 0^n & \text{if } \pi(x) \text{ represents a satisfying assignment of variables for } \phi, \\ x & \text{otherwise.} \end{cases}$$

where $\pi(x_1,\dots,x_n) = (t_1,\dots,t_n)$ with t_i true if and only if $x_i = 0 \bmod 2$. It is clear that for any assignment of variables $(t_1,\dots,t_n)$ there are 2^n possible choices of x such that $\pi(x) = (t_1,\dots,t_n)$. Moreover, if $x \neq 0^n$, then $x \notin \Omega_f$ when $\pi(x)$ is a satisfying assignment for ϕ. Therefore if ϕ is satisfiable then $\left|\Omega_f\right| \leq 4^n - 2^n + 1 < \lambda(4,n)$. On the contrary, if ϕ is not satisfiable, then $f(x) = x$ for all $x \in [4]^n$ so we have $\left|\Omega_f\right| \geq \lambda(4,n)$. co-NP-hardness of problem P_λ follows. ◀

5 First-order properties of transition digraphs are hard

In this section, a *graph* is the transition digraph of some deterministic automata network, i.e., a simple digraph where all vertices have out-degree 1 and where self-loops are allowed. A *formula* means a closed first-order logic formula over the signature $\{=, \to\}$ (binary relations). Formulas will be evaluated in graphs, so "$\forall x$" is understood as "for all vertex x" and "$x \to y$" is understood as "there is an edge from x to y". For all formula ψ, define:

ψ-Dynamics
Input: an automata network f.
Question: does $\mathcal{G}_f \models \psi$?

Note that the formula ψ is *not* part of the input, but rather a parameter of the problem.

▶ **Definition 5.1.** *A formula ψ is ω-nontrivial if there are infinitely many models and infinitely many countermodels.*

▶ **Theorem 5.2.** *If ψ is ω-nontrivial, then ψ-Dynamics is either* NP- *or* co-NP*-hard.*

The condition of ω-nontriviality is optimal: indeed, if ψ is ω-trivial, then solving ψ-Dynamics amounts to testing whether the given AN belongs to a finite fixed list of objects, which can be done in time $O(1)$. (Recall from Section 2 that the circuits sizes are bounded by $|A_i|^{|A|}$.) Whether the problem is NP- or co-NP-hard varies with ψ. The proof consists of the next three subsections.

5.1 Encoding SAT instances into the dynamics of AN

The results in this subsection provide a general tool to deduce hardness from pumping constructions. We recommend that first-time readers skip the definitions of $\sqcup_2$ and $\sqcup_3$ and fix $z = 1$ everywhere, because the cases $z = 2, 3$ will not be needed until much later in the paper. In the next definition, "pointed" nodes are simply distinguished vertices in a graph.

▶ **Definition 5.1.1.** Let G, G' denote graphs; we define three operators $\sqcup_1, \sqcup_2, \sqcup_3$.

- The graph $G \sqcup_1 G'$ (or $G \sqcup G'$) is the disjoint union of a copy of G and a copy of G'.
- If G has a pointed node v and G' has any number of pointed nodes (possibly zero), then the graph $G \sqcup_2 G'$ is $G \sqcup_1 G'$ except that each edge going out of a pointed node of G' points to v instead. The result has one pointed node, v.
- If G has a pair of pointed nodes (u, v) and G' has a pair of pointed nodes (u', v'), then $G \sqcup_3 G'$ is $G \sqcup_1 G'$ except that: the edge going out of v points to u'; and the edge going out of v' points to u. Besides, $G \sqcup_3 G'$ has pointed nodes (u', v).

If G is a graph, k is an integer, and z is in $\{1, 2, 3\}$, then $\bigsqcup_z^k G$ denotes $G \sqcup_z \cdots \sqcup_z G$, with k copies of G. Now let n be an integer, $\Gamma = (G_1, \dots, G_n)$ a n-tuple of graphs, and w a word over alphabet $\{1, \dots, n\}$. Define $\mathcal{U}_z^{G,\Gamma}(w)$ by induction as follows: $\mathcal{U}_z^{G,\Gamma}(\varepsilon) = G$, and $\mathcal{U}_z^{G,\Gamma}(w_1 \dots w_k) = \mathcal{U}_z^{G,\Gamma}(w_1 \dots w_{k-1}) \sqcup_z G_{w_k}$ (where ε is the empty word). See Figure 2.

▶ **Proposition 5.1.2.** *Let ψ be a formula and z be an element of $\{1, 2, 3\}$. If there exist nonempty graphs G, J, D such that for all integers k and k', we have $G \sqcup_z (\bigsqcup_z^k J) \not\models \psi$ and $G \sqcup_z (\bigsqcup_z^k J) \sqcup_z (\bigsqcup_z^{k'} D) \models \psi$, then ψ-Dynamics is* NP*-hard.*

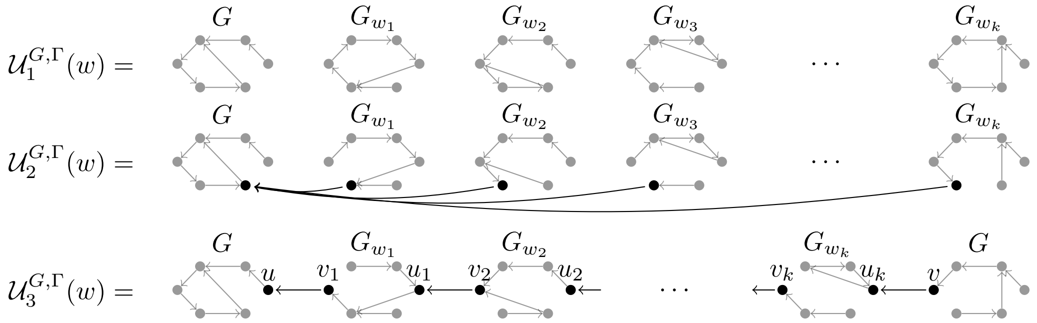


Figure 2 Illustration of $\mathcal{U}_z^{G,\Gamma}(w)$. In the illustration of $\mathcal{U}_3^{G,\Gamma}(w)$, G is not connected.

▶ **Definition 5.1.3.** Let S denote an instance of SAT with s variables. Then $\overline{S}$ is the word of length 2^s over alphabet $\{1,2\}$ whose i^{th} letter is 1 if $S(i)$ is false, and 2 if it is true (viewing the binary expansion of i as an assignment for S).

▶ **Lemma 5.1.4.** *Let S be an instance of SAT, $z \in \{1,2,3\}$ and G, J, D be graphs such that $1 < |G| < |J| = |D|$. There are an AN f and an integer k such that $\mathcal{G}_f = \mathcal{U}_z^{G,(J,D)}(\overline{S}) \sqcup_z (\bigsqcup_z^k J)$. Moreover, f is computable in polynomial time from S if G, J, D are constant.*

Proof. Let $\delta = \gcd(|G|,|J|)$, and write $|G| = g \cdot \delta$ and $|J| = j \cdot \delta$ for some coprime integers g, j. Call s the number of variables in S. First, find an integer t such that $g \leq 2^t$ and $\gcd(s+t, \varphi(j)) = 1$, where φ denotes Euler's totient. To do so, let $t' = s/\gcd(s,\varphi(j))$, so that t' and $\varphi(j)$ have no common prime factors. Then let t'' denote a power of t' that exceeds $s + \lceil \log_2 g \rceil$ (compute it by successive squarings). Finally, take $t = t'' - s$. Since $\gcd(g,j) = 1$, we can use Algorithm 17.1 of [21] to find an integer $x \geq 1$ such that $x^{s+t} \equiv g \mod j$. For the rest of the proof, assume that $x \geq 2$: indeed, if $x = 1$, then $g \equiv 1 \mod j$, so we can choose $x = g^{\varphi(j)}$ instead by Euler's formula. Since Algorithm 17.1 runs in polynomial time and g, j are constants, we can find x and t in polynomial time.

Assume that $V(G) = \{0,\dots,|G|-1\}$ and $V(J) = V(D) = \{0,\dots,|J|-1\}$ (recall that $|J| = |D|$). For all relevant integer n, write $G(n)$ (resp. $J(n)$, $D(n)$) the unique successor of n in G (resp. J, D). The automata network f has $1+s+t$ nodes: one node with alphabet $\{0,\dots,\delta-1\}$ and $s+t$ nodes with alphabet $\{0,\dots,x-1\}$. It reads its current configuration as an integer N (with $0 \leq N \leq \delta \cdot x^{s+t} - 1$) and transitions as follows:

- If $N < |G|$, then $f(N) = G(N)$.
- If $0 \leq N - |G| < 2^s \cdot |J|$, then by Euclidean division let q, r be the integers such that $N - |G| = |J| \cdot q + r$ and $0 \leq r < |J|$. View q in binary as a valuation for S.
  If $S(q)$ is true, then $f(N) = |J| \cdot q + D(r)$. If $S(q)$ is false, then $f(N) = |J| \cdot q + J(r)$.
- If $2^s \cdot |J| \leq N - |G|$, then let q, r be as in the previous case, and $f(N) = |J| \cdot q + J(r)$.

If $z = 1$, the description of f is complete. If $z = 2$, the pointed nodes of each copy of J and D transition to the pointed node of G instead. If $z = 3$, order all the graphs (G, J's, and D's) according to the configuration number N that encodes their first vertex. Encode the pointed nodes of each graph in their vertices 0 and 1. Make the first pointed node of each graph transition to the second pointed node of the next graph, looping around $\delta \cdot x^{s+t}$.

Since $2 \leq x$, we have $g \leq 2^t \leq x^t$, so one copy of G and at least 2^s copies of J or D fit in the dynamics of f. Besides, since $x^{s+t} \equiv g \mod j$, there are no leftover configurations. The circuits encoding f can be produced in polynomial time: the only part depending on S merely requires to evaluate S. ◀

Proof of Proposition 5.1.2. Let $\widetilde{J} = \bigsqcup_z^{|D|} J$ and $\widetilde{D} = \bigsqcup_z^{|J|} D$, so that $|\widetilde{J}| = |\widetilde{D}|$. The statement follows from Lemma 5.1.4, as the graphs G, $\widetilde{J}$ and $\widetilde{D}$ can be padded with copies of $\widetilde{J}$ to meet the other size constraints. ◀

5.2 From transition digraphs to disjoint unions of labeled cycles

Recall that all our graphs have out-degree 1, so each connected component of a graph is a cycle, in which each vertex is the root of an upward tree (a rooted tree where arcs point towards the root). Define $\mathcal{T}$ as the set of finite, nonempty upward trees. Any graph may be seen as a multiset of cyclic words over alphabet $\mathcal{T}$.

If G and G' are graphs, we write $G \equiv_m G'$ if and only if they satisfy the same formulas of quantifier rank m. Let $\mathcal{E}_m$ denote the set of equivalence classes of $\equiv_m$ over $\mathcal{T}$.

▶ **Lemma 5.2.1.** *For all m, the set $\mathcal{E}_m$ is finite.*

Proof. Without loss of generality, all formulas are in prenex form (quantifiers are at the beginning). Thus, a formula ϕ is of the form $Q_1 x_1 \ldots Q_m x_m\ \phi'(x_1, \ldots, x_m)$, where Q_i belongs to $\{\exists, \forall\}$ for all i and ϕ' is a quantifier-free formula. There are 2^m ways to assign quantifiers to the Q_i's. A quantifier-free formula $\phi'(x_0, \ldots, x_{m-1})$ is a Boolean formula over $2m^2$ variables: "$x_i \to x_j$" and "$x_i = x_j$", for $0 \leq i, j < m$. Two Boolean formulas are equivalent if and only if they have the same truth table. There are 2^{2m^2} possible assignments for the "variables", thus $2^{2^{2m^2}}$ possible truth tables. Consequently, there are at most $2^{m+2^{2m^2}}$ nonequivalent formulas of quantifier rank m. Any structure satisfying (resp. falsifying) a formula has to satisfy (resp. falsify) all formulas equivalent to it. Therefore, there are finitely many possible sets of formulas of quantifier rank m that a given structure may satisfy. ◀

For all T in $\mathcal{T}$, let $\mathcal{E}_m(T)$ denote the equivalence class of T for $\equiv_m$. We extend the map $\mathcal{E}_m$ to finite words, cyclic or not: if $w = w_1 w_2 \ldots w_k$ is a word over $\mathcal{T}$, then $\mathcal{E}_m(w)$ is the word $\mathcal{E}_m(w_1)\mathcal{E}_m(w_2)\ldots\mathcal{E}_m(w_k)$. We further extend $\mathcal{E}_m$ to sets and multisets of words: if $Y = \{y_1, \ldots, y_n\}$ is a (multi)set of finite words over $\mathcal{T}$, then $\mathcal{E}_m(Y)$ denotes $\{\mathcal{E}_m(y_1), \ldots, \mathcal{E}_m(y_n)\}$. Since any graph may be viewed as a multiset of cyclic words over $\mathcal{T}$, it makes sense to write $\mathcal{E}_m(G)$ for all graph G.

▶ **Definition 5.2.2.** A *DULC* is a finite digraph that is a vertex-Disjoint Union of Labeled Cycles, where the labels are in $\mathcal{E}_m$.

All graphs of the form $\mathcal{E}_m(G)$ are DULC. Now define a new signature, with two binary relation synbols $=$ and $\to$ as before, and one unary relation symbol per element of $\mathcal{E}_m$. Formulas ϕ with this signature talk about graphs where vertices are $\mathcal{E}_m$-labeled (possibly with some multiply-labeled vertices, but this does not matter), such as DULC.

▶ **Theorem 5.2.3.** *For all m and all graphs G, G', if $\mathcal{E}_m(G) \equiv_m \mathcal{E}_m(G')$ then $G \equiv_m G'$.*

Proof. By Lemma 5.2.1, the set $\mathcal{E}_m$ is finite. Assume that $\mathcal{E}_m(G) \equiv_m \mathcal{E}_m(G')$; we show that $G \equiv_m G'$ with the Ehrenfeucht-Fraïssé method (see for instance [5, Theorem 2.2.8] or [11, Theorem 6.10]), by giving a winning strategy for Duplicator. Suppose that Spoiler plays somewhere in a tree t of G (the case of G' is symmetric). Let u be the node of $\mathcal{E}_m(G)$ corresponding to t. Imagine a game in $\mathcal{E}_m(G)/\mathcal{E}_m(G')$ where Spoiler just picked u in $\mathcal{E}_m(G)$ and let u' be the node picked by Duplicator in $\mathcal{E}_m(G')$ as a response (since $\mathcal{E}_m(G) \equiv_m \mathcal{E}_m(G')$, Duplicator has a winning strategy there). Let t' denote the tree of G' corresponding to u'

in $\mathcal{E}_m(G')$. Since u and u' have the same label (otherwise Duplicator would not win in the $\mathcal{E}_m(G)/\mathcal{E}_m(G')$ game), by definition of $\mathcal{E}_m$ we have $t \equiv_m t'$, so Duplicator has a winning strategy in the game t/t'. Therefore, in order to choose which node of t' to pick, Duplicator applies her t/t' winning strategy. The next turns go on similarly: Duplicator maintains a virtual game in $\mathcal{E}_m(G)/\mathcal{E}_m(G')$, and one more virtual game for each tree touched in the main game. In that manner, she can always retort to Spoiler in a way that maintains a local isomorphism. ◀

▶ **Theorem 5.2.4.** *For all integer m and all formula ψ of rank m, there is a formula $\mathcal{E}(\psi)$ such that for all graph G, we have $G \models \psi$ if and only if $\mathcal{E}_m(G) \models \mathcal{E}(\psi)$.*

Theorem 5.2.4 does not imply the converse of Theorem 5.2.3 because the rank of $\mathcal{E}(\psi)$ may be higher than m. We do not know whether the converse of Theorem 5.2.3 is true. To prove Theorem 5.2.4, we first rephrase Hanf's lemma for DULC.

▶ **Definition 5.2.5.** An *r-ball* in a graph, where r is an integer, is a subgraph induced by vertices linked to a given vertex by a path of length at most r. An *r-ball type* occuring in a graph is the graph-isomorphism class for a ball (for isomorphisms preserving the center).

▶ **Remark 5.2.6.** The possible 3^m-ball types in DULC are the pointed cycles of length at most $2 \cdot 3^m + 1$ and the path of length exactly $2 \cdot 3^m + 1$, pointed in its center.

▶ **Definition 5.2.7.** Let m be an integer, $e = 2 \cdot 3^m + 1$ the maximum number of vertices in a 3^m-ball of a DULC, and B_m the (finite) set of possible 3^m-balls types in DULC. Given a DULC H, its *profile* is the function $\pi_{H,m} : B_m \to \{0, \dots, m \cdot e\} \sqcup \{\omega\}$ defined as follows: $\pi_{H,m}(b)$ is the number of balls in H that are isomorphic to b in the case that it does not exceed $m \cdot e$, and ω otherwise.

We extend the usual order $\leq$ to $\{0, \dots, m \cdot e\} \sqcup \{\omega\}$ by making ω a global maximum. This yields a partial order over profiles: $\pi \leq \pi'$ if for all b, we have $\pi(b) \leq \pi'(b)$.

▶ **Lemma 5.2.8** (Hanf's lemma [10, Lemma 2.3] along with Remark 5.2.6). *Let m be an integer, and H and H' be DULC. If $\pi_{H,m} = \pi_{H',m}$, then $H \equiv_m H'$.*

We call a profile *ϕ-positive* if its graphs are models of ϕ, and *ϕ-negative* otherwise (or simply *positive* and *negative* when no confusion ensues). We might write π_H for $\pi_{H,m}$ when m is clear from the context.

Proof of Theorem 5.2.4. Fix an integer m, and a formula ψ of quantifier rank m. Since there are finitely many possible DULC m-profiles, we can denote $\{\pi_0, \dots, \pi_{k-1}\}$, for some integer k, the set of profiles of $\mathcal{E}_m(G)$, where G ranges over graphs satisfying ψ. Now let $\mathcal{E}_m(\psi)$ be the formula expressing "this graph has profile either π_0, or π_1, ..., or π_{k-1}."

The property "having profile π" is indeed expressible by a first-order formula: for all ball b, if $\pi(b) \neq \omega$ (respectively, $\pi(b) = \omega$), make a formula saying "there exist exactly $\pi(b)$ nodes (respectively, at least $m \cdot e + 1$ nodes) that are the center of a ball of type b." For a given ball type b, "being the center of a copy of b" is expressible as well: require that there exist $|b|$ distinct nodes, forming a cycle or a path (depending on b), with the right labels.

Now, if $G \models \psi$, by definition, $\{\pi_0, \dots, \pi_{k-1}\}$ contains the profile of $\mathcal{E}_m(G)$; thus $\mathcal{E}_m(G) \models \mathcal{E}(\psi)$. Conversely, if $\mathcal{E}_m(G) \models \mathcal{E}(\psi)$, then the profile of $\mathcal{E}_m(G)$ is the profile of some $\mathcal{E}_m(G')$, where $G' \models \psi$. By Lemma 5.2.8, $\mathcal{E}_m(G) \equiv_m \mathcal{E}_m(G')$, and by Theorem 5.2.3, $G \equiv_m G'$, so that $G \models \psi$. ◀

▶ **Proposition 5.2.9.** *If ϕ is an ω-nontrivial formula over DULC, then there is a nonempty DULC H and nonempty labeled cycles J' and D' such that either:*

(i) *for all $k \geq 0$ and $k' \geq 1$, we have $H \sqcup (\bigsqcup^k J') \models \phi$ and $H \sqcup (\bigsqcup^k J') \sqcup (\bigsqcup^{k'} D') \not\models \phi$; or*

(ii) *for all $k \geq 0$ and $k' \geq 1$, we have $H \sqcup (\bigsqcup^k J') \not\models \phi$ and $H \sqcup (\bigsqcup^k J') \sqcup (\bigsqcup^{k'} D') \models \phi$.*

Proof. Let m be the quantifier rank of ϕ. Since the profile of the disjoint union of two DULC is greater than either profile, there is a maximal DULC m-profile ρ. Assume that ρ is ϕ-negative (otherwise replace ϕ by $\neg\phi$). Since there are finitely many possible profiles and ϕ is ω-nontrivial, there is a positive profile having infinitely many models. Let π denote a maximal profile for this property.

If there is a cycle J' whose number of occurrences is unbounded among the models with profile π, then there is such a model H such that $\pi_{G'}(J') = \omega$, and $H \sqcup^k J'$ has the same profile as H for all k. If not, then the models with profile π have unbounded cycle lengths; so there is a model H and a word u over alphabet $\mathcal{E}_m$ of length $|\mathcal{E}_m|^e + 1$ such that u, as a path, occurs more than $m \cdot e$ times in H. For counting reasons, there is a word v of length e that occurs at least twice in u. So there is a cycle J' of length at least $e+1$ whose label (as a word) is a factor of u. The graph $H \sqcup^k J'$ has the same profile as H for all k.

Observe that there is no profile greater than $\pi = \pi_H$ with finitely many models, so by construction, any profile greater than π is negative. Let D'' be a ball such that $\pi(D'') < \rho(D'')$ and D' any cycle containing D''. For all $k > 0$, we have $\pi_H < \pi_{H \sqcup^k D'}$, so the DULC $H \sqcup^k D'$ is a countermodel of ϕ. Since $\pi_{H \sqcup^k J'} = \pi_H$, we have $\pi_{H \sqcup^k J' \sqcup^{k'} D'} = \pi_{H \sqcup^{k'} D'}$ for all k, k'. ◀

5.3 Proof of Theorem 5.2

We proceed to a case disjunction. In a graph G, a *hanging trees* is a connected component of the graph obtained from G by removing all the edges in cycles. A *subtree* of a tree T is always *complete*, i.e., spanned by the set of nodes coaccessible from a given node (the root of the subtree). An *immediate subtree* is a tree whose root has depth 1 in the ambient tree.

Unbounded cycles

▶ **Proposition 5.3.1.** *Let ψ denote a formula such that ψ and $\neg\psi$ both have models with unbounded cycles. Then ψ-Dynamics is either* NP*-hard or* co-NP*-hard.*

Proof. Since both ψ and $\neg\psi$ have models with unbounded cycles, the projection $\phi = \mathcal{E}(\psi)$ is ω-nontrivial. Apply Proposition 5.2.9 to get a nonempty DULC H and nonempty cycles D' and J' with either the property (i) or (ii) from the proposition. Let m be the quantifier rank of ϕ, and G, D, J be nonempty graphs such that $\mathcal{E}_m(G) = H$, that $\mathcal{E}_m(D) = D'$ and that $\mathcal{E}_m(J) = J'$. By Theorem 5.2.4, G, J, D satisfy the corresponding property (i) or (ii) (with G, J, D, ψ instead of H, J', D', ϕ), because $\mathcal{E}_m$ behaves correctly with respect to $\sqcup$: $\mathcal{E}_m(G) \sqcup \mathcal{E}_m(G') = \mathcal{E}_m(G \sqcup G')$. The statement follows from Proposition 5.1.2 with $z = 1$. ◀

Unbounded degrees

By Lemma 5.2.1, the set $\mathcal{E}_m$ of equivalence classes of $\equiv_m$ for trees is finite. If T is a tree and $\alpha \in \mathcal{E}_m$, write $|T|_\alpha$ for the number of immediate subtrees of T of type α.

▶ **Lemma 5.3.2.** *Let T and T' be trees such that, for each $\alpha \in \mathcal{E}_m$, we have either $|T|_\alpha = |T'|_\alpha$ or $|T|_\alpha, |T'|_\alpha \geq m$. Then $T \equiv_m T'$.*

Proof. We give a winning strategy for Duplicator. If Spoiler plays in a subtree t of T that was never touched before (the case of T' is symmetric), then Duplicator chooses a subtree t' of T' such that $t \equiv_m t'$. By the Ehrenfeucht-Fraïssé theorem, Duplicator has a winning strategy for the game t/t', so she uses it to play her turn. If Spoiler plays subsequent turns in t or t', then Duplicator continues the game in t/t' with her winning strategy. Since the global game lasts m turns, by the condition on T and T', it is always possible for Duplicator to find a t' such that $t \equiv_m t'$ as needed. Thus this is indeed a winning strategy. ◀

▶ **Proposition 5.3.3.** *Let ψ denote a formula whose models have bounded cycles but unbounded degrees. Then ψ-Dynamics is* NP*-hard.*

Proof. Let ψ be a formula of quantifier rank m, whose models have unbounded degrees and bounded cycles, say of length at most ℓ.

By assumption, ψ admits a model with a hanging tree having a node v of degree at least $m \cdot |\mathcal{E}_m|$. Hence, the node v has at least m equivalent immediate subtrees $J_1 \equiv_m \cdots \equiv_m J_m$. Lemma 5.3.2 implies that, if we add more copies of J_1 as immediate subtrees of v in G, resulting in a graph G', then $G \equiv_m G'$. So, in particular, G' also satisfies ψ. Let J denote $\bigsqcup^{\ell+1} J_1$, with the pointed nodes of J being the roots of the copies of J_1. Let D be a cycle of length $|J|$, without pointed nodes. We have $|D| = |J| > \ell$. For all k, k', with v the pointed node of G, the graph $G \sqcup_2 (\bigsqcup_2^k J)$ is a model of ψ, while $G \sqcup_2 (\bigsqcup_2^k J) \sqcup_2 (\bigsqcup_2^{k'} D)$ is a countermodel by assumption on l. The statement follows from Proposition 5.1.2, with G, J, D as defined above and $z = 2$. ◀

Unbounded subtree depths

▶ **Lemma 5.3.4.** *Let ψ denote a formula whose models have bounded cycles, degrees, but unbounded hanging tree depths. Then there are a model G of ψ and two subtrees T, T' of a hanging tree of G such that $T' \subset T$ and $T \equiv_m T'$.*

Proof. Suppose that the models have bounded cycles. By Lemma 5.2.1, $\mathcal{E}_m$ is finite. For any graph G, call $E_m(G)$ the $\mathcal{E}_m$-labeled copy of G where each node v is labeled by the equivalence class of the subtree rooted in v – the ambient trees being the hanging ones. By assumption, the graphs $E_m(G)$, for $G \models \psi$, contain arbitrarily deep subtrees, whilst the number of colors in $\mathcal{E}_m$ is fixed and finite. By the pigeonhole principle, one of those subtrees in one of those models admits two nodes with the same label, the first one being an ancestor of the other one. The lemma then follows from the definition of the labels. ◀

▶ **Lemma 5.3.5.** *Let T be a tree, t a subtree of T and t' a tree such that $t \equiv_m t'$. If T' is the tree T where the occurences of t have been replaced with t', then $T \equiv_m T'$.*

The proof goes by induction on the depth of the root of t in T, and Lemma 5.3.2.

▶ **Proposition 5.3.6.** *Let ψ denote a formula whose models have bounded degrees, but unbounded hanging tree depths. Then ψ-Dynamics is* NP*-hard.*

Proof. Let ψ be a formula of quantifier rank m, whose models have unbounded hanging tree depths, and bounded degrees, say by d.

By Lemma 5.3.4, there is a model $\widetilde{G}$ of ψ that contains a tree T (i.e. a subtree of a hanging tree), which in turn has a subtree T', such that $T' \equiv_m T$, and such that the only vertices of T and T' linked to the rest of the graph are their roots. Call T'' the tree $T \setminus T'$. Let G denote $(\widetilde{G} \setminus T') \sqcup T'$: it is a disconnected graph. We equip it with two pointed nodes, u and v, like in Definition 5.1.1 (case $\sqcup_3$). The pointed node u of G is the leaf of T that should have been the parent of the root of T'. The pointed node v of G is the root of the disconnected copy of T'. See Figure 3 for an illustration.

Now let $T''_0 = T''$, and T''_{n+1} denote the tree T where T' have been replaced by a copy of T''_n. For all n, equip the graph T''_n with two pointed nodes: the node u is the leaf where another copy of T'' would be inserted to build T''_{n+1}; the node v is the root. See again Figure 3 for an illustration.

Let J be the graph T''_{d+2}, so that $|J| > d+1$. By Lemma 5.3.5, we have $G \sqcup_3 (\bigsqcup_3^k J) \equiv_m \widetilde{G}$ for all integer k. Let D be a tree of depth 1 having $|J|$ nodes, i.e., it consists only of a root and its direct children; its pointed node u is any leaf, and its pointed node v is the root. The tree D has degree at least $d+1$. Therefore, for all k, k', the graph $G \sqcup_3 (\bigsqcup_3^k J)$ is a model of ψ, while the graph $G \sqcup_3 (\bigsqcup_3^k J) \sqcup_3 (\bigsqcup_3^{k'} D)$ is a countermodel. The statement follows by Proposition 5.1.2 with $z = 3$. ◀

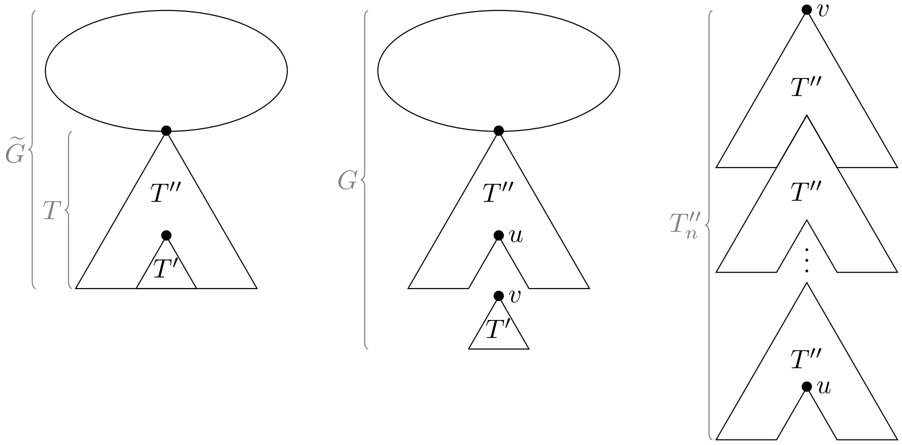


Figure 3 Illustration of the construction in the proof of Proposition 5.3.6. The node v of each graph transitions to the node u of another one.

Unbounded number of occurrences of each connected component

Here, *connected* means *strongly connected.* The *number of occurrences* of a connected component C in a graph G is the number of connected components of G isomorphic to C.

▶ **Lemma 5.3.7.** *Let G and J be graphs and m an integer. For all integers $k, k' \geq m$, we have $G \sqcup (\bigsqcup^k J) \equiv_m G \sqcup (\bigsqcup^{k'} J)$.*

Proof of Lemma 5.3.7. We give a winning strategy for Duplicator. If Spoiler plays in either copy of G, then Duplicator picks the same node in the other copy of G. If Spoiler plays in a copy of J that was never touched before, then Duplicator chooses a fresh copy of J in the other graph and picks the same node there. If Spoiler plays in a copy of J that was already touched before, then Duplicator chooses the same copy of J as in the previous moves and picks the same node there. Since there are at least m copies of J on both graphs and only m turns in the game, this is indeed a winning strategy. ◀

▶ **Proposition 5.3.8.** *Let ψ is a formula whose models have bounded cycles, but unbounded number of occurrences of each connected component. Then ψ-Dynamics is* NP*-hard.*

Proof. Let ψ be a formula of quantifier rank m, whose models have unbounded number of occurrences of each connected component, and bounded cycles, say of length at most ℓ. By our assumptions on ψ, there are graphs G and J' such that $G \sqcup (\bigsqcup^m J')$ is a model. Let J

denote $\bigsqcup^{\max(\ell+1,m)} J'$, and D a cycle of length $|J|$. For all k, k', by Lemma 5.3.7, the graph $G \sqcup (\bigsqcup^k J) = G \sqcup (\bigsqcup^{k\cdot\max(\ell+1,m)} J')$ is a model of ψ. On the other hand, by assumption on $\ell < |J|$, the graph $G \sqcup (\bigsqcup^k J) \sqcup (\bigsqcup^{k'} D)$ is a countermodel.

The statement follows from Proposition 5.1.2 with G, J, D defined above and $z = 1$. ◀

Combining the cases

▶ **Lemma 5.3.9.** *Every formula with infinitely many models has models with either unbounded cycles, unbounded degrees, unbounded hanging tree depths, or an unbounded number of occurrences of each connected component.*

Proof. The number of nonisomorphic connected graphs with a cycle of length at most ℓ, degree at most d and hanging tree depth at most h is bounded by ł $\cdot\, d^{h+1}$. Thus there are only finitely many graphs with bounded cycles, degrees, subtree depths and number of occurrences of connected components. ◀

Lemma 5.3.9 concludes the proof of Theorem 5.2: if the formula and its negation both have unbounded cycles, then Proposition 5.3.1 applies; otherwise one among Propositions 5.3.3, 5.3.6 and 5.3.8 applies to either the formula or its negation.

▶ **Remark 5.3.10.** The machinery developed to prove Theorem 5.2 is rather flexible.

In particular, it remains true if restricted to deterministic automata networks, and also to the limit subgraphs ($\mathcal{G}_f^\omega$) instead of transition digraphs ($\mathcal{G}_f$). However, the meaning of "ω-nontrivial" changes: it respectively means "having infinitely many bijective (counter)models" and "having infinitely many networks whose limit graph is a (counter)model."

Both transition digraphs of bijective networks and limit graphs are merely disjoint unions of unlabeled cycles. Thus, Proposition 5.2.9 and Proposition 5.1.2 may be reused and the proof is similar.

6 First-order dynamical properties are arbitrarily high in PH

The previous section gave a lower bound for the ψ-Dynamics problem. Here, we give tighter bounds. As a consequence of those bounds, the AN-Dynamics problem, which is similar to ψ-Dynamics except that ψ *is part of the input,* is hard.

▶ **Theorem 6.1.** *For all even integer N, there is a formula ψ_N such that ψ_N-Dynamics Σ_{N+1}-complete.*

▶ **Theorem 6.2.** *The following problem is* PSPACE*-complete:*

AN-Dynamics
Input: an automata network f and a first-order formula ψ.
Question: does $\mathcal{G}_f \models \psi$?

The proofs rely on the following constructions. Let $N \geq 1$, and S be a QBF formula of the form $\exists b_1, \forall b_2, \ldots, \exists b_{N+1} R(b_1, \ldots, b_{N+1})$. Call C the set $\{\top, \bot\} \sqcup \bigsqcup_{i=1}^{N+1} \{0,1\}^i$, where $\top, \bot$ are fresh symbols. Observe that $|C| = 2^{N+2}$ and define f_S the ANU with $N+2$ nodes over alphabet $\{0,1\}$ that realizes the function $f_S : C \to C$ defined as follows. For arbitrary bits $b_1, \ldots, b_{N+1}$, set $f_S(\bot) = \bot$, $f_S(\top) = \top$, $f_S((b_1)) = \top$, and:

$$f_S((b_1,\ldots,b_i)) = (b_1,\ldots,b_{i-1}) \quad f_S((b_1,\ldots,b_{N+1})) = \begin{cases} (b_1,\ldots,b_N) \text{ if } R(b_1,\ldots,b_{N+1}) \\ \bot \text{ otherwise.} \end{cases}$$

Intuitively, the dynamics of f_S consists of two upward trees: one rooted in $\top$, of depth $N+1$, whose leaves are the Boolean tuples $(b_1, \dots, b_{N+1})$ that satisfy R; and one rooted in $\bot$, of depth 1, whose leaves are the Boolean tuples $(b_1, \dots, b_{N+1})$ that falsify R. The only part of f_S that depend on S merely evaluates R, so circuits encoding f_S can be produced in polynomial time given S.

Now define the formula ψ_N as follows (observe that ψ_N depends only on N):

$$\begin{aligned}\psi_N =& \exists x_0, x_1, x_2' : x_0 \neq x_1 \wedge x_2' \to x_1 \to x_0 \to x_0 \\ & \wedge \forall x_2 \to x_1 : \exists x_3 \to x_2 : \dots \forall x_N \to x_{N-1} : \exists x_{N+1} \to x_N : \mathsf{true},\end{aligned}$$

where "$\exists x \to y : \phi$" and "$\forall x \to y : \phi$" stand for "$\exists x : (x \to y) \wedge \phi$"and "$\forall x : (x \to y) \implies \phi$"; and where "$x \to y \to z$" stands for "$x \to y \wedge y \to z$". Observe that ψ_N is a Σ_{N+1}-formula. Besides, when evaluating ψ_N in $\mathcal{G}_{f_S}$, the first line ensures that x_0 is a fixed point with an ingoing path of length 2, so it has to be $\top$. The rest of the formula straightforwardly implements S, by linking Booleans into a configuration $(b_1, \dots, b_{N+1})$ where the "$x_{N+1} \to x_N$" part ensures that $R(b_1, \dots, b_{N+1})$, by definition of f_S. Hence we have the following lemma, that implies both Theorem 6.1 and 6.2.

▶ **Lemma 6.3.** **(a)** *The network f_S satisfies $\mathcal{G}_f \models \psi_N$ if and only if S is a true QBF.*
(b) *Given a* $\mathrm{QBF}(\Sigma_{N+1})$ *formula S, the network f_S can be produced in polynomial time.*

7 Conclusion

Our goal was to obtain broad complexity lower bounds for dynamical properties of automata networks. However, as explained in the introduction, there is a large degree of freedom in the formalization of Metatheorem 1.2. We do not claim that the results above are the only Rice-like theorems on automata networks worth investigating.

It would be interesting to know how various restrictions on the AN may lessen the complexity of those problems. For instance, if we restrict ourselves to AN whose interaction graph has bounded degree, then the question "does this AN compute a constant function?" becomes testable in polynomial time, while it is first-order expressible and nontrivial.

Another restriction pertains to the set of states of the nodes. If we restrict ourselves to ANU, i.e., automata networks where all nodes have the same alphabet $Q = \{0, \dots, q-1\}$ for some positive integer q, then the concept of "ω-nontriviality" changes. Indeed, some first-order formulas have infinitely many models and countermodels, but no model with uniform alphabet. The proof of Lemma 5.1.4 does not seem to generalize easily to that case, because finding x and δ becomes an open challenge.

References

1 J. Aracena. Maximum number of fixed points in regulatory Boolean networks. *Bull. Math. Biol.*, 70:1398–1409, 2008.

2 B. Borchert and F. Stephan. Looking for an analogue of Rice's theorem in circuit complexity theory. *Mathematical Logic Quarterly*, 46(4):489–504, 2000. doi:10.1002/1521-3870(200010)46:4<489::AID-MALQ489>3.0.CO;2-F.

3 P. Cull. Linear analysis of switching nets. *Biol. Cybernet.*, 8:31–39, 1971.

4 J. Demongeot, M. Noual, and S. Sené. Combinatorics of Boolean automata circuits dynamics. *Discr. Appl. Math.*, 160:398–415, 2012.

5 H.-D. Ebbinghaus and J. Flüm. *Finite Model Theory.* Springer-Verlag, 2nd edition, 1995. doi:10.1007/3-540-28788-4.

6 B. Elspas. The theory of autonomous linear sequential networks. *IRE Trans. Circ. Theory*, 6:45–60, 1959.

7 C. Espinosa-Soto, P. Padilla-Longoria, and E. R. Alvarez-Buylla. A gene regulatory network model for cell-fate determination during *Arabidopsis thaliana* flower development that is robust and recovers experimental gene expression profiles. *The Plant Cell*, 16:2923–2939, 2004.

8 E. Goles and S. Martinez. *Neural and Automata Networks: Dynamical Behavior and Applications*. Kluwer Academic Publishers, 1990.

9 S. W. Golomb. *Shift Register Sequences*. Holden-Day Inc., 1967.

10 W. Hanf. Model-theoretic methods in the study of elementary logic. In J.W. Addison, L. Henkin, and A. Tarski, editors, *The Theory of Models*, pages 132–145. North-Holland, 1963. doi:10.1016/B978-0-7204-2233-7.50020-4.

11 N. Immerman. *Descriptive Complexity*. Springer-Verlag, 1999. doi:10.1007/978-1-4612-0539-5.

12 J. Kari. Rice's theorem for the limit sets of cellular automata. *Theoretical Computer Science*, 127:229–254, 1994.

13 G. Karlebach and R. Shamir. Modelling and analysis of gene regulatory networks. *Nature Rev. Mol. Cell Biol.*, 9:770–780, 2008.

14 S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22:437–467, 1969. doi:10.1016/0022-5193(69)90015-0.

15 S. C. Kleene. *Automata Studies*, chapter Representation of events in nerve nets and finite automata, pages 3–41. Princeton University Press, 1956.

16 W. S. McCulloch and W. H. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.*, 5:115–133, 1943.

17 L. Mendoza and E. R. Alvarez-Buylla. Dynamics of the genetic regulatory network for *Arabidopsis thaliana* flower morphogenesis. *J. Theoret. Biol.*, 193:307–319, 1998.

18 H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74:358–366, 1953. doi:10.1090/S0002-9947-1953-0053041-6.

19 A. Richard. Local negative circuits and fixed points in non-expansive Boolean networks. *Discr. Appl. Math.*, 159:1085–1093, 2011.

20 F. Robert. *Discrete Iterations: A Metric Study*. Springer Verlag, 1986.

21 J. H. Silverman. *A friendly introduction to number theory*. Pearson Education, 4th edition, 2012.

22 L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time(preliminary report). In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC '73, pages 1–9, New York, NY, USA, 1973. ACM. doi:10.1145/800125.804029.

23 D. Thieffry and R. Thomas. Dynamical behaviour of biological regulatory networks – II. Immunity control in bacteriophage lambda. *Bull. Math. Biol.*, 57:277–297, 1995.

24 R. Thomas. Boolean formalization of genetic control circuits. *Journal of Theoretical Biology*, 42:563–585, 1973. doi:10.1016/0022-5193(73)90247-6.

Auction Algorithms for Market Equilibrium with Weak Gross Substitute Demands and Their Applications

Jugal Garg
University of Illinois at Urbana-Champaign, IL, USA

Edin Husić
Department of Mathematics, The London School of Economics and Political Science, UK

László A. Végh
Department of Mathematics, The London School of Economics and Political Science, UK

Abstract

We consider the Arrow–Debreu exchange market model where agents' demands satisfy the weak gross substitutes (WGS) property. This is a well-studied property, in particular, it gives a sufficient condition for the convergence of the classical tâtonnement dynamics. In this paper, we present a simple auction algorithm that obtains an approximate market equilibrium for WGS demands. Such auction algorithms have been previously known for restricted classes of WGS demands only. As an application of our technique, we obtain an efficient algorithm to find an approximate spending-restricted market equilibrium for WGS demands, a model that has been recently introduced as a continuous relaxation of the Nash social welfare (NSW) problem. This leads to a polynomial-time constant factor approximation algorithm for NSW with budget separable piecewise linear utility functions; only a pseudopolynomial approximation algorithm was known for this setting previously.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Algorithmic game theory and mechanism design

Keywords and phrases auction algorithm, weak gross substitutes, Fisher equilibrium, Gale equilibrium, Nash social welfare

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.33

Related Version *Full Version*: https://arxiv.org/abs/1908.07948

Funding Jugal Garg was supported by the NSF CAREER Award 1942321. Edin Husić and László A. Végh were supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement ScaleOpt–757481).

Acknowledgements We would like to thank anonymous referees for their comments and suggestions that have helped to improve the presentation of the paper.

1 Introduction

Market equilibrium is a fundamental and well-established notion to analyze and predict the outcomes of strategic interaction in large markets. In the classic Arrow-Debreu exchange model, a set of agents arrive at the market with initial endowments of divisible goods. A market equilibrium comprises a set of prices and allocations of goods to the agents such that each agent spends their income from selling their initial endowment on a bundle that maximizes their utility, and the market clears: demand of each good meets its supply. This model was first studied by Walras in 1874 [61], who also introduced a natural market dynamics, called the *tâtonnement* process. A continuous version of the process was shown to converge to an equilibrium if the utility functions satisfy the *weak gross substitutability (WGS)* property, namely, that if the prices of some goods increase and the others remain unchanged, then the demand for the latter goods may not decrease (see Arrow, Block, and Hurwitz [3], Arrow and Hurwitz [6], and references therein). However, Scarf [59] showed, using an example of Leontief utilities, that tâtonnement may not always converge to an equilibrium. We refer the reader to [54, Chapter 17] on the stability of the tâtonnement process.

The polynomial-time computability of market equilibrium for WGS utilities was first established by Codenotti, Pemmaraju, and Varadarajan [25]. Later, a simple ascending-price algorithm using *global demand queries* was given by Bei, Garg, and Hoefer [9]. Further, Codenotti, McCune, and Varadarajan [23] have shown that a simple discrete variant of the tâtonnement algorithm converges to an approximate equilibrium (see also [57, Section 6.3]). This was followed by a number of papers providing tâtonnement algorithms for various classes of utility functions and restricted models, some of them substantially weakening the need for central coordination among agents, see e.g., [7, 19, 20, 27, 37].

However, most of these algorithms still rely on global demand queries, and hence they are less realistic. In a sense, they require a central authority (responsible for updating prices) to have some general information about the demands of all agents in the market.

Auction algorithms. In this paper, we focus on an even simpler subclass of tâtonnement-type algorithms, called *auction algorithms*. Whereas prices in tâtonnement may increase as well as decrease, in auctions prices may only go up. Auction algorithms are appealing due to their simplicity and distributed nature: under simple "ground rules" the agents outbid each other and in the process converge to an approximate market equilibrium. Unlike the above mentioned works, these algorithms do not require a central authority and need only minimal coordination between the agents. Further, these algorithmic frameworks are quite robust and easily allow for various extensions and generalizations. For exchange market models, the first such algorithm was established for linear utilities by Garg and Kapoor [44] (see also [57, Section 5.12]). The algorithm was later improved [45] and generalized to separable concave gross substitute utility functions [47], to a subclass of non-separable gross-substitutes called *uniformly separable* [46], and to a production model with linear production constraints and linear utilities [50].

There is a long history of auction algorithms both in the optimization and in the economics literature. Bertsekas [11, 12] introduced auction algorithms for assignment and transportation problems. Closely related algorithms were introduced for markets with indivisible goods, by Kelso and Crawford [52], and Demange, Gale, and Sotomayor [30]. We discuss markets with indivisible goods later in this section.

Our contributions. Our first main contribution is an auction algorithm that computes an approximate market equilibrium for arbitrary WGS utilities, given via demand oracles, settling an open question from [46]. This result shows that for WGS utilities, this restricted class of tâtonnement algorithms already suffices to obtain an equilibrium. The result affirms the natural intuition that the WGS property is geared for auction algorithms. A main invariant in auction algorithms is that at every price increase, the agents will still hold on to the goods they have purchased previously at the lower prices. This property is almost identical to the definition of the WGS property; nevertheless, making an auction algorithm work for general WGS utilities requires some careful technical ideas. The previously mentioned auction algorithms operate with two prices for each good, a lower price p_j and a higher price $(1+\epsilon)p_j$. For linear utilities, [44] maintains that all purchases are maximum bang-per-buck goods with respect to the lower or higher price. This idea can be extended to separable [45] and to uniformly separable utilities [47], but does not work if the utilities are genuinely non-separable. For this general case, our main technical idea is to maintain subsets of optimal bundles for each agent with respect to some individual prices. These individual prices can be different for each agent but fall between the higher and lower prices p and $(1+\epsilon)p$.

This results in the first "agent-driven" algorithm for the entire range of WGS utilities that avoids the need of a central authority, where each agent uses only their own black-box oracle `FindNewPrices` (Section 3), which depends only on their own preference to *outbid* another agent on a particular good. The process of outbidding another agent can also be implemented in an uncoordinated manner. Overall, this lessens the level of coordination needed in the market, making it more plausible mechanism in a decentralized environment.

We also study auction algorithms for multiple models of *Fisher markets*. These are a special case of exchange markets where every agent arrives with a fixed budget instead of an endowment of goods. A particular motivation comes from recent study of the *Nash social welfare (NSW)* problem: allocating indivisible goods to agents so that the geometric mean of their utilities is maximized. This problem is NP-hard already for simple classes of utilities, and there has been a considerable recent literature on approximation algorithms for the problem and its extensions. Cole and Gkatzelis [28] gave the first constant-factor approximation for linear utilities, followed by further work with stronger guarantees as well as extensions for other utility classes [1, 2, 8, 17, 26, 28, 38, 40, 41].

The algorithm in [28] and many others start by studying a continuous relaxation corresponding to a specific market equilibrium problem with *spending restrictions*: namely, if the price p_i of good i is above 1, then the amount of good i sold is decreased to $1/p_i$ from the initial total amount of 1. Whereas a market equilibrium with spending restrictions can be obtained via a convex program for linear utilities [26], it becomes challenging to find for more general utilities: currently known cases are budget-additive valuations [38] and separable piecewise-linear concave (SPLC) utility functions [2]. The set of equilibria in the former case turned out to be not even convex.

In this paper, we show that auction algorithms are particularly well-suited for spending restricted equilibrium computation: once the price of a good goes above one, we can naturally decrease the total available amount of these goods within the auction framework. This enables us to find simple approximation algorithms for spending restricted equilibria for a broad class of utility functions, including the models above as well as their common generalization: *budget-SPLC*. A surprising feature here is that we do not even have to make the standard non-satiation assumption. Moreover, our algorithm can be used to obtain a constant-factor approximation for maximizing NSW in polynomial-time when agents have budget-SPLC utilities and goods come in multiple copies. The previous algorithm for this setting in [17] runs in pseudopolynomial time. We expect that our algorithm for finding approximate spending restricted equilibria will find more applications for the NSW and other related problems.

Markets with indivisible goods. Auction algorithms have been widely studied in the context of markets with *indivisible goods*. Equilibria may not always exists in markets with indivisible goods. The class of (discrete) gross substitute utilities was introduced by Kelso and Crawford [52]. For this class, an equilibrium is guaranteed to exist, and an approximate equilibrium can be efficiently found via a simple auction algorithm, extending [29]. It turned out that the discrete gross substitutes property is essentially a necessary and sufficient condition for the auction algorithm to work. We refer the reader to the survey by Paes Leme [53] on the role of gross substitute utilities in markets with indivisible goods, and their connections to discrete convex analysis.

Whereas the definitions of discrete gross substitutes and continuous WGS utilities are very similar, there does not appear to be a direct connection between these notions. The main difference is in the utility concepts: for indivisible markets, the standard model is to maximize the valuation minus the price of the set at given prices, whereas the standard divisible market models operate with *fiat money*: the prices appear via the budget constraints but not in the utility value. Still, our result can be interpreted as the continuous analogue of the strong link between auction algorithms and the gross substitutes property for markets with indivisible goods: we show that auction algorithms are applicable for the entire class of WGS utilities for markets with divisible goods. We suspect that the converse should also be true, namely, that the applicability of auction algorithms should be limited to WGS utilities. In contrast, tâtonnement algorithms have been successfully applied beyond the WGS class, see e.g. [19, 20, 37].

Let us also comment on the oracle model we use. Typically, (continuous) WGS utilities in the literature are given in an explicit form such as CES or Cobb-Douglas utilities. This is in contrast with the discrete WGS setting, where the common model is via a value or demand oracle [53], since direct preference elicitation, that is, the explicit description of the valuation function would be exponential. The class of continuous WGS functions also appears to be very rich and expressive, and hence an oracle approach seems more appropriate to devise algorithms for this class. In our model, the agent preferences are represented via a *demand oracle* (Definition 3).

The auction algorithm relies on the more powerful `FindNewPrices` subroutine, which can be seen as a strengthening of the demand oracle, incorporating a mechanism for price increments. There are various ways to implement such a subroutine: we use a simple iterative application of the demand oracle for the case of bounded price elasticities; we use a convex programming approach for Gale demand systems; and we devise a combinatorial algorithm for budget-additive SPLC utilities.

Further related work. The existence of a market equilibrium is always guaranteed under some mild assumptions, as shown by Arrow and Debreu [4], using Kakutani's fixed point theorem. The computational aspects of finding a market equilibrium have been extensively studied in the theoretical computer science community over the last two decades, establishing hardness results as well as polynomial-time algorithms for certain cases. We refer the reader to [14, 18, 24, 31, 34, 42, 49, 60, 63, 43] for an overview of the literature.

The other famous dynamics to study market equilibrium is *proportional response* where in each round agents bid on goods in proportional to the utility they receive from them in the previous round. The goods are then allocated in proportion of the agents' bids. It has been shown that proportional response converges to market equilibrium in a variety of Fisher markets [13, 21, 22, 64], and some special cases of exchange markets [15, 16, 62].

The rest of the paper is structured as follows. Section 2 defines the exchange market model and provides examples of WGS demand systems. Section 3 presents the auction algorithm for exchange markets. Section 4 discusses the applicability of the algorithm to the Fisher market model, spending restricted equilibrium, Gale demand systems, and the NSW problem. Several proofs and some significant arguments are deferred to the Appendices, as indicated at the respective parts. For missing proofs and other details, we refer the reader to the full version [39].

2 Models and concepts

Notation $[k]$ denotes $\{1, 2, \dots, k\}$, and $\mathbb{1}^k$ denotes the k dimensional vector with all entries 1. We use $\mathbb{1}$ if the dimension is clear from the context. We consider an *exchange market* with a set of agents $A = [n]$ and divisible goods $G = [m]$. Each agent $i \in [n]$ arrives at the market with an initial endowment of goods $e^{(i)} \in \mathbb{R}^m_+$. Thus, the total amount of good $j \in [m]$ is e_j where $e = \sum_{i=1}^n e^{(i)}$; w.l.o.g. $e_j > 0$. Given a non-negative price vector $p \in \mathbb{R}^m_+$, the budget of agent i at prices p is defined as $b_i = b_i(p) = p^\top e^{(i)}$. It follows that $p^\top e = \sum_i p^\top e^{(i)} = \sum_i b_i$.

We now define the market equilibrium using *demand systems.* A *bundle* x is a non-negative vector $x \in \mathbb{R}^m_+$. A *demand system* is a function $D : \mathbb{R}^{m+1}_+ \to 2^{\mathbb{R}^m_+}$; $D(p, b)$ denotes the set of preferred bundles of an agent at prices p and budget b. Bundles in $D(p, b)$ are called the *optimal* or *demand* bundles at prices p and budget b. This corresponds to the standard concept of a demand function, except that we do not assume the uniqueness of a preferred bundle. For example, in case of a linear utility function $u(x) = \sum_{j\in G} v_j x_j$, $D(p, b)$ includes all fractional assignment of goods maximizing v_j/p_j with a total price b. If $|D(p, b)| = 1$ for all $(p, b) \in \mathbb{R}^{m+1}$ we say that D is *simple*, and use $D(p, b)$ to denote the unique bundle. We include the budget b in the definition of the demand system, even though for exchange markets the budget of agent i is uniquely defined by the prices as $p^\top e^{(i)}$. This formalism will be useful for our algorithm where the budgets are defined according to a slightly different set of prices.

▶ **Definition 1** (Market equilibrium). *Let D_i denote the demand system of agent $i \in A$. We say that the prices $p \in \mathbb{R}^m_+$ and bundles $x^{(i)} \in \mathbb{R}^m_+$ form a* market equilibrium *if (i) $x^{(i)} \in D_i(p, p^\top e^{(i)})$, and (ii) $\sum_{i=1}^n x^{(i)}_j \le e_j$, with equality whenever $p_j > 0$, for all $j \in G$.*

That is, p and optimal bundles $x^{(i)}$ form an equilibrium if no good is overdemanded and goods at a positive price are fully sold. Note that this implies that every agent fully spends their budget.

▶ **Definition 2.** *Let $(p, b) \in \mathbb{R}^{m+1}_+$ and $x \in D(p, b)$. If for any $p' \ge p$ and $b' \ge b$ there exists $y \in D(p', b')$ such that $y_j \ge x_j$ whenever $p'_j = p_j$, we say that the demand system D satisfies the* weak gross substitutes (WGS) *property.*

We will also say that $D(p, b)$ is a WGS demand system. In the context of the tâtonnement process, the weak gross substitutes property is usually defined with respect to the *aggregate* excess demand function of all agents. We use the stronger requirement of having a WGS demand system for each individual agent. The previous auction algorithms [46, 47] have also used WGS on the level of agents as this seems to be the necessary condition that allows agents to update their bundles individually, as opposed to tâtonnement, where the prices adjustments react to the aggregate demands.

▶ **Definition 3** (Demand oracle). *For a WGS demand system $D(p, b)$, a* demand oracle *requires two vectors $(p, b), (p', b') \in \mathbb{R}^{m+1}_+$ such that $(p', b') \ge (p, b)$, and a vector $x \in D(p, b)$. The output is a vector $y \in D(p', b')$ such that that $y_j \ge x_j$ whenever $p'_j = p_j$.*

In other words, the oracle provides the allocations guaranteed by the definitions of WGS systems. The complex form of the definition is due to the possible non-uniqueness of demand bundles. For simple demand systems, the input to the oracle is simply a vector $(p', b') \in \mathbb{R}^{m+1}_+$, and the output is the unique vector $y \in D(p', b')$.

For exchange markets, we will make the following assumptions:

▶ **Assumption 4** (Scale invariance). *For every agent i, $D_i(p, b_i) = D_i(\alpha p, \alpha b_i)$ for all $\alpha > 0$.*

▶ **Assumption 5** (Non-satiation). *For all demand systems, and for every $(p, b) \in \mathbb{R}_+^{m+1}$, and every $x \in D(p, b)$, we have $p^\top x = b$.*

In scale invariance, we require that the demand is homogeneous of degree 0; informally, the demand does not depend on the currency. This is a standard assumption in microeconomics and exchange markets, see e.g. [5, 33, 35, 55].

Non-satiation states that in every optimal bundle the agents must fully spend their budgets. This is a standard assumption for exchange markets as it is necessary for the fundamental theorems of welfare economics (see e.g. [54, Chapter 16]). However, we note that we do not require this assumption for spending restricted Fisher markets.

Approximate equilibria. We define the concept of an ϵ-equilibrium in exchange markets that our algorithm finds. We require that each agent gets an approximate optimal bundle and market clears approximately.

▶ **Definition 6** (Approximate equilibrium). *For an $\epsilon > 0$, the prices $p \in \mathbb{R}^m$ and bundles $x^{(i)} \in \mathbb{R}_+^m$ form an ϵ-*approximate market equilibrium *if*

(i) $x^{(i)} \le z^{(i)}$ *for some* $z^{(i)} \in D_i(p^{(i)}, p^\top e^{(i)})$, *where* $p \le p^{(i)} \le (1+\epsilon)p$,

(ii) $\sum_{i=1}^n x_j^{(i)} \le e_j$, *and*

(iii) $\sum_{j=1}^m p_j \left(e_j - \sum_{i=1}^n x_j^{(i)}\right) \le \epsilon p^\top e$.

That is, every agent owns a subset of their optimal bundle at prices that are within a factor $(1+\epsilon)$ from p, and all goods are nearly sold: the value of the unsold goods is at most an ϵ fraction of the total value of the goods. The total value of the goods "taken away" from the near-optimal bundles of the agents is $\sum_{i=1}^n p^\top (z^{(i)} - x^{(i)})$. Parts (i) and (iii), together with the fact that ${p^{(i)}}^\top z^{(i)} \le p^\top e^{(i)}$ for all i, imply that this amount is $\le 2\epsilon p^\top e$.

The definition (i) can be seen as a natural extension of the corresponding approximate optimality conditions in [44, 46, 47]. For linear utilities, [44] requires the approximate maximum bang-per-buck condition $v_{ij}/p_j \le (1+\epsilon)v_{ik}/p_k$ for any agent i, goods j and k such that $x_{ik} > 0$. Thus, one can set approximate prices $p \le p^{(i)} \le (1+\epsilon)p$ for each agent for which they purchase maximum bang-per-buck goods.

Condition (iii) corresponds to the definition of approximate equilibrium in [32] and [48]. This notion is weaker than the ones used in [44, 46, 47]. The most important difference is that the latter papers guarantee that each agent recovers approximately their optimal utility. Such a property could be achieved by strengthening the bound in (iii) from $\epsilon p^\top e$ to $\epsilon p_{\min} e_{\min}$, where $p_{\min}$ is the minimum price and $e_{\min}$ is the smallest total fractional amount in the initial endowment of any agent. However, this would come at the expense of substantially worse running time guarantees in our algorithmic framework.

2.1 Examples of WGS demand systems

A standard way to implement a demand oracle is via an explicitly given utility function. Assume the agent is equipped with a concave utility function $u : \mathbb{R}_+^m \to \mathbb{R}_+$. The set of demand bundles at prices p and budget b it given as the set of optimal solutions of

$$\max \quad u(x) \quad \text{s.t.} \quad p^\top x \le b; \quad x \ge 0\,. \tag{1}$$

Then, $D(p, b) := D^u(p, b) = \arg\max_{x \in \mathbb{R}_+^m} \{u(x) : p^\top x \le b\}$. We say that a utility function is WGS if the corresponding demand system is WGS. Most models studied in the literature assume strictly concave utilities and thus have a unique optimal solution; a notable exception is the case of linear utility functions. If the solution is not unique, we can implement the demand oracle for inputs $(p,b),(p',b')$ and $x \in D(p,b)$ by imposing the constraints that $u(y)$ equals the optimal utility in $D(p',b')$, and $y_i \geq x_i$ for every i with $p'_i = p_i$. Thus, the optimal demand system can also be implemented via convex programming (we now ignore the question of numerical precision).

We now present some classical examples of WGS utilities studied in the literature:

- For $v \in \mathbb{R}^m_+$ the *linear (additive) utility* is given by $u(x) = v^\top x$. Then, $D^u(p,b) = \arg\max\{v^\top x : p^\top x \leq b\}$.
- The *constant elasticity of substitution (CES)* utility is defined by $u(x) = \left(\sum_j \beta_j^{\frac{1}{\sigma}} x_j^{\frac{\sigma-1}{\sigma}}\right)^{\frac{\sigma}{\sigma-1}}$, where $\sum_j \beta_j = 1$. Then, $D(b,p) = \{x\}$ for the unique optimal bundle x given by $x_j = \dfrac{\beta_j p_j^{-\sigma} b}{\sum_k \beta_k p_k^{1-\sigma}}$. It is well-known that CES demand system satisfies the WGS property iff $\sigma > 1$.
- The *Cobb-Douglas* utility function is given by $u(x) = \prod_j x_j^{\alpha_j}$ where $\sum_j \alpha_j = 1$, $\alpha \geq 0$. The unique optimal bundle is therefore $x_j = b\alpha_j / p_j$ and $D^u(p,b) = \{x\}$. The Cobb-Douglas utility function satisfies the WGS property for any parameter choices.
- The *nested CES* utility function is defined recursively (see [49]). Any CES function is a nested CES function. If $g, h_1, \ldots, h_t$ are nested CES functions, then $f(x) = \max g(h_1(x^1), \ldots, h_t(x^t))$ over all $x^1, \ldots, x^t$ such that $\sum_{k=1}^t x^k = x$, is a nested CES function. In a well-studied special case (see e.g., [51]), each good j can only be used in at most one of the h_i's.

Conic combinations of demand systems. Given two WGS utility functions u and u', the demand system corresponding to their sum $u + u'$ may not be WGS. On the other hand, consider two simple WGS demand systems D and D' and nonnegative coefficients λ, λ'. Then it is easy to see that $\lambda D + \lambda' D'$ is also a simple WGS demand system. This enables the construction of some interesting demand systems. For example, [55] has studied hybrids of CES and Cobb-Douglas demands, where the demand system is given as a conic combination of the two. [1]

$$x_j = \frac{b}{p_j}\left[\epsilon\alpha_j + (1-\epsilon)\frac{\beta_j p_j^{1-\sigma}}{\sum_k \beta_k p_k^{1-\sigma}}\right], \text{ for some } 0 \leq \epsilon \leq 1 \text{ and } \sigma > 1 \ .$$

Note that if $D = D^u$ and $D' = D^{u'}$ for some concave utility functions u and u', the demand system $\lambda D + \lambda' D'$ in general does *not* correspond to the utility function $\lambda u + \lambda' u'$. In fact, it is unclear if one can find explicitly utility functions corresponding to such conic combinations. Our model does *not* require the demand system to be given in the form $D = D^u$ for some function u.

Price elasticity of demands. One possible implementation of the key subroutine `FindNewPrices` (Section 3) relies on the *(price) elasticity of the demands.*[2] The standard definition of the elasticity for good j with respect to the price of good k is $e_{j,k} = \partial \log x_j(p,b) / \partial \log p_k$, where $x_j(p,b)$ is the (unique) demand for good j at prices p and budget b. The WGS

[1] We note that this demand function does not seem to correspond to a nested CES utility function.

[2] No finite lower bound exists on the elasticity of linear demand systems. If we are buying a positive amount of good j, then j maximizes v_k/p_k. If there is another good ℓ with $v_j/p_j = v_\ell/p_\ell$, then if we increase p_j but leave the other prices unchanged, then $x'_j = 0$ for every optimal bundle x' w.r.t. the new prices. Consequently, for this case, we have another way to implement `FindNewPrices`.

property guarantees that $e_{j,k} \geq 0$ if $j \neq k$, and consequently, $e_{k,k} \leq 0$. The definition below corresponds to $e_{k,k} \geq -f$ for all $k \in [m]$, for the more general model of non-simple demand systems.

▶ **Definition 7.** *Consider a WGS demand system $D(p,b)$. For some $f > 0$, we say that the* elasticity of $D(p,b)$ is at least $-f$, *if for any $\mu \geq 0$, $j \in [m]$, $(p,b) \in \mathbb{R}_+^{m+1}$ and $x \in D(p,b)$, if we define p' as $p'_j = p_j(1+\mu)$ and $p'_k = p_k$ for $k \in [m] \setminus \{j\}$, then there exists a bundle $x' \in D(p',b)$ such that $x'_j \geq \frac{1}{(1+\mu)^f} x_j$.*

In can be shown that the CES demand system with parameter $\sigma > 1$ has elasticity at least $-\sigma$, and the Cobb-Douglas demand system has elasticity at least -1.

Separable and uniformly separable WGS utility functions. The auction algorithm in [44] was later extended in [47] to separable WGS utility functions, that is, $u = \sum_{j \in G} u_j$ where each u_j is a WGS utility function depending only on good j. This model was further generalized in [46] to *uniformly separable* WGS utility functions, that is, $\frac{\partial u(x)}{\partial x_j} = f_j(x_j)g(x)$, where each f_j is a strictly decreasing function. This class already includes CES and Cobb-Douglas utilities; however, it does not appear to extend to demand systems obtained as their conic combinations, where even the explicit form of the utility function is unclear. Further, the running time bound stated in [46] is unbounded for the CES and Cobb-Douglas cases; see the full version of the paper for further discussion.

3 Auction algorithm for exchange markets

The algorithm (shown in Algorithm 1) uses the accuracy parameter $0 < \epsilon < 0.25$, and returns a 4ϵ-approximate equilibrium. We initialize all prices $p_j = 1$ and the prices will only increase during the algorithm, in increments by a factor $(1+\epsilon)$. This initialization is enabled by Assumption 4 that guarantees the existence of market clearing prices where all positive prices are ≥ 1.[3]

We maintain a price vector p called the *market prices*; the budget of agent $i \in [n]$ is $b_i = p^\top e^{(i)}$ at the current prices. Further, every agent $i \in [n]$ maintains individual prices $p^{(i)}$ such that $p \leq p^{(i)} \leq (1+\epsilon)p$. At any point of the algorithm, agent i owns a bundle $c^{(i)}$ of the goods such that $c^{(i)} \leq x^{(i)}$ for some $x^{(i)} \in D_i(p^{(i)}, b_i)$. Some amount of good j is sold at the lower price p_j, and some at the higher price $(1+\epsilon)p_j$. The price agent i has to pay for good j is the higher price $(1+\epsilon)p_j$ if $p_j^{(i)} = (1+\epsilon)p_j$ and the lower price p_j otherwise. (Note that this is in contrast with [44] and the other previous auction algorithms where i may pay p_j for some amount of good j and $(1+\epsilon)p_j$ for another amount.)

We consider the agents one-by-one. If an agent i has surplus money, they use the subroutine `FindNewPrices` to update their prices $p^{(i)}$ and bundle $x^{(i)}$, by maintaining $x_j^{(i)} \geq c_j^{(i)}$ – this latter requirement turns out to be the main challenge. They will then try to purchase $x_j^{(i)} - c_j^{(i)}$ amount of good j in the `Outbid` procedure. They start by purchasing any unsold amount of good at price p_j. If they still need more, then they will outbid other agents who have been paying the lower price p_j for this good, by offering the higher price $(1+\epsilon)p_j$. Once good j is sold only at the higher price $(1+\epsilon)p_j$, we increase the price of the good. If no price is increased, we move to the next agent. Otherwise, we announce the new prices p and repeat. The algorithm terminates once the total surplus of the agents drops below $3\epsilon p^\top e$. At this point, we can conclude that the current prices and allocations form a 4ϵ-approximate equilibrium.

[3] Even though there might be goods priced at 0 in an equilibrium, we can always find an ϵ-approximate equilibrium where all prices are positive.

We express the running time of the algorithm in terms of the running time T_F of the subroutine `FindNewPrices`, as well as the upper bound on the ratio $p_{\max}/p_{\min}$ of the largest and smallest nonzero prices at any ϵ-equilibrium. Such an upper bound may be obtained for the specific demand systems. Alternatively, one can follow the approach of the papers [23, 25] by adding a dummy agent with a Cobb-Douglas demand system and an initial endowment of a small fraction of all goods. In the presence of such an agent, we can obtain a strong bound on $p_{\max}/p_{\min}$, at the expense of obtaining a slightly worse approximation guarantee (see the full version of the paper).

Note that for (approximate-)equilibrium prices p, αp also gives (approximate-)equilibrium prices with the same allocation, for any $\alpha > 0$. In our algorithm, the minimum price will remain at most $1+\epsilon$ throughout, see Lemma 10.

▶ **Theorem 8.** *Let T_F be an upper bound on the running time of the subroutine `FindNewPrices`. Algorithm 1 finds a 4ϵ-approximate market equilibrium in time $O\left(\frac{nmT_F}{\epsilon^2}\cdot\log\left(\frac{p_{\max}}{p_{\min}}\right)\right)$.*

There are various options for implementing `FindNewPrices`. A simple price can be implemented increment procedure for the case of bounded elasticities; recall the elasticity bound f from Definition 7. Using this subroutine and Lemma 13, we obtain the following overall bound.

▶ **Theorem 9.** *If all agents have elasticity at least $-f$ for some $f>0$, then an ϵ-approximate equilibrium can be computed in time $O\left(\frac{nm^2f\cdot T_D}{\epsilon^2}\cdot\log\left(\frac{p_{\max}}{p_{\min}}\right)\right)$, where T_D is the time needed for one call to the demand oracle.*

As noted earlier, there are demand systems (such as linear) where the flexibility parameter cannot be bounded. However, in case the demand system is given in the form (1) via a utility function that is homogeneous of degree one, we can obtain an implementation of `FindNewPrices` by solving a convex program. This is in particular applicable for Cobb-Douglas and CES utilities with $\sigma > 1$. One could find further possible ways for implementing `FindNewPrices` for particular demand systems; e.g., we give a simple direct procedure for linear utilities, and for budget-SPLC utilities. For details, see the full version of the paper.

The full version also contains an overview of the running times of previous auction algorithms.

Invariants. Let us now summarize the invariant properties maintained throughout the algorithm. We say that a bundle y dominates the bundle x if $x \le y$.

(a) Each good is partitioned into three parts according to the price it is being sold at:
- amount w_j is the unsold part of the good,
- amount l_j is sold at the lower price p_j, and
- amount h_j is sold at the higher price $(1+\epsilon)p_j$.

Moreover, $w_j + l_j > 0$, i.e., there is always a part of the good that is unsold or owned by an agent at the lower price.

(b) The unsold amount w_j of each good is non-increasing. If $w_j > 0$ then $p_j = 1$.

(c) The budget of agent i is $b_i = p^\top e^{(i)}$. Each agent i maintains prices $p^{(i)}$ such that $p \le p^{(i)} \le (1+\epsilon)p$, and owns a bundle $c^{(i)}$ that is dominated by a bundle $x^{(i)} \in D_i(p^{(i)}, b_i)$.

(d) For the amount $c_j^{(i)}$ of good j, agent i pays
- price p_j for goods in $L_i := \{j \in [m] : p_j^{(i)} < (1+\epsilon)p_j\}$, and
- the price $(1+\epsilon)p_j$ for goods in $H_i := \{j \in [m] : p_j^{(i)} = (1+\epsilon)p_j\} = [m] \setminus L_i$.

Algorithm 1 Auction algorithm for exchange markets.

```
Input: Demand systems $D_i$, and the endowment vectors $e^{(i)}$, and $\epsilon \in (0, 0.25)$.
Output: A $4\epsilon$-approximate market equilibrium.
Initialization: $\forall i, j$ set $p_j \leftarrow 1$, $p_j^{(i)} \leftarrow 1$, $c_j^{(i)} \leftarrow 0$, $w_j = e_j = \sum_i e_j^{(i)}$, and $l_j = 0$;
NewIt  for $i \in [n]$ do                                  // recompute the budgets and surpluses
    $b_i \leftarrow p^\top e^{(i)}$; $s_i \leftarrow b_i - \sum_{j \in L_i} c_j^{(i)} p_j - \sum_{j \in H_i} c_j^{(i)} (1+\epsilon) p_j$
if $\sum_{i=1}^n s_i \le 3\epsilon p^\top e$ then return $p$, $\{p^{(i)}\}_{i \in [n]}$ and $\{c^{(i)}\}_{i \in [n]}$;
NewStp  for $i \in [n]$ with $s_i > 0$ do                  // step for agent $i$
    $(\tilde{p}, y) \leftarrow$ FindNewPrices($i, p^{(i)}, p, \epsilon, c^{(i)}, b_i$);
    for $j = 1$ to $m$ do
        if $p_j^{(i)} < (1+\epsilon)p_j$ and $\tilde{p}_j = (1+\epsilon)p_j$ then          // Case 1
            $s_i \leftarrow s_i - c_j^{(i)} \cdot \epsilon p_j$; $l_j \leftarrow l_j - c_j^{(i)}$ ;   // $i$ pays $(1+\epsilon)p_j$ instead of $p_j$
            Outbid($i$, $j$, $y_j - c_j^{(i)}$);
        else if $p_j^{(i)} = (1+\epsilon)p_j$ and $\tilde{p}_j = (1+\epsilon)p_j$ then     // Case 2
            Outbid($i$, $j$, $y_j - c_j^{(i)}$);
        // Skip the goods with $p_j^{(i)} < (1+\epsilon)p_j$ and $\tilde{p}_j < (1+\epsilon)p_j$.   Case 3
    $p^{(i)} \leftarrow \tilde{p}$; flag $\leftarrow 0$;
    for $j \in [m]$ with $w_j + l_j = 0$ do
        $p_j \leftarrow (1+\epsilon)p_j$; $l_j = e_j$;                  // price increase
        foreach $k \in [n]$ do $p_j^{(k)} \leftarrow (1+\epsilon)p_j$;
        flag $\leftarrow 1$;
    if flag $= 1$ then go to NewIt;
```

Procedure Outbid(i, j, t).

```
// $t$ is the amount of good $j$ agent $i$ wants to outbid.
if $w_j > 0$ then                                   // a part of $j$ is unsold
    $\tau = \min\{w_j, t\}$;
    $w_j \leftarrow w_j - \tau$; $c_j^{(i)} \leftarrow c_j^{(i)} + \tau$; $t \leftarrow t - \tau$;
    $s_i \leftarrow s_i - \tau \cdot (1+\epsilon)p_j$;                // here $p_j = 1$ always
while $t > 0$ and $l_j > 0$ do
    Let $k \in [n]$ be such that $c_j^{(k)} > 0$ and $p_j^{(k)} = p_j$. Set $\tau = \min\{c_j^{(k)}, t\}$;
    $c_j^{(k)} \leftarrow c_j^{(k)} - \tau$; $c_j^{(i)} \leftarrow c_j^{(i)} + \tau$;         // $i$ outbids $k$
    $s_k \leftarrow s_k + \tau \cdot p_j$; $s_i \leftarrow s_i - \tau \cdot (1+\epsilon)p_j$; $l_j \leftarrow l_j - \tau$; $t \leftarrow t - \tau$;
```

In accordance with (d), the *surplus* of agent i is $s_i := b_i - \sum_{j \in L_i} c_j^{(i)} p_j - \sum_{j \in H_i} c_j^{(i)} (1+\epsilon) p_j$.

The Outbid subroutine. An important subroutine, described in Procedure Outbid, controls how the ownership of goods may change. If agent k has paid price p_j on a certain amount of good j, then agent i may take over some of this amount by offering a higher price $(1+\epsilon)p_j$. Possibly $i = k$, in which case the agent outbids herself. We also incorporate into the procedure the case when a certain amount of a good is being purchased for the first time. Note that $p_j = 1$ at this point due to invariant (b).

Main iterations. The algorithm is partitioned into iterations. Each iteration finishes when the price of a good increases from p_j to $(1+\epsilon)p_j$. At every such event, the budgets b_i of the agents also increase. Therefore, at the start of an iteration each agent i recomputes their budget at line NewIt. An iteration is further partitioned into steps, which are single executions of the main for loop in Algorithm 1. The algorithm terminates as soon as the total surplus drops below $3\epsilon p^\top e$.

Steps. Suppose we are considering agent i. By invariant (c), the agent is buying a bundle $c^{(i)} \le x^{(i)}$ for some $x^{(i)} \in D_i(p^{(i)}, b_i)$. The subroutine $\texttt{FindNewPrices(}i, p^{(i)}, p, \epsilon, c^{(i)}, b_i\texttt{)}$ delivers new prices $\tilde{p}$ and a bundle y such that

(A) $y \ge c^{(i)}$ for $y \in D_i(\tilde{p}, b_i)$, and

(B) $p^{(i)} \le \tilde{p} \le (1+\epsilon)p$, and $\tilde{p}_j = (1+\epsilon)p_j$ whenever $y_j > (1+\epsilon)\, c_j^{(i)}$.

Condition (A) says that agent i still wants whatever they own even at the increased prices $\tilde{p}$. Condition (B) is the crucial one for the outbid. It guarantees that $\tilde{p} \ge p^{(i)}$, and whenever an agent wants to buy more of some good than they already own at least by a factor $1+\epsilon$, then they are willing to pay the higher price $(1+\epsilon)p_j$ for it. (They might already be paying the increased price to start with if $p_j^{(i)} = (1+\epsilon)p_j$. In this case $\tilde{p}_j = (1+\epsilon)p_j = p_j^{(i)}$.) The description of this subroutine is given in the full version of the paper. Observe that FindNewPrices will make progress whenever $c^{(i)}$ is far from $x^{(i)}$ for some agent i. When they are very close for each agent i, then we have already reached an approximate equilibrium.

The above properties suggest the following update rules for each good $j \in [m]$.

Case 1. $p_j^{(i)} < (1+\epsilon)p_j$ and $\tilde{p}_j = (1+\epsilon)p_j$. The good j was in L_i and needs to be moved to H_i, i.e., agent i used to pay p_j but now is willing to pay the higher price for j. Agent i first outbids themselves for the amount $c_j^{(i)}$ they already own and starts paying $p_j(1+\epsilon)$ for this amount. Additionally, agent i outbids on good j up to the amount they want and that is available from the other agents.

Case 2. $p_j^{(i)} = (1+\epsilon)p_j$ and $\tilde{p}_j = (1+\epsilon)p_j$. The good j was in H_i and stays in H_i, i.e., agent i continues to pay the higher price. The agent i still keeps the amount $c_j^{(i)}$ of good j that they already had and outbids for as much as they can from the other agents.

Case 3. $p_j^{(i)} < (1+\epsilon)p_j$ and $\tilde{p}_j < (1+\epsilon)p_j$. The good j remains in L_i, i.e., agent i continues to pay the lower price. By (B), we must have $c_j^{(i)} \le y_j \le (1+\epsilon)c_j^{(i)}$; the agent will not seek to buy more of these goods.

The cases above have covered all possibilities since $p_j^{(i)} \le \tilde{p}_j$. Note that in the first two cases the agent will own $\min(y_j, l_j + w_j)$ amount of good j, whereas they will own $c_j^{(i)}$ amount in the third case. Once all of the goods have been considered we set $p^{(i)} = \tilde{p}$, $x^{(i)} = y$, and update $c^{(i)}$ as the current allocation. If $w_j + l_j = 0$ for some j then $h_j = e_j$, i.e., the whole j is sold at the higher price $p_j(1+\epsilon)$. For each such good j we increase the market price p_j to $(1+\epsilon)p_j$, and for all agents k we set $p_j^{(k)} = p_j$ for the new increased p_j; finally, we set $l_j = e_j$ and $h_j = 0$. The step ends.

3.1 Analysis

The missing proofs are presented in the full version. Here, we analyze the running time.

▶ **Lemma 10.** *The smallest price $\min_{j \in G}\{p_j\}$ remains at most $(1+\epsilon)$ throughout the algorithm.*

Next, we give a bound on the number of iterations, using the same basic idea of organizing the steps into rounds as in [44]. A *round* consists of going over all agents exactly once in the main "for" loop and doing a step for each agent; i.e, a round comprises at most n steps.

▶ **Lemma 11.** *The number of rounds in an iteration is at most* $2/\epsilon$.

Proof. Let us fix an iteration and denote with p the market prices at the start of the iteration. Consider a step of an agent i within the iteration. If from a good j, i buys everything that is available at the cheaper price p_j, then the market price of j increases and the iteration finishes. So for the rest of the proof we assume that the market price increase does not happen; consequently, the budget of each agent is unchanged and agent i gets the amount of each good it desires.

Let φ denote the total amount of money spent at a certain point of this iteration that is spent by the agents on higher price goods. That is, $\varphi = (1+\epsilon)\sum_{i=1}^{n}\sum_{j\in H_i} c_j^{(i)} p_j$.

▷ Claim 12. Let s_i denote the surplus of agent i at the beginning of their step. Then the value of φ increases at least by $s_i - 2.25\epsilon b_i$ in the step of agent i.

Proof. Recall Cases 1-3 in the description of the step. Let T_k be the set of goods that fall into case k, that is, $T_1 \cup T_2 \cup T_3 = [m]$.

- If $j \in T_1$, then $(1+\epsilon)p_j y_j$ amount will be added to φ in the `Outbid` subroutine: In this case, the agent also outbids itself, moving the good from L_i to H_i.
- If $j \in T_2$, then $(1+\epsilon)p_j(y_j - c_j^{(i)})$ amount will be added to φ in the `Outbid` subroutine.
- If $j \in T_3$, then we do not increase φ. Nevertheless, (B) guarantees that $\tilde{p}_j(y_j - c_j^{(i)}) \le \epsilon\tilde{p}_j c_j^{(i)}$. Consequently,

$$\sum_{j\in T_3} \tilde{p}_j(y_j - c_j^{(i)}) \le \epsilon\tilde{p}^\top c^{(i)}. \tag{2}$$

Also note that $\tilde{p}_j = (1+\epsilon)p_j$ if $j \in T_1 \cup T_2$. Assumption 5 on non-satiation guarantees that $\tilde{p}^\top y = b_i$. Let $\Delta\varphi$ denote the increment in φ; this can be lower bounded as

$$\begin{aligned}\Delta\varphi &= \sum_{j\in T_1}\tilde{p}_j y_j + \sum_{j\in T_2}\tilde{p}_j(y_j - c_j^{(i)}) = \tilde{p}^\top y - \sum_{j\in T_3}\tilde{p}_j y_j - \sum_{j\in T_2}\tilde{p}_j c_j^{(i)}\\ &\ge b_i - \sum_{j\in T_3}\tilde{p}_j(y_j - c_j^{(i)}) - \tilde{p}^\top c^{(i)} \ge b_i - (1+\epsilon)\tilde{p}^\top c^{(i)} \quad ,\end{aligned}$$

using (2). The money spent by the agent at the beginning of the step is $b_i - s_i$. Good j is purchased at price at least p_j according to (d), and $\tilde{p}_j \le (1+\epsilon)p_j$. Consequently, $\tilde{p}^\top c^{(i)} \le (1+\epsilon)(b_i - s_i)$. With the above inequality and using that $\epsilon < 0.25$, we obtain $\Delta\varphi \ge b_i - (1+\epsilon)^2(b_i - s_i) \ge s_i - 2.25\epsilon b_i$. ◁

As long as $\sum_{i=1}^n s_i > 3\epsilon p^\top e$, the claim guarantees that φ increases in every round by at least $3\epsilon p^\top e - 2.25\epsilon\sum_{i=1}^n b_i > 0.5\epsilon p^\top e$. Since $\varphi \le p^\top e$, the number of rounds is at most $2/\epsilon$. ◀

Proof of Theorem 8. In their steps, agents use their surpluses to outbid for the goods. We bound the number of repeats in the "while" cycles (lines 5–8) in all calls to `Outbid` in a given iteration. When `Outbid`(i,j,t) is called, the "while" loop is repeated until $t = 0$ or good j is sold only at the higher price. Moreover, `Outbid`(i,j,t) possibility sets some $c_j^{(k)}$ to zero. The total number of such events within a single iteration is bounded by nm – each agent loses a good through the outbid at most once before the prices increases and iteration finishes.

Hence, the number of "while" calls is at most nm plus the total number of calls to `Outbid`. This is at most m in each step, and thus nm in each round. According to Lemma 11, the number of repeats "while" calls in every iteration is $2nm/\epsilon$; each repeat takes $O(1)$ time. The same bound holds for the 'if' calls in lines 1–4 in `Outbid`.

Every step calls the procedure `FindNewPrices` exactly once. Therefore, the time taken by `FindNewPrices` in an iteration is $O(nT_F/\epsilon)$. According to Lemma 10, the minimum price remains at most $1+\epsilon$ throughout. Hence, the number of iterations is bounded by $O(m\log_{1+\epsilon}(p_{\max}/p_{\min})) = O(\frac{m}{\epsilon}\log(p_{\max}/p_{\min}))$. The claimed running time bound follows, using also $T_F = \Omega(m)$ since the output needs to return an m-dimensional vector of goods.

It is left to show that the prices p and bundles $c^{(i)}$ form a 4ϵ-approximate market equilibrium. The first two properties in the definition are clear: $c^{(i)}$ is dominated by an optimal bundle with respect to the prices $p^{(i)}$, and no good is oversold. At termination, the total surplus of the agents is bounded by $3\epsilon p^\top e$. However, this surplus is computed assuming that some goods are sold at price p_j and others at price $(1+\epsilon)p_j$. Decreasing the price of the latter goods to p_j releases an additional excess of at most $\epsilon p^\top e$. Consequently, $\sum_{j=1}^m p_j(e - \sum_{i=1}^n c_j^{(i)}) \le 4\epsilon p^\top e$. ◀

3.2 Implementing FindNewPrices

We now describe the subroutine `FindNewPrices`$(i, p^{(i)}, p, \epsilon, c^{(i)}, b_i)$. Recall that the outputs are new prices $\tilde{p} \ge p^{(i)}$ and a bundle y with

(A) $y \ge c^{(i)}$ for $y \in D_i(\tilde{p}, b_i)$, and

(B) $p^{(i)} \le \tilde{p} \le (1+\epsilon)p$, and $\tilde{p}_j = (1+\epsilon)p_j$ whenever $y_j > (1+\epsilon)\, c_j^{(i)}$.

Let us assume that the demand system D_i has elasticity at least $-f$ for some $f > 0$. Our Algorithm 2 for this case is a simple price increment procedure. First, we obtain $y \in D_i(p^{(i)}, b_i)$ from the demand oracle with $y \ge c^{(i)}$. This is possible due to invariant (c), which guarantees that $c^{(i)} \le x^{(i)}$ for some $x^{(i)} \le D_i(p^{(i)}, b_i)$. Then, the demand oracle is able to return a bundle y such that $y \ge x^{(i)} \ge c^{(i)}$. Then, we iterate the following step. As long as (B) is violated for a good j, we increase its price by a factor $(1+\epsilon)^{1/f}$ until it reaches the upper bound $(1+\epsilon)p_j$.

Algorithm 2 Finding new prices.

```
Input: $i, p^{(i)}, p, \epsilon, c^{(i)}, f, b_i$.
Output: Prices $\tilde{p}$ and bundle $y$.
Initialization: $\tilde{p} \leftarrow p^{(i)}$ ;
Obtain $y \in D_i(\tilde{p}, b_i)$ from the demand oracle with $y \ge c^{(i)}$ ;
while $\exists j : \tilde{p}_j < (1+\epsilon)p_j$ and $y_j > (1+\epsilon)c_j^{(i)}$ do
    $\tilde{p}_j \leftarrow \min\{(1+\epsilon)^{1/f}\tilde{p}_j, (1+\epsilon)p_j\}$ ;
    Obtain $y' \in D_i(\tilde{p}, b_i)$ from the demand oracle such that $y'_k \ge y_k$ for $k \ne j$ ;
    $y \leftarrow y'$ ;
return $(\tilde{p}, y)$ ;
```

▶ **Lemma 13.** *Assume the demand system D_i has elasticity at least $-f$ for some $f > 0$. Algorithm 2 terminates with $\tilde{p}$ and y satisfying* (A) *and* (B) *in time $O(mf \cdot T_D)$, where T_D is the time for a call to the demand oracle.*

We will assume that $T_D = \Omega(m)$, since the demand oracle needs to output an m-dimensional vector.

Proof. The bound on the number of iterations is clear: since we have $p \le \tilde{p} \le (1+\epsilon)p$ throughout, the price of every good can increase at most f times. Condition (A) is satisfied due to the WGS property and the bound on the demand elasticity. When increasing $\tilde{p}_j$, the demand y_k for $k \neq j$ is non-decreasing as guaranteed by the demand oracle. Further, y_j may decrease only by a factor $(1+\epsilon)$, and since we had $y_j > (1+\epsilon)c_j^{(i)}$ before the price update, we still have $y_j > c_j^{(i)}$ after the price update. Condition (B) is satisfied at termination since the while loop keeps running as long as it is violated. Checking the while condition each time requires $O(m)$ time; however, this will be dominated by the time T_D according to the comment on $T_D \ge m$ above. ◀

As explained in Section 3, this is only one of the possible ways of implementing `FindNewPrices`. A convex programming approach for utilities that are homogeneous of degree 1 can be developed. For example, for CES with parameter $\sigma > 1$, the running time of Algorithm 2 depends linearly on σ, whereas the running time of the convex programming is independent on this parameter. Nevertheless, for small values of σ the simple price increment procedure may be preferable to solving a convex program.

Further, more direct approaches for implementing `FindNewPrices` may be possible for particular demand systems. For Cobb-Douglas demands with parameter vector $\alpha^{(i)}$, it is easy to devise an $O(m)$ time algorithm implementing the procedure. The algorithm relies on the fact that the optimal bundle is the bundle that allocates $\alpha_j^{(i)} b_i$ money for good j. Hence, each price can be set independently of the others. Similarly, there is $O(m)$ procedure for implementing `FindNewPrices` for linear utilities; recall from Section 2.1 that the elasticity is unbounded in this case.

4 Fisher markets and the Nash social welfare problem

Fisher markets are a well-studied special case of exchange markets, where the initial endowment of agent i is $\delta_i e$ for $\delta_i > 0$ and therefore the relative budgets of the agents are independent of the prices. With appropriate normalization of the prices, we can assume that agent i arrives with a fixed budget b_i and that there is exactly one unit of each good. At an equilibrium, the agents spend these budgets on their most preferred goods at the given prices. Let us now assume that the demand systems are given via utility functions as in (1). Eisenberg and Gale [36] gave a convex programming formulation of the market equilibrium problem for linear utilities. Eisenberg [35] showed that the optimal solutions to the following convex program are in one-to-one correspondence with the market equilibria assuming that the utility functions are homogeneous of degree one, that is, $u_i(\alpha x) = \alpha u_i(x)$ for any $\alpha > 0$.

$$\max \sum_{i=1}^{n} b_i \log u_i(x^{(i)}) \quad \text{subject to} \quad \sum_{i=1}^{n} x_j^{(i)} \le 1, \quad \forall j = 1, \dots, m. \tag{3}$$

We note that the equilibrium prices are given by the optimal Lagrange multipliers.

The Nash social welfare problem. In the Nash social welfare (NSW) problem, we need to allocate m indivisible items to n agents ($m \ge n$), with agent i equipped with a utility function on the subsets of goods. The goal is to find a partition $S_1 \cup S_2 \cup \dots S_n = [m]$ of the goods in order to maximize the geometric mean of the utilities, $\left(\prod_{i=1}^{n} u_i(S_i)\right)^{1/n}$. This problem is NP-hard already for additive utilities, that is, if $u_i(S) = \sum_{j \in S} v_{ij}$.

The first constant factor approximation for this problem was given by Cole and Gkatzelis [28]. Their approach was to first solve a continuous relaxation that corresponds to a divisible market problem, and round the fractional optimal solution. The natural relaxation is exactly the program (3) above with all $b_i = 1$. For linear utilities, we can use the natural continuous extension $u_i(x) = \sum_{j\in S} v_{ij}x_{ij}$ of the additive utility function. However, it is easy to see that this relaxation has an unbounded integrality gap. Cole and Gkatzelis [28] introduced the notion of *spending restricted equilibrium* that we now define in a slightly more general form.

▶ **Definition 14.** *Suppose there are n agents with demand systems $D_i(p,b_i)$ and fixed budgets $b\in\mathbb{R}^n_+$. Further, let us be given bounds $t\in(0,\infty)^m$. The prices $p\in\mathbb{R}^m$ and allocations $x^{(i)}\in D_i(p,b_i)$ form a* Spending Restricted (SR) *equilibrium with respect to t, if $\sum_i x^{(i)}_j = \min\{1,t_j/p_j\}, \forall j\in[m]$.*

Note that the set of equilibria can be non-convex already for budget-additive utilities as shown in [38].

At given prices p, we let $a_j(p) = a_j = \min\{1,t_j/p_j\}$ denote the *available amount* of good j. That is, the amount of money spent on good j is bounded by t_j. By setting $t_j=\infty$ for all j, the above reduces to the standard definition of Fisher market equilibrium.

The algorithm in [28] first computes a spending restricted equilibrium for linear Fisher markets with bounds $t_j = 1$, and show that this can be rounded to an integer solution of cost at most $2e^{1/e}$ times the optimal NSW solution. Note that the spending restrictions cannot be directly added to the formulation (3) since they involve the Lagrange multipliers p. An SR-equilibrium in [28] was found via an extension of algorithms by Devanur et al. [31] and Orlin [58] for linear Fisher markets.

Subsequent work by Cole et al. [26] showed that a spending restricted equilibrium for the linear markets can be obtained as an optimal solution of a convex program (extending a convex formulation of linear Fisher market equilibrium that is different from (3)), and also improved the approximation guarantee to 2 (the current best factor is 1.45 [8]). However, this convex formulation is only known to work for linear utility functions.

Further work has studied the NSW problem for more general utility functions, following the same strategy of first solving a spending-restricted market equilibrium problem then rounding. Anari et al. [2] studied NSW with *separable, piecewise-linear concave (SPLC)* utilities. The paper [38] studied *budget-additive valuations*, that correspond to the utility function $u_i(x) = \min(c_i, \sum_j u_{ij}x_j)$. Both papers find (exact or approximate) solutions to the corresponding spending-restricted market equilibrium problem via fairly complex combinatorial algorithms.

The Gale demand systems. The demand systems of the market models in [2, 38] do not exactly correspond to (1). In [38] one needs additional conditions on the agents being "thrifty"; in [2] a "utility market model" is used. In both cases, the total spending of the agents can be below their budgets. A natural unified way of capturing these equilibrium concepts is via *Gale demand systems*, defined as

$$G^u(p,b) = \operatorname*{arg\,max}_{x\in\mathbb{R}^m_+} b\log u(x) - p^\top x\,. \tag{4}$$

We call $b\log u(x) - p^\top x$ the *Gale objective function.* It is easy to verify using Lagrangian duality that if all u_i's are concave functions, and the utility functions correspond to the Gale demand systems $D_i(p,b) = G^{u_i}(p,b)$, then the program (3) always finds a market equilibrium; see [56] for details. Moreover, if the utilities are homogeneous of degree one, then this equilibrium coincides with the equilibrium for the "standard" demand systems given by (1). For general concave utility functions, the optimal bundles stay within the budget b (that is, $p^\top x \le b$), but may not exhaust it. Finding a spending-restricted equilibrium for

Gale demand systems appears to be the right setting for NSW; in fact, the concepts used by [2] and [38] correspond to the Gale equilibrium in these settings, and moreover, these Gale demand systems admit the WGS property. On contrary, the demand systems arising from the previously mentioned utility functions do not satisfy the WGS property in the usual setting (1).

We refer the reader to the paper by Nesterov and Shikhman [56] on Gale demand systems as well as the more general concept of Fisher-Gale equilibrium; they also give a tâtonnement type algorithm for finding such an equilibrium.

Approximate spending-restricted equilibrium. We use an extension of Definition 6 as our approximate SR-equilibrium notion. The main difference is that we require all goods to be fully consumed.

▶ **Definition 15** (Approximate SR-equilibrium). *Let $t \in [1,\infty]^m$. For an $\epsilon > 0$, the prices $p \in \mathbb{R}^m$ and bundles $x^{(i)} \in \mathbb{R}^m_+$ form an ϵ-*approximate SR-equilibrium *w.r.t. t if*

(i) $x^{(i)} \le z^{(i)}$ *for some* $z^{(i)} \in D_i(p^{(i)}, b_i)$*, where* $p \le p^{(i)} \le (1+\epsilon)p$*,*

(ii) $\sum_{i=1}^n x_j^{(i)} = a_j := \min\{1, t_j/p_j\}$ *for all j, and*

(iii) $\sum_{j=1}^m p_j \left(\sum_{i=1}^n z_j^{(i)} - a_j\right) \le \epsilon \sum_{i=1}^n b_i$.

We note that whereas an equilibrium will always exist for WGS utilities, the existence of an SR-equilibrium is a nontrivial question. For example, suppose an agent i has budget b_i and Cobb-Douglas utility function $\prod_{j=1}^m (x_j^{(i)})^{\beta_j}$, where $\sum_j \beta_j = 1$, such that $\beta_k > \frac{1}{b_i}$ for some k with $t_k = 1$. Then the agent i would like to spend at least $\beta_k b_i > 1$ on good j for any prices p, but the total money that can be spent on this good is capped at 1. Hence, there doesn't exist any SR-equilibrium in this case.

While we do not have general necessary and sufficient conditions on the existence of an SR-equilibrium, we show that the objectives previously studied in the context of NSW admit an SR-equilibrium. In the case of budget-additive utilities, we have all $t_j = 1$, and all $b_i = 1$. An ϵ/n-approximate SR-equilibrium satisfies the required accuracy in [38]. Whereas [2] computes an exact SR-equilibrium, an approximate SR-equilibrium is sufficient to obtain a (slightly worse) approximation guarantee.

We show that our algorithmic framework is applicable to compute an ϵ-equilibrium for *budget-SPLC*, the common generalization of the models in [2] and [38]. Using a similar rounding as in [38], we obtain a constant-factor approximation algorithm for maximizing NSW in polynomial-time when agents have budget-SPLC utilities and goods come in multiple copies. The previous algorithm for this setting in [17] runs in pseudopolynomial time. For the special case of additive utilities, [10] gives such an algorithm.

References

1 Nima Anari, Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. Nash social welfare, matrix permanent, and stable polynomials. In *Proceedings of the 8th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 67, page 36. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.

2 Nima Anari, Tung Mai, Shayan Oveis Gharan, and Vijay V Vazirani. Nash social welfare for indivisible items under separable, piecewise-linear concave utilities. In *Proceedings of the 29th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2274–2290. SIAM, 2018.

3 Kenneth J Arrow, Henry D Block, and Leonid Hurwicz. On the stability of the competitive equilibrium, II. *Econometrica: Journal of the Econometric Society*, pages 82–109, 1959.

4 Kenneth J Arrow and Gerard Debreu. Existence of an equilibrium for a competitive economy. *Econometrica: Journal of the Econometric Society*, pages 265–290, 1954.

5 Kenneth J Arrow and Leonid Hurwicz. On the stability of the competitive equilibrium, I. *Econometrica: Journal of the Econometric Society*, pages 522–552, 1958.

6 Kenneth J Arrow and Leonid Hurwicz. Competitive stability under weak gross substitutability: The “Euclidean distance” approach. *International Economic Review*, 1(1):38–49, 1960.

7 Noa Avigdor-Elgrabli, Yuval Rabani, and Gala Yadgar. Convergence of tâtonnement in Fisher markets. *arXiv preprint*, 2014. `arXiv:1401.6637`.

8 Siddharth Barman, Sanath Kumar Krishnamurthy, and Rohit Vaish. Finding fair and efficient allocations. In *Proceedings of the 2018 ACM Conference on Economics and Computation (EC)*, pages 557–574. ACM, 2018.

9 Xiaohui Bei, Jugal Garg, and Martin Hoefer. Ascending-price algorithms for unknown markets. *ACM Transactions on Algorithms (TALG)*, 15(3):37:1–37:33, 2019.

10 Xiaohui Bei, Jugal Garg, Martin Hoefer, and Kurt Mehlhorn. Earning and utility limits in Fisher markets. *ACM Trans. Economics and Comput.*, 7(2):10:1–10:35, 2019.

11 Dimitri P Bertsekas. A new algorithm for the assignment problem. *Mathematical Programming*, 21(1):152–171, 1981.

12 Dimitri P Bertsekas. The auction algorithm for assignment and other network flow problems: A tutorial. *Interfaces*, 20(4):133–149, 1990.

13 Benjamin Birnbaum, Nikhil Devanur, and Lin Xiao. Distributed algorithms via gradient descent for Fisher markets. In *Proceedings of the 12th Conf. Electronic Commerce (EC)*, pages 127–136, 2011.

14 William C Brainard and Herbert E Scarf. How to compute equilibrium prices in 1891. *American Journal of Economics and Sociology*, 64(1):57–83, 2005.

15 Simina Brânzei, Nikhil R. Devanur, and Yuval Rabani. Proportional dynamics in exchange economies. *CoRR*, abs/1907.05037, 2019. `arXiv:1907.05037`.

16 Simina Brânzei, Ruta Mehta, and Noam Nisan. Universal growth in production economies. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, page 1975, 2018.

17 Bhaskar Ray Chaudhury, Yun Kuen Cheung, Jugal Garg, Naveen Garg, Martin Hoefer, and Kurt Mehlhorn. On fair division for indivisible items. In *Proceedings of the 38th IARCS annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 25:1–25:17. Springer, 2018.

18 Xi Chen, Decheng Dai, Ye Du, and Shang-Hua Teng. Settling the complexity of Arrow-Debreu equilibria in markets with additively separable utilities. In *Proceedings of the 50th Symposium Foundations of Computer Science (FOCS)*, pages 273–282. IEEE, 2009.

19 Yun Kuen Cheung, Richard Cole, and Nikhil R Devanur. Tâtonnement beyond gross substitutes? Gradient descent to the rescue. *Games and Economic Behavior*, 2019.

20 Yun Kuen Cheung, Richard Cole, and Ashish Rastogi. Tatonnement in ongoing markets of complementary goods. In *Proceedings of the 2012 ACM Conference on Electronic Commerce (EC)*, 2012.

21 Yun Kuen Cheung, Richard Cole, and Yixin Tao. Dynamics of distributed updating in Fisher markets. In *Proceedings of the 2018 ACM Conference on Economics and Computation, Ithaca, NY, USA, June 18-22, 2018*, pages 351–368, 2018.

22 Yun Kuen Cheung, Martin Hoefer, and Paresh Nakhe. Tracing equilibrium in dynamic markets via distributed adaptation. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS ’19, Montreal, QC, Canada, May 13-17, 2019*, pages 1225–1233, 2019.

23 Bruno Codenotti, Benton McCune, and Kasturi Varadarajan. Market equilibrium via the excess demand function. In *Proceedings of the 37th ACM symposium on Theory of Computing (STOC)*, pages 74–83. ACM, 2005.

24 Bruno Codenotti, Sriram Pemmaraju, and Kasturi Varadarajan. The computation of market equilibria. *Acm Sigact News*, 35(4):23–37, 2004.

25 Bruno Codenotti, Sriram Pemmaraju, and Kasturi Varadarajan. On the polynomial time computation of equilibria for certain exchange economies. In *Proceedings of the 16th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 72–81. SIAM, 2005.

26 Richard Cole, Nikhil Devanur, Vasilis Gkatzelis, Kamal Jain, Tung Mai, Vijay V Vazirani, and Sadra Yazdanbod. Convex program duality, Fisher markets, and Nash social welfare. In *Proceedings of the 2017 ACM Conference on Economics and Computation (EC)*, pages 459–460. ACM, 2017.

27 Richard Cole and Lisa Fleischer. Fast-converging tatonnement algorithms for one-time and ongoing market problems. In *Proceedings of the 40th ACM symposium on Theory of Computing (STOC)*, pages 315–324. ACM, 2008.

28 Richard Cole and Vasilis Gkatzelis. Approximating the Nash social welfare with indivisible items. *SIAM J. Comput.*, 47(3):1211–1236, 2018.

29 Vincent P Crawford and Elsie Marie Knoer. Job matching with heterogeneous firms and workers. *Econometrica: Journal of the Econometric Society*, pages 437–450, 1981.

30 Gabrielle Demange, David Gale, and Marilda Sotomayor. Multi-item auctions. *Journal of Political Economy*, 94(4):863–872, 1986.

31 Nikhil Devanur, Christos Papadimitriou, Amin Saberi, and Vijay Vazirani. Market equilibrium via a primal–dual algorithm for a convex program. *Jounal of the ACM*, 55(5), 2008.

32 Nikhil R Devanur and Vijay V Vazirani. An improved approximation scheme for computing Arrow–Debreu prices for the linear case. In *Proceedings of the 23rd IARCS annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 149–155. Springer, 2003.

33 Nikhil R Devanur and Vijay V Vazirani. The spending constraint model for market equilibrium: Algorithmic, existence and uniqueness results. In *Proceedings of the 36th ACM Symposium on Theory of Computing (STOC)*, volume 36, pages 519–528. ACM, 2004.

34 Ran Duan and Kurt Mehlhorn. A combinatorial polynomial algorithm for the linear Arrow-Debreu market. *Information and Computation*, 243:112–132, 2015.

35 Edmund Eisenberg. Aggregation of utility functions. *Management Science*, 7(4):337–350, 1961.

36 Edmund Eisenberg and David Gale. Consensus of subjective probabilities: The pari-mutuel method. *The Annals of Mathematical Statistics*, 30(1):165–168, 1959.

37 Lisa Fleischer, Rahul Garg, Sanjiv Kapoor, Rohit Khandekar, and Amin Saberi. A fast and simple algorithm for computing market equilibria. In *Proceedings of the 4th International Workshop on Internet and Network Economics (WINE)*, pages 19–30. Springer, 2008.

38 Jugal Garg, Martin Hoefer, and Kurt Mehlhorn. Approximating the Nash social welfare with budget-additive valuations. In *Proceedings of the 29th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2326–2340. SIAM, 2018.

39 Jugal Garg, Edin Husic, and László A. Végh. Auction algorithms for market equilibrium with weak gross substitute demands. *CoRR*, abs/1908.07948, 2019. `arXiv:1908.07948`.

40 Jugal Garg, Edin Husić, and László A. Végh. Approximating Nash social welfare under Rado valuations, 2020. `arXiv:2009.14793`.

41 Jugal Garg and Peter McGlaughlin. Improving Nash social welfare approximations. In *Proceedings of the 28th International Joint Conferences on Artificial Intelligence (IJCAI)*, 2019.

42 Jugal Garg, Ruta Mehta, Vijay V Vazirani, and Sadra Yazdanbod. Settling the complexity of Leontief and PLC exchange markets under exact and approximate equilibria. In *Proceedings of the 49th ACM Symposium on Theory of Computing (STOC)*, pages 890–901. ACM, 2017.

43 Jugal Garg and László A Végh. A strongly polynomial algorithm for linear exchange markets. In *Proceedings of the 51st Symp. Theory of Computing (STOC)*, 2019.

44 Rahul Garg and Sanjiv Kapoor. Auction algorithms for market equilibrium. *Mathematics of Operations Research*, 31(4):714–729, 2006.

45 Rahul Garg and Sanjiv Kapoor. Price roll-backs and path auctions: An approximation scheme for computing the market equilibrium. In *Proceedings of the 2nd International Workshop on Internet and Network Economics (WINE)*, pages 225–238. Springer, 2006.

46 Rahul Garg and Sanjiv Kapoor. Market equilibrium using auctions for a class of gross-substitute utilities. In *Proceedings of the 3rd International Workshop on Web and Internet Economics (WINE)*, pages 356–361. Springer, 2007.

47 Rahul Garg, Sanjiv Kapoor, and Vijay Vazirani. An auction-based market equilibrium algorithm for the separable gross substitutability case. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 128–138. Springer, 2004.

48 Mehdi Ghiyasvand and James B Orlin. A simple approximation algorithm for computing Arrow–Debreu prices. *Operations Research*, 60(5):1245–1248, 2012.

49 K. Jain and K. Varadarajan. Equilibria for economies with production: Constant-returns technologies and production planning constraints. In *Proceedings of the 17th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 688–697. SIAM, 2006.

50 Sanjiv Kapoor, Aranyak Mehta, and Vijay Vazirani. An auction-based market equilibrium algorithm for a production model. *Theoretical Computer Science*, 378(2):153–164, 2007.

51 Wouter J. Keller. A nested CES-type utility function and its demand and price-index functions. *European Economic Review*, 7:175–186, 1976.

52 Alexander S Kelso Jr and Vincent P Crawford. Job matching, coalition formation, and gross substitutes. *Econometrica: Journal of the Econometric Society*, pages 1483–1504, 1982.

53 Renato Paes Leme. Gross substitutability: An algorithmic survey. *Games and Economic Behavior*, 106:294–316, 2017.

54 Andreu Mas-Colell, Michael Dennis Whinston, Jerry R Green, et al. *Microeconomic theory*, volume 1. Oxford university press New York, 1995.

55 Kiminori Matsuyama and Philip Ushchev. Beyond CES: Three alternative cases of flexible homothetic demand systems. Buffett Institute Global Poverty Research Lab Working Paper No. 17-109, 2017.

56 Yurii Nesterov and Vladimir Shikhman. Computation of Fisher–Gale equilibrium by auction. *Journal of the Operations Research Society of China*, 6(3):349–389, 2018.

57 Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani. *Algorithmic game theory*. Cambridge University Press, 2007.

58 James B Orlin. Improved algorithms for computing Fisher's market clearing prices: Computing Fisher's market clearing prices. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC)*, pages 291–300. ACM, 2010.

59 Herbert Scarf. Some examples of global instability of the competitive equilibrium. *International Economic Review*, 1(3):157–172, 1960.

60 Vijay Vazirani and Mihalis Yannakakis. Market equilibrium under separable, piecewise-linear, concave utilities. *Jounal of the ACM*, 58(3):10, 2011.

61 Léon Walras. *Éléments d'économie politique pure, ou, Théorie de la richesse sociale*. F. Rouge, 1896.

62 Fang Wu and Li Zhang. Proportional response dynamics leads to market equilibrium. In *Proceedings of the 39th Symp. Theory of Computing (STOC)*, pages 354–363, 2007.

63 Yinyu Ye. A path to the Arrow-Debreu competitive market equilibrium. *Mathematical Programming*, 111(1-2):315–348, 2008.

64 Li Zhang. Proportional response dynamics in the Fisher market. *Theoretical Computer Science*, 412(24):2691–2698, 2011.

Efficiently Testing Simon's Congruence

Paweł Gawrychowski
Faculty of Mathematics and Computer Science, University of Wrocław, Poland

Maria Kosche
Computer Science Department, Universität Göttingen, Germany

Tore Koß
Computer Science Department, Universität Göttingen, Germany

Florin Manea
Computer Science Department, Universität Göttingen, Germany
Campus-Institut Data Science, Göttingen, Germany

Stefan Siemer
Computer Science Department, Universität Göttingen, Germany

Abstract

Simon's congruence $\sim_k$ is a relation on words defined by Imre Simon in the 1970s and intensely studied since then. This congruence was initially used in connection to piecewise testable languages, but also found many applications in, e.g., learning theory, databases theory, or linguistics. The $\sim_k$-relation is defined as follows: two words are $\sim_k$-congruent if they have the same set of subsequences of length at most k. A long standing open problem, stated already by Simon in his initial works on this topic, was to design an algorithm which computes, given two words s and t, the largest k for which $s \sim_k t$. We propose the first algorithm solving this problem in linear time $O(|s| + |t|)$ when the input words are over the integer alphabet $\{1, \ldots, |s| + |t|\}$ (or other alphabets which can be sorted in linear time). Our approach can be extended to an optimal algorithm in the case of general alphabets as well.

To achieve these results, we introduce a novel data-structure, called Simon-Tree, which allows us to construct a natural representation of the equivalence classes induced by $\sim_k$ on the set of suffixes of a word, for all $k \geq 1$. We show that such a tree can be constructed for an input word in linear time. Then, when working with two words s and t, we compute their respective Simon-Trees and efficiently build a correspondence between the nodes of these trees. This correspondence, which can also be constructed in linear time $O(|s| + |t|)$, allows us to retrieve the largest k for which $s \sim_k t$.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Design and analysis of algorithms

Keywords and phrases Simon's congruence, Subsequence, Scattered factor, Efficient algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.34

Related Version *Full Version*: `https://arxiv.org/abs/2005.01112`

Funding The work of the four authors from Göttingen was supported by the DFG-grant 389613931 *Kombinatorik der Wortmorphismen*.

1 Introduction

A subsequence of a word w (also called scattered factor or subword, especially in automata and language theory) is a word u such that there exist (possibly empty) words $v_1, \ldots, v_{\ell+1}$, $u_1, \ldots, u_\ell$ with $u = u_1 \ldots u_n$ and $w = v_1 u_1 v_2 u_2 \ldots v_\ell u_\ell v_{\ell+1}$. Intuitively, the subsequences of a word w are exactly those words obtained by deleting some of the letters of w, so, in a sense, they can be seen as lossy-representations of the word w. Accordingly, subsequences may be a natural mathematical model for situations where one has to deal with input strings with missing or erroneous symbols sequencing, such as processing DNA data or

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 34; pp. 34:1–34:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

digital signals [27]. Due to this very simple and intuitive definition as well as the apparently large potential for applications, there is a high interest in understanding the fundamental properties that can be derived in connection to the sets of subsequences of words. This is reflected in the consistent literature developed around this topic. J. Sakarovitch and I. Simon in [21, Chapter 6] overview some of the most important combinatorial and language theoretic properties of sets of subsequences. The theory of subsequences was further developed in various directions, such as combinatorics on words, automata theory, formal verification, or string algorithms. For instance, subword histories and Parikh matrices (see, e.g., [23, 26, 28]) are algebraic structures in which the number of specific subsequences occurring in a word are stored and used to define combinatorial properties of words. Strongly related, the binomial complexity of words is a measure of the multiset of subsequences that occur in a word, where each occurrence of such a factor is considered as an element of the respective multiset; combinatorial and algorithmic results related to this topic are obtained in, e.g., [25, 9, 20, 19], and the references therein. Further, in [35, 12, 18] various logic-theories were developed, starting from the subsequence-relation, and analysed mostly with automata theory and formal verification tools. Last, but not least, many classical problems in the area of string algorithms are related to subsequences: the longest common subsequence and the shortest common supersequence problems [22, 1, 3, 4], or the string-to-string correction problem [34]. Several algorithmic problems connected to subsequence-combinatorics are approached and (partially) solved in [7].

One particularly interesting notion related to subsequences was introduced by Simon in [30]. He defined the relation $\sim_k$ (called now Simon's congruence) as follows. Two words are $\sim_k$-congruent if they have the same set of subsequences of length at most k. In [30], as well as in [21, Chapter 6], many fundamental properties (mainly of combinatorial nature) of $\sim_k$ are discussed; this line of research was continued in, e.g., [15, 16, 17, 6, 2] where the focus was on the properties of some special classes of this equivalence. From an algorithmic point of view, a natural decision problem and its optimisation variant stand out:

▶ **Problem 1.** SimK*: Given two words s and t over an alphabet Σ, with $|s| = n$ and $|t| = n'$, with $n \geq n'$, and a natural number k, decide whether $s \sim_k t$.*

▶ **Problem 2.** MaxSimK*: Given two words s and t over an alphabet Σ, with $|s| = n$ and $|t| = n'$, with $n \geq n'$, find the maximum k for which $s \sim_k t$.*

The problems above were usually considered assuming (although not always explicitly) the Word RAM model with words of logarithmic size. This is a standard computational model in algorithm design in which, for an input of size n, the memory consists of memory-words consisting of $\Theta(\log n)$ bits. Basic operations (including arithmetic and bitwise Boolean operations) on memory-words take constant time, and any memory-word can be accessed in constant time. In this model, the two input words are just sequences of integers, each integer stored in a single memory-word. Without losing generality, we can assume the alphabet to be simply $\{1, \ldots, n + n'\}$ by sorting and renaming the letters occurring in the input in linear time. For a detailed discussion on the computational model, see the full version of this paper, available on arXiv [24].

Due to the central role played by the $\sim_k$-congruence in the study of piecewise testable languages, as well as in the many other areas mentioned above, both problems SimK and MaxSimK were considered highly interesting and studied thoroughly in the literature.

In particular, Hebrard [13] presents MaxSimK as computing a similarity measure between strings and mentions a solution of Simon [29] for MaxSimK which runs in $O(|\Sigma|nn')$ (the same solution is mentioned in [11]). He goes on and improves this (see [13]) in the case when

Σ is a binary alphabet: given two bitstrings s and t, one can find the maximum k for which $s \sim_k t$ in linear time. The problem of finding optimal algorithms for MaxSimK, or even SimK, for general alphabets was left open in [29, 13] as the methods used in the latter paper for binary strings did not seem to scale up. In [11], Garel considers MaxSimK and presents an algorithm based on finite automata, running in $O(|\Sigma|n)$, which computes all *distinguishing words* u of minimum length, i.e., words which are factors of only one of the words s and t from the problem's statement. Several further improvements on the aforementioned results were reported in [5, 32]. Finally, in an extended abstract from 2003 [31], Simon presented another algorithm based on finite automata solving MaxSimK which runs in $O(|\Sigma|n)$, and he conjectures that it can be implemented in $O(|\Sigma| + n)$. Unfortunately, the last claim is only vaguely and insufficiently substantiated, and obtaining an algorithm with the claimed complexity seems to be non-trivial. Although Simon announced that a detailed description of this algorithm will follow shortly, we were not able to find it in the literature.

In [8], a new approach to efficiently solving SimK was introduced. This idea was to compute, for the two given words s and t and the given number k, their shortlex forms: the words which have the same set of subsequences of length at most k as s and t, respectively, and are also lexicographically smallest among all words with the respective property. Clearly, $s \sim_k t$ if and only if the shortlex forms of s and t for k coincide. The shortlex form of a word s of length n over Σ was computed in $O(|\Sigma|n)$ time, so SimK was also solved in $O(|\Sigma|n)$. A more efficient implementation of the ideas introduced in [8] was presented in [2]: the shortlex form of a word of length n over Σ can be computed in linear time $O(n)$, so SimK can be solved in optimal linear time. By binary searching for the smallest k, this gives an $O(n \log n)$ time solution for MaxSimK. This brings up the challenge of designing an optimal linear-time algorithm for non-binary alphabets.

Our results. In this paper we confirm Simon's claim from 2003 [31]. We present a complete algorithm solving MaxSimK in linear time on Word RAM with words of size $\Theta(\log n)$. This closes the problem of finding an optimal algorithm for MaxSimK. Our approach is not based on finite automata (as the one suggested by Simon), nor on the ideas from [8, 2]. Instead, it works as follows. Firstly, looking at a single word, we partition the respective word into k-blocks: contiguous intervals of positions inside the word, such that all suffixes of the word inside the same block have exactly the same subsequences of length at most k (i.e., they are $\sim_k$-equivalent). Since the partition in $(k+1)$-blocks refines the partition in k-blocks, one can introduce the *Simon-Tree* associated to a word: its nodes are the k-blocks (for k from 1 to at most n), and each node on level k has as children exactly the $(k+1)$-blocks in which it is partitioned. We first show how to compute efficiently the Simon-Tree of a word. Then, to solve MaxSimK, we show that one can maintain in linear time a connection between the nodes on the same levels of the Simon-Trees associated to the two input words. More precisely, for all ℓ, we connect two nodes on level ℓ of the two trees if the suffixes starting in those blocks, in their respective words, have exactly the same subsequences of length at most ℓ. It follows that the value k required in MaxSimK is the lowest level of the trees on which the blocks containing the first position of the respective input words are connected. Using the Simon-Trees of the two words and the connection between their nodes, we can also compute in linear time a distinguishing word of minimal length for s and t. Achieving the desired complexities is based on a series of combinatorial properties of the Simon-Trees, as well as on a rather involved data structures toolbox.

Our paper is structured as follows: we firstly introduce the basic combinatorial and data structures notions, then we show how Simon-Trees are constructed efficiently, and, finally, we show how MaxSimK is solved by connecting the Simon-Trees of the two input words. We end this paper with a series of concluding remarks, extensions, and further research questions. The formal proofs of our results are given in the full version of this paper [24], due to space constraints; most statements are, however, accompanied by explanations substantiating our claims. Similarly, for space reasons, a discussion on how our results can be extended to an optimal algorithm for MaxSimK in the case of input words over general alphabets is also given in the paper's full version [24].

2 Preliminaries

Let $\mathbb{N}$ be the set of natural numbers, including 0. An alphabet Σ is a nonempty finite set of symbols called *letters*. A *word* is a finite sequence of letters from Σ, thus an element of the free monoid Σ^*. Let $\Sigma^+ = \Sigma^* \backslash \{\varepsilon\}$, where ε is the empty word. The *length* of a word $w \in \Sigma^*$ is denoted by $|w|$. The i^{th} letter of $w \in \Sigma^*$ is denoted by $w[i]$, for $i \in [1 : |w|]$. For $m, n \in \mathbb{N}$, we let $[m : n] = \{m, m+1, \ldots, n\}$ and $w[m : n] = w[m]w[m+1]\ldots w[n]$.

A word $u \in \Sigma^*$ is a *factor* of $w \in \Sigma^*$ if $w = xuy$ for some $x, y \in \Sigma^*$. If $x = \varepsilon$ (resp. $y = \varepsilon$), u is called a *prefix* (resp. *suffix* of w). For some $x \in \Sigma$ and $w \in \Sigma^*$, let $|w|_x = |\{i \in [1 : |w|] \mid w[i] = x\}|$ and $\mathrm{alph}(w) = \{x \in \Sigma \mid |w|_x > 0\}$ for $w \in \Sigma^*$; in other words, $\mathrm{alph}(w)$ denotes the smallest subset $S \subset \Sigma$ such that $w \in S^*$.

▶ **Definition 1.** *We call u a subsequence of length k of w, where $|w| = n$, if there exist positions $1 \leq i_1 < i_2 < \ldots < i_k \leq n$, such that $u = w[i_1]w[i_2]\cdots w[i_k]$. Let $\mathrm{Subseq}_{\leq k}(i, w)$ denote the set of subsequences of length at most k of $w[i : n]$. Accordingly, the set of subsequences of length at most k of the entire word w will be denoted by $\mathrm{Subseq}_{\leq k}(1, w)$.*

Equivalently, $u = u_1 \ldots u_\ell$ is a subsequence of w if there exist $v_1, \ldots, v_{\ell+1} \in \Sigma^*$ such that $w = v_1 u_1 \ldots v_\ell u_\ell v_{\ell+1}$. For $k \in \mathbb{N}$, $\mathrm{Subseq}_{\leq k}(1, w)$ is called the full k-spectrum of w.

▶ **Definition 2** (Simon's Congruence).

(i) *Let $w, w' \in \Sigma^*$. We say that w and w' are equivalent under Simon's congruence $\sim_k$ (or, alternatively, that w and w' are k-equivalent) if the set of subsequences of length at most k of w equals the set of subsequences of length at most k of w', i.e., $\mathrm{Subseq}_{\leq k}(1, w) = \mathrm{Subseq}_{\leq k}(1, w')$.*

(ii) *Let $i, j \in [1 : |w|]$. We define $i \sim_k j$ (w.r.t. w) if $w[i : n] \sim_k w[j : n]$, and we say that the positions i and j are k-equivalent.*

(iii) *A word u of length k distinguishes w and w' w.r.t. $\sim_k$ if u occurs in exactly one of the sets $\mathrm{Subseq}_{\leq k}(1, w)$ and $\mathrm{Subseq}_{\leq k}(1, w')$.*

Following the discussion from the introduction, for our algorithmic results we assume the Word RAM model with words of size $\Theta(\log n)$.

We start by recalling two data structures which play an important role in our results. These are the interval split-find and interval union-find data structures. Their formal definition is given in the full version of this paper [24]. Rather informally, in the union-find (respectively, split-find) structure we maintain a partition of an interval (also called universe) $V = [1 : n]$ in sub-intervals, under two operations: `union` of adjacent intervals (respectively, `split` an interval in two sub-intervals around an element of the interval), and `find` the representative of the interval containing a given value. In our algorithms, when using these structures, we usually describe the intervals stored initially in the structure, and then the `unions` (respectively, the `splits`) which are made, as well as the `find` operations, without going into the formalism behind these operations. In usual implementations of these structures the representative of each interval, which is returned by `find`, is its maximum; we can easily enhance the data structures so that the `find` operation returns both borders of the interval containing the searched value. The following lemma was shown in [10, 14].

▶ **Lemma 3.** *One can implement the interval split-find (respectively, union-find) data structures, such that, the initialisation of the structures followed by a sequence of $m \in O(n)$* `split` *(respectively,* `union`*) and* `find` *operations can be executed in $\mathcal{O}(n)$ time and space.*

3 Constructing the Simon-Tree of a word

In this section, we introduce a new data structure, which is fundamental to our approach – the *Simon-Tree*. The Simon-Tree is used as a representation for the equivalence classes in a word, which are explained in Section 3.1. The definition of Simon-Trees is then given in Section 3.2, and the construction is described in Section 3.3.

3.1 Equivalence classes of a Word

In this section, we develop a method to efficiently partition the positions of a given word w, of length n, into equivalence classes w.r.t. $\sim_k$, such that all suffixes starting with positions of the same class have the same set of subsequences of length at most k. As in this section we only deal with one input word w of length n, we will sometimes omit the reference to this word in our notation: e.g. $\mathrm{Subseq}_k(i) = \mathrm{Subseq}_k(i, w)$; in the case of such omissions, the reader may safely assume that we are referring to the aforementioned input word.

Firstly, we will examine the equivalence classes that each congruence relation $\sim_k$ induces on the set of suffixes of w for all k. Let $1 \leq i < j \leq n$, then $w[j : n]$ is a suffix of $w[i : n]$, hence $\mathrm{Subseq}_k(i) \supseteq \mathrm{Subseq}_k(j)$ holds for all $k \in \mathbb{N}$. For any $l \in [i : j]$ we obtain $\mathrm{Subseq}_k(i) \supseteq \mathrm{Subseq}_k(l) \supseteq \mathrm{Subseq}_k(j)$. If we additionally let $i \sim_k j$, then the sets of subsequences corresponding to i and j respectively are equal, so $\mathrm{Subseq}_k(i) = \mathrm{Subseq}_k(l) = \mathrm{Subseq}_k(j)$ and $i \sim_k l \sim_k j$. Hence, the equivalence classes of the set of suffixes of w w.r.t. $\sim_k$ correspond to sets of consecutive indices (i.e., intervals) in $[1 : |w|]$, namely the starting positions of the suffixes in each class. We call these classes *k-blocks*.

A k-block consisting only of a single position (i.e., it is a *singleton-k-block*), remains an ℓ-block for all $\ell > k$. For a k-block $b = [m_b : n_b]$, m_b is its starting position and n_b its ending position. For the ending position of a $k-$block we also use the following definition.

▶ **Definition 4.** *For some $k > 0$, if $i \sim_{k-1} i+1$ and $i \nsim_k i+1$, then we will say that i splits its $(k-1)$-block or that i is a k-splitting position.*

If $a = [m_a : n_a]$ is a k-block and $b = [m_b : n_b]$ is a $(k+1)$-block with $m_a \leq m_b \leq n_b \leq n_a$, then we say that b is a $(k+1)-$block in a (alternatively, of a).

Since, for $1 \leq i, j \leq n$, $i \sim_k j$ holds if $i \sim_{k+1} j$, the relation $\sim_{k+1}$ is a refinement of $\sim_k$. In our setting, this means that the $(k+1)-$blocks of w are obtained by partitioning the k-blocks of w into subintervals. To obtain a partition of the positions of w into equivalence classes and the corresponding blocks, we can use this refinement property. We get the following inductive procedure.

0-blocks. For any i, with $1 \leq i \leq n$, we have $\mathrm{Subseq}_0(i) = \{\varepsilon\}$. Thus, we refer to $[1 : n]$ as the 0-*block* of w. Note, however, that position n cannot be referred to as a 0-splitting position.

$(k+1)$-blocks. For $k \geq 0$ and a k-block $a = [m_a : n_a]$ in w with $|a| \geq 2$, we can find the $(k+1)$-splitting positions inside of a. Except for the case when a is a 0-block, position n_a marks a k-splitting position. So, if $k > 0$, we slice off position n_a and obtain a truncated block $a^\dagger$; if $k = 0$, then $a^\dagger = a$. Going from right to left through $a^\dagger$, the position of every character we encounter for the first time (so, which we have not seen before in this

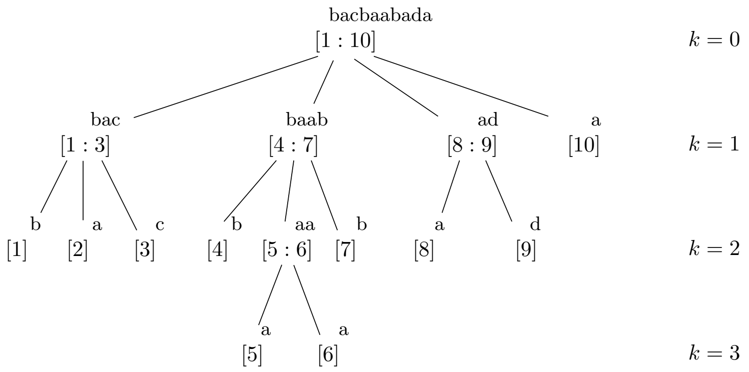


Figure 1 Simon-Tree of the word *bacbaabada*. Above each block $[i : j]$ we wrote the word $w[i : j]$.

traversal of $a^\dagger$) is a splitting position of $a^\dagger$. Consequently, those splitting positions and n_a (only if $k > 0$) will split a into $(k + 1)$-blocks. The correctness of this approach follows from Lemma 5.

▶ **Lemma 5.** *Let $a = [m_a, n_a]$ be a k-block with $1 \leq m_a < n_a \leq n$. Let $a^\dagger = a$ for $k = 0$ and $a^\dagger = [m_a : n_a - 1]$ for $k > 0$. Then the following holds for all $i, j \in a^\dagger$:*

(i) *if $k > 0$ then $i \nsim_{k+1} n_a$;*

(ii) *$i \sim_{k+1} j$ if and only if $\mathrm{Subseq}_1(i, a^\dagger) = \mathrm{Subseq}_1(j, a^\dagger)$.*

3.2 Simon-Tree definition

Before introducing the Simon-Tree, we recall some basic notions. An ordered rooted tree is a rooted tree which has a specified order for the subtrees of a node. We say that the depth of a node is the length of the unique simple path from the root to that node. Generally, the nodes with smaller depth are said to be *higher* (the root is the highest node with depth 0), while the nodes with greater depth are *lower* in the tree.

We can now define a new data structure called *Simon-Tree*. The Simon-Tree of a word w gives us a hierarchical representation of the equivalence classes inside of w. While an example of a Simon-Tree can be seen in Figure 1, the formal definition of a Simon-Tree is as follows.

▶ **Definition 6.** *The* Simon-Tree *T_w associated to the word w, with $|w| = n$, is an ordered rooted tree. The nodes of depth k represent $k-$blocks of w, for $0 \leq k \leq n$, and are defined recursively.*

- *The root corresponds to the 0-block of the word w, i.e., the interval $[1 : n]$.*
- *For $k > 1$ and for a node a of depth $k - 1$, which represents a $(k - 1)$-block $[i : j]$ with $i < j$, the children of a are exactly the blocks of the partition of $[i : j]$ in k-blocks, ordered decreasingly (right-to-left) by their starting position.*
- *For $k > 1$, each node of depth $k - 1$ which represents a singleton-$(k - 1)$-block is a leaf.*

The nodes of depth k in a tree T_w are called *explicit k-nodes (or simply k-nodes)*; by abuse of notation, we identify each k-node by the k-block it represents.

With respect to their starting positions in the word, we number the children nodes (which are blocks) of a node b from right to left. That is, the i^{th} child of b is the i^{th} block of the partition of a, counted from right to left. The singleton-j-blocks, for $j < k$, are also k-blocks, but they do not appear explicitly as nodes of depth k in the tree T_w. We will say that they are *implicit k-nodes*. In other words, an explicit singleton-j-node is an implicit k-node, for all $k > j$, and the only child of a k-node $[i : i]$ is the implicit $(k+1)$-node $[i : i]$. The nodes of depth k in the Simon-Tree T_w do not necessarily comprise all the k-blocks of w, but they contain explicitly exactly those k-blocks of w that were obtained by non-trivially splitting a $(k-1)$-block of w which was not a singleton.

3.3 Simon-Tree construction

We are interested in constructing the Simon-Tree T_w associated to a word w, with $|w| = n$, in linear time. In this section we give a description of the construction algorithm and its analysis. The corresponding pseudocode can be seen in Algorithms 1–3.

For the algorithms, we use the array X of size n which holds, for a given position i, the next position of $w[i]$ in the word $w[i+1 : n]$. We formally define this with $X[i] = \min\{j \mid w[j] = w[i], j > i\}$, while we assume $X[i] = \infty$ if $w[i] \notin \text{alph}(w[i+1 : n])$. The array can be calculated in $\mathcal{O}(n)$ time and space as stated in the full version of this paper [24]. As an example consider now the word $w = \texttt{bacbaabada}$. The array X is then depicted as follows.

i	1	2	3	4	5	6	7	8	9	10	11
$w[i]$	b	a	c	b	a	a	b	a	d	a	\$
$X[i]$	4	5	∞	7	6	8	∞	10	∞	∞	∞

When applying our algorithm to w, we get the tree shown in Figure 1, where we represent each node with the block $[i : j]$ it represents accompanied by the word $w[i : j]$.

Algorithm 1 Building the Simon-Tree T_w for a word w.

```
Input: Word $w$ with $|w| = n$
Result: Simon-Tree $T_w$
$w' \leftarrow w\$$;
Let $T$ be the tree with the root associated to the block $[? : n+1]$ of $w'$;
Let $p$ be a pointer to the root of $T$;
Compute the array $X[i]$;
for $i$=$n$ to $1$ do
    $a \leftarrow$ findNode($i$,$T$,$p$);
    ($T$,$p$) $\leftarrow$ splitNode($i$,$T$,$a$);
end
Set starting position for all blocks from leftmost branch including the root to 1;
Remove $-letter from tree: Remove the node associated to $[n+1 : n+1]$ from $T$ and
  set all right ends $r$ of blocks on the rightmost branch to $n$;
return T ;
```

Algorithm description. In general, we consider the individual letters of the word w from right to left. After considering $w[i]$, the tree we constructed so far corresponds to the Simon-Tree of the suffix $w[i : n]$. By traversing the word from right to left, we also construct the Simon-Tree in a right-to-left manner. Accordingly, it holds that at each time step only the nodes on the leftmost branch of the tree are possible to be enhanced. This is because for the tree of the word $w[i+1 : n]$, prepending a new letter to the word $w[i+1 : n]$ can only affect the leftmost node/block on each level of the tree, as the nodes of level k store the

Algorithm 2 findNode.

```
Input: Position $i$ in $w$, Simon-Tree $T_{w[i+1:n]\$}$, Pointer to leftmost leaf $a$ of $T_{w[i+1:n]\$}$
Result: Pointer to node on leftmost branch of $T_{w[i+1:n]\$}$
while $a$ is not the root of $T_{w[i+1:n]\$}$ do
    $r \leftarrow$ ending position of the block represented by $a$;
    $r_p \leftarrow$ ending position of block represented by $parent(a)$;
    if $X[i] \geq r$ and $X[i] < r_p$ then
        return $a$;
    else
        Close the block represented by $a$: Set its starting position to $i+1$;
        $a \leftarrow parent(a)$;
    end
end
return $a$;
```

Algorithm 3 splitNode.

```
Input: Position $i$, Simon-Tree $T_{w[i+1:n]\$}$, Pointer to node $a$ on leftmost branch of
       $T_{w[i+1:n]\$}$
Result: Simon-Tree $T_{w[i:n]\$}$, Pointer to leftmost leaf of $T_{w[i:n]\$}$
$T \leftarrow T_{w[i+1:n]\$}$ ;
if $a$ is the leftmost leaf of $T$ then
    // node $a$ represents the open block $[? : i+1]$
    Add a child to $a$ associated to the now completed block $[i+1 : i+1]$;
    Add a leftmost child $b$ to $a$ associated to the open block $[? : i]$;
else
    Add a leftmost child $b$ to $a$ associated to the open block $[? : i]$;
end
return $(T$, pointer to $b)$
```

k-blocks, and, accordingly, build a (possibly intermittent, if we only consider the explicit nodes) partition of the word w into non-overlapping intervals, for all k, while the nodes of one level are ordered with regard to their position in the word.

This means that a newly considered position of our word can be only added to a node on the leftmost branch of the tree that was constructed so far during the application of the algorithm. Therefore, we call the nodes on the leftmost branch *open blocks*. These open blocks are not complete and have a yet unknown starting position. We use $[? : i]$ to denote the open block with unknown starting position and ending position i. For the nodes on the leftmost branch, we only store the ending position (or splitting position) of their represented block. For all nodes that are not on the leftmost branch in the tree, we store both starting and ending position of their represented block.

In the beginning of the construction algorithm, we append the letter \$ at the end of w to ensure that all the positions of w are treated in a uniform way. More precisely, the usage of the \$-letter allows us to uniformly find the splitting points in a block according to case (ii) of Lemma 5 only. That is, by adding the letter \$ at the end, we avoid position n as being falsely recognized as a 0-splitting position since it is the ending position of the 0-block $[1 : n]$ of w. As seen in Algorithm 1, we define $w' = w\$$ and start the algorithm with the tree that only has one node, the root, representing the open block $[? : n + 1]$.

When considering a new position i of the word, and, essentially, inserting it into the current tree, we want to find the correct tree level where position i would mark the splitting of a new block or a new node, respectively. According to Algorithm 2, by starting at the leftmost leaf (which is the node associated to the open block $[? : i + 1]$) and going up the leftmost branch of the current tree, we look for the first node where the character $w[i]$ occurs on a non-ending position. Let this be node A of depth k, representing an open k-block. Node A cannot be a leaf since leafs only represent singleton-blocks, consist therefore only of one position, and $w[i]$ could not occur on a non-ending position. Let the leftmost child of A be the node a of depth $k + 1$. By utilizing Lemma 5, we get the information that position i is a $(k + 2)$-splitting position in a, and consequently, our new block with ending position i is mapped to level $(k + 2)$ of the Simon-Tree. Following Algorithm 3, we then insert the new block $[? : i]$ in the respective level of the Simon-Tree as a leftmost child of node a.

All nodes we traversed from leftmost leaf up to node a represent l-blocks with $l \geq (k + 2)$. These blocks are closed during the process of finding the correct position as seen in Algorithm 2. Since i is a $(k + 2)$-splitting position we set the starting position for all open l-blocks, with $l \geq k + 2$, to $i + 1$.

It remains only to mention the special case, where we do not find an occurrence of $w[i]$ on our traversal from leftmost leaf to the root. In this case, the letter did not appear yet in the word. It therefore marks a 1-splitting position and as per Algorithm 2, we return the tree root, to which the block $[? : i]$ is then added as a leftmost child as per Algorithm 3.

Algorithm analysis. The pseudocode for our algorithm is shown in Algorithms 1–3. Theorem 7 states the main result of this section. While the correctness of the algorithm follows mainly from the explanations above, its linear running time requires an amortized analysis. We observe that for each position i in w we traverse t nodes (representing open blocks) while going up on the leftmost branch, then insert one leaf on the leftmost branch while closing the t traversed nodes and moving them all to the right of the inserted leaf (so out from the leftmost branch). As the total number of nodes in the Simon-Tree T_w is linear in $|w|$, and each node is inserted once and traversed once, the conclusion follows. For the interested reader we point out that our analysis resembles to a certain extent the one of the algorithm constructing the Carthesian-Tree for a set of numbers [33].

▶ **Theorem 7.** *Given a word w, with $|w| = n$, we can construct its Simon-Tree in $O(n)$ time.*

4 Connecting two Simon-Trees

In this section, we propose a linear-time algorithm for the MaxSimK problem. The general idea of this algorithm is to analyse simultaneously the Simon-Trees of the two input words s and t of length n and n', respectively, and establish a connection between their nodes.

4.1 The S-Connection

In our solution of MaxSimK, we construct a relation called S-Connection (abbreviation for Simon-Connection) between the nodes of the Simon-Trees T_s and T_t constructed from the two input words s and t.

▶ **Definition 8.** *The (explicit or implicit) k-node a of T_s and the (explicit or implicit) k-node b of T_t are S-connected (i.e., the pair (a, b) is in the S-Connection) if and only if $s[i : n] \sim_k t[j : n']$ for all positions i in block a and positions j in block b.*

If two k-nodes a and b are S-connected, we say that b is a's S-Connection (and vice versa). Additionally, if two nodes are S-connected, then the corresponding blocks are said to be S-connected too.

Adapted from the equivalence classes within a word, each explicit or implicit k-node of T_s can be S-connected to at most one k-node of T_t (since they are then representing blocks which on their part represent the same equivalence class of the set of suffixes of w w.r.t. $\sim_k$).

▶ Remark 9. The S-Connection is *non-crossing.* This means that if the k-block $a = [m_a : n_a]$ of T_s is S-connected to the k-block $b = [m_b : n_b]$ of T_t, the k'-block $c = [m_c : n_c]$ of T_s is S-connected to the k'-block $d = [m_d : n_d]$ of T_t, and $m_a < m_c$, then $m_b < m_d$. Similarly, if $n_a < n_c$ then $n_b < n_d$.

4.2 The P-Connection

For constructing the S-Connection efficiently, we define a coarser relation called P-Connection (abbreviation for potential-connection) that covers the S-Connection. The P-Connection defines, for each node of T_s, a unique node of T_t to which it may be S-connected. Later, we will attempt to determine and *split*, for each level k from 1 to maximally n, all pairs of (explicit and implicit) k-nodes which were P-connected but are not S-connected. In a sense, this splitting allows us to gradually refine the P-Connection until we get exactly the S-Connection. The P-Connection for the words s and t is defined as follows.

▶ **Definition 10.** *The 0-nodes of T_s and T_t are P-connected. For all levels k of T_s, if the explicit or implicit k-nodes a and b (from T_s and T_t, respectively) are P-connected, then the i^{th} child of a is P-connected to the i^{th} child of b, for all i. No other nodes are P-connected.*

If k-nodes a and b are P-connected, we say that b is a's P-Connection (and vice versa).

According to its definition, the P-Connection can be computed efficiently in a straightforward manner. This definition is essentially based on the following Lemma 11. However, because Lemma 11 is not both necessary and sufficient (unlike, e.g., Lemma 5), it can only be used to define a relation coarser than the S-Connection and cannot be used to characterise (and, consequently, compute in a simple way) the S-Connection itself. Recall that in Simon-Trees the children of a node are numbered right to left.

▶ **Lemma 11.** *Let $k \geq 1$. Let $a = [m_a : n_a]$ be a k-block in s and $b = [m_b : n_b]$ a k-block in t with $a \sim_k b$. Then the i^{th} child of the node a of T_s can only be S-connected (but it is not necessarily connected) to the i^{th} child of the node b of T_t, for all $i \geq 1$.*

It is not hard to see that, in the spirit of Remark 9, the P-Connection is non-crossing. Moreover, by Lemma 11, if the k-blocks a and b are S-connected, they are also P-connected. It is very important to note that a pair of nodes whose parent-nodes are not S-connected is also not S-connected. So, as our approach is to refine the P-Connection till the S-Connection is reached, we can immediately decide that a pair of nodes (a, b) is not in the S-Connection when the pair consisting of their respective parent-nodes is not in the S-Connection.

4.3 From P- to S-Connection

Preliminary transformation. As mentioned, our algorithm solving MaxSimK uses the Simon-Trees of s and t. To make the exposure simpler, we make the following simple transformation of the trees. If a is a k-node such that a is a singleton, we add as a child of this node a $(k+1)$-node representing the same block a (this was an implicit node before, now made explicit); the newly added node on level $k+1$ does not have any children (i.e., this procedure is not applied recursively). Before, by Lemma 5, all blocks $[i:i]$ of w appeared explicitly exactly once in T_w. Therefore, each singleton-block $[i:i]$ of s (respectively, t) appears now exactly twice in T_s (respectively, T_t).

In general, these now explicit nodes are used to guarantee the existence of a P-connected node (implicit or explicit) for every explicit singleton node on some level $k+1$ that was on a splitting position on level k, so we can determine singleton nodes that are $\sim_k$- but not $\sim_{k+1}$-congruent to the corresponding nodes in the other tree. The transformation has the following direct consequence that we will use: each singleton-block a appears now on two consecutive levels. While the node corresponding to a on the higher level may be S-connected to a node corresponding to a non-singleton-block, the node corresponding to a on the lower level may be S-connected only to a singleton-node.

As a second consequence, it is worth noting that explicit nodes might be connected to implicit nodes, too. However, this is only true for explicit nodes which were added during the transformation described above, i.e., singleton explicit nodes. Explicit nodes which are not singletons cannot be connected to implicit nodes.

Refining the P-Connection. The main step of our approach is, while considering the levels of the trees T_s and T_t in increasing order, to identify the pairs of P-connected nodes from the respective levels which are not S-connected and consequently *split* them. At the same time, we identify the pairs of singleton-blocks occurring explicitly on higher levels (and only implicitly on the current levels) which are not S-connected on this level, and also *split* them *on the current level.* For simplicity of exposure, when we split two k-blocks, we say that we k-split them. In order to implement this idea, we use the following Lemma 12 to define a splitting criterium.

We introduce first some notations. For $w \in \{s,t\}$, a position $j \leq |w|$, and a letter x, we define $\texttt{next}_w(j,x)$ as the leftmost position where x occurs in $w[j:|w|]$, or as ∞ if $x \notin \mathrm{alph}(w[j:|w|])$. For a block $a = [m_a : n_a]$ of the word w and a letter x, we define $\texttt{next}_w(a,x) = \texttt{next}_w(n_a,x)$. We generally omit the subscript w when it is clear from the context. Furthermore, we define $\mathrm{alph}(a)$ for a block $a = [m_a : n_a]$ as $\mathrm{alph}(w[m_a : n_a])$.

▶ **Lemma 12.** *Let $k \geq 1$. Let $a = [m_a : n_a]$ be a k-block in the word s and $b = [m_b : n_b]$ a k-block in the word t with $a \sim_k b$. Let $a' = [m_{a'} : n_{a'}]$ be a $(k+1)$-block in a and $b' = [m_{b'} : n_{b'}]$ be a $(k+1)$-block in b. Then $a' \nsim_{k+1} b'$ if and only if there exists a letter x such that $s[\texttt{next}(a',x)+1 : n] \nsim_k t[\texttt{next}(b',x)+1 : n']$.*

The main idea of this lemma (illustrated in Figure 2) is that two $(k+1)$-blocks a' and b' are not S-connected, although their parents were S-connected, if and only if we can find a letter x such that $s[\texttt{next}(a',x)+1 : n]$ and $t[\texttt{next}(b',x)+1 : n']$ are not k-equivalent but $(k-1)$-equivalent. That is, $\texttt{next}(a',x)+1$ and $\texttt{next}(b',x)+1$ should occur, respectively, in two k-blocks which were split, but whose parents were S-connected. A word distinguishing the suffixes starting in a' from those starting in b' has the first letter x, and is continued by the word of length k which distinguishes $s[\texttt{next}(a',x)+1 : n]$ and $t[\texttt{next}(b',x)+1 : n']$.

Identifying P-connected pairs to be split. When going through the trees level by level, the 1-blocks (all occuring explicitly on level 1 of T_s and respectively T_t) which are S-connected can be easily and efficiently identified: the i^{th} node $a = [m_a : n_a]$ on level 1 of T_s is connected to the i^{th} node $b = [m_b : n_b]$ of T_t if and only if $\mathrm{alph}(s[n_a : n]) = \mathrm{alph}(t[n_b : n'])$. All the other P-connected pairs of 1-blocks are not S-connected, so they are 1-split.

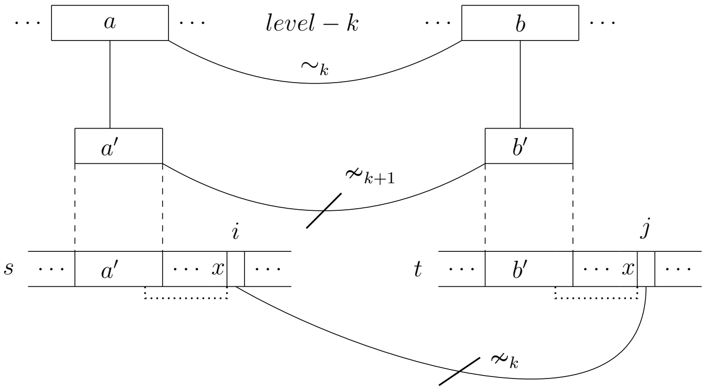


Figure 2 Illustration of Lemma 12.

The identification of the pairs of $(k+1)$-blocks and pairs of singletons which need to be $(k+1)$-split is based on Lemma 12. The idea is the following. A pair of P-connected $(k+1)$-blocks $a' = [m_{a'} : n_{a'}]$ of T_s and $b' = [m_{b'} : n_{b'}]$ of T_t is not S-connected if and only if there exists a letter x such that $s[\texttt{next}(a', x) + 1 : n] \nsim_k t[\texttt{next}(b', x) + 1 : n']$. So, in order to be able to $(k+1)$-split two nodes (whose parents are S-connected), we need to identify two positions i and j (and a corresponding letter x), with $i = \texttt{next}(a', x) + 1$ and $j = \texttt{next}(b', x) + 1$ which were k-split but not $(k-1)$-split. We search for position i inside the k-blocks of T_s, and try to see where position j may occur in the blocks of T_t such that these two positions are not in S-connected k-blocks. To find the position i (and the corresponding j) we analyse two cases.

The first case (A) is when i occurs inside an implicit k-node, which is the singleton-k-block $[i : i]$. On the lowest level where this block appeared as an explicit node, it was S-connected to a node $[j : j]$ representing a singleton too, according to Section 4.1 and Lemma 5. Thus, position i can only be k-split from the position j of t to which it was S-connected (it was already disconnected from all other positions on level ℓ). If i and j are both directly preceded by the same symbol (say x), then the pair (i, j) gives us exactly the positions we were searching for.

The second case (B) is when i occurs inside an explicit k-node in T_s. Let $A = [m_A : n_A]$ and $B = [m_B : n_B]$ be two $(k-1)$ blocks from T_s and T_t, respectively, such that $A \sim_{k-1} B$, and $a = [m_a : n_a]$ and $b = [m_b : n_b]$ be the ℓ^{th} child of A and B, respectively. Clearly, b might be explicit, implicit, or even empty. If b is non-empty, the following holds. All positions of a are k-split from the positions $[m_B : m_b - 1]$ and from the positions $[n_b + 1 : n_B]$, because a is not P-connected to the blocks covering those positions. Also, if a and b are not S-connected then all positions of a are also k-split from the positions of b.

An example. Let $s = acab$ and $t = acabba$ be two words. Their respective Simon-Trees T_s and T_t are depicted in Figure 3 along with their P- and S-Connection. Note that for the sake of not crossing edges and simplifying the presentation, the Simon-Trees in Figure 3 are rotated by 90 degrees to the right and to the left, respectively. Thus, the roots of the trees are on the outer left and right side of the figure. Additionally, the tree T_t on the right is mirrored, so that nodes from a P-connected pair are vis-à-vis. Also, the trees already contain the singleton nodes that were originally implicit but are now made explicit by our aforementioned

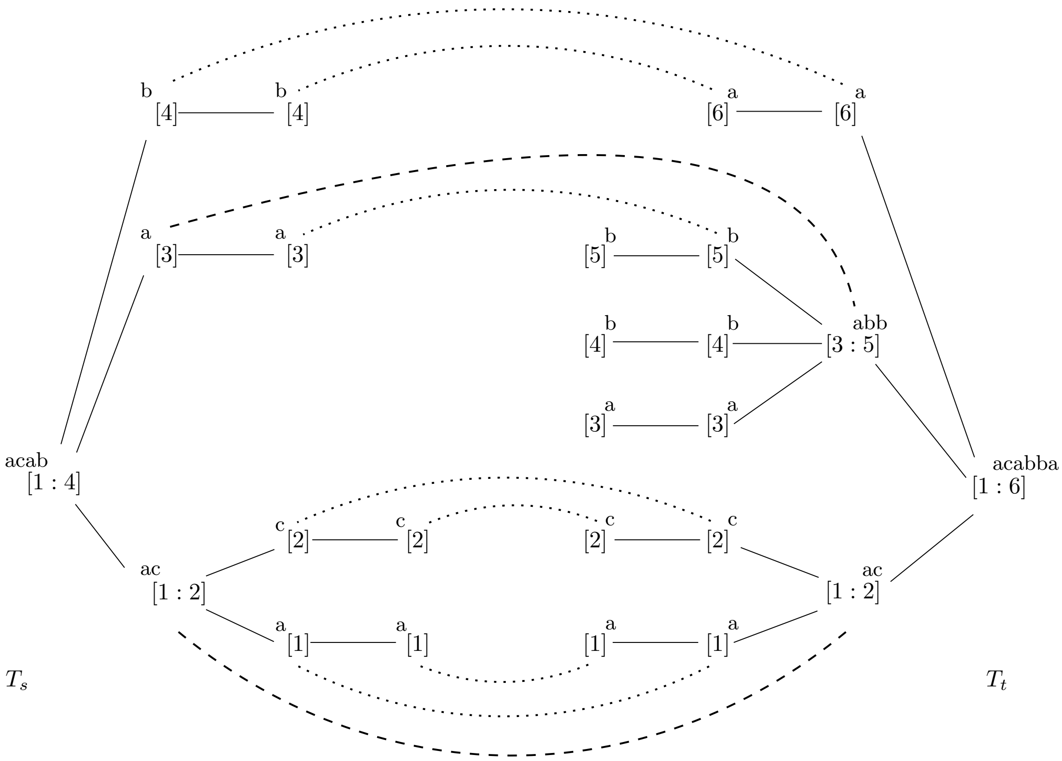


Figure 3 Two transformed Simon-Trees with their P- and S-Connection.

transformation. From the figures it becomes also clear that this transformation is needed in the case of the singleton-node $[5 : 5]$ from the 2nd level of T_t which is P-connected to the singleton-node $[3 : 3]$ from the 2nd level of T_s.

In the beginning we are considering all possibly connected blocks by determining all P-connected pairs. While the dashed and dotted lines connect, respectively, the nodes of all the pairs from the P-Connection, the S-Connection is obtained by splitting step by step P-connected pairs that cannot be equivalent with regard to their respective level. The dotted edges symbolize exactly these split pairs, and in the end, the S-Connection consists only of the pairs connected with a dashed edge.

Following Theorems 16 and 17 stated at the very end of this paper, we get the largest k for which the two words s and t are k-equivalent by finding the largest k for which the k-blocks containing position 1 of both words are S-connected. In our example, the blocks $s[1 : 4]$ and $t[1 : 6]$ representing the complete words are naturally 0-equivalent. Furthermore, as seen in Figure 3, the blocks $s[1 : 2]$ and $t[1 : 2]$ on level 1 are S-connected, but the blocks $s[1 : 1]$ and $t[1 : 1]$ are not S-connected on level 2. Thus, the largest k for which $s \sim_k t$ holds is 1.

Path to efficiency. Taking (i, j) to be each position of block a paired with each of the positions from which it is k-split, according to the above, might not be efficient. However, the combinatorial Lemma 12 allows us to switch slightly the point of view and ultimately obtain an efficient method. We will traverse the kth level of a Simon-Tree T_s from right to left and when considering a node a on this level, our approach is to determine which nodes should be $k + 1$-split due to any pair (i, j), where i is some position occurring in the block a.

The mechanism allowing us to do this is stated in Lemma 13, and this essentially explains how to determine all the $k+1$-splits determined by positions of a (and their corresponding pairing from b). We show how to proceed in both cases (A) and (B). Clearly, a symmetrical approach would also work (so looking at nodes in T_t and positions in t).

Firstly, we need a few more notations. For a block $a = [m_a : n_a]$ of s or t and a letter x, let $\texttt{prev}(a, x)$ be the rightmost occurrence of x in $s[1 : m_a - 2]$ (or 0 if $x \notin \mathrm{alph}(s[1 : m_a - 2])$), and let $\texttt{right}(a, x)$ be the rightmost occurrence of x in $s[1 : n_a - 1]$ (or 0 if $x \notin \mathrm{alph}(s[1 : n_a - 1])$).

The setting in which Lemma 13 is stated is the following. We have two P-connected k-blocks $a = [m_a : n_a]$ and $b = [m_b : n_b]$ from T_s and T_t, respectively, whose parent-nodes (explicit or implicit) are S-connected. The lemma defines a necessary and sufficient condition for a pair of (explicit or implicit) $(k+1)$-nodes (a', b') to be $(k+1)$-split because there exists a letter x and a pair of positions (i, j), with $i = \texttt{next}(a', x)+1$, $i \in [m_a : n_a]$, $j = \texttt{next}(b', x)+1$, and $s[i : n] \not\sim_k t[j : n']$. Such a pair (a', b') is called $(a, k+1)$-split (that is, a causes the respective split on level $k+1$). Note that a' and b' are the (explicit or implicit) children of either a and b or of two k-blocks which are left of a and b. In any way, their parents are S-connected; otherwise a' and b' would have already been split on a higher level.

This setting is also illustrated in Figure 4.

▶ **Lemma 13.** *For $k \geq 1$, let $a = [m_a : n_a]$ be a k-block in s and $b = [m_b : n_b]$ its P-Connection (which is a k-block in t). Then a pair of P-connected $(k+1)$-blocks $a' = [m_{a'} : n_{a'}]$ in s and $b' = [m_{b'} : n_{b'}]$ in t is $(a, k+1)$-split if and only if there exists a letter x in alph$(s[m_a - 1 : n_a - 1])$ such that at least one of the following holds:*

1. *a' ends strictly between $\texttt{prev}(a, x)$ and m_a (i.e., $\texttt{prev}(a, x) < n_{a'} < m_a$), and b' ends to the left of $\texttt{prev}(b, x) + 1$ (i.e., $n_{b'} \leq \texttt{prev}(b, x)$).*
2. *a' ends between $\texttt{prev}(a, x)$ and $\texttt{right}(a, x)$ (i.e., $\texttt{prev}(a, x) < n_{a'} \leq \texttt{right}(a, x)$), and b' ends between $\texttt{right}(b, x)$ and n_b (i.e., $\texttt{right}(b, x) < n_{b'} \leq n_b$).*
3. *$a \not\sim_k b$, a' ends between $\texttt{prev}(a, x)$ and $\texttt{right}(a, x)$ (i.e., $\texttt{prev}(a, x) < n_{a'} \leq \texttt{right}(a, x)$), and b' ends between $\texttt{prev}(b, x)$ and $\texttt{right}(b, x)$ (i.e., $\texttt{prev}(b, x) < n_{b'} \leq \texttt{right}(b, x)$).*

In Lemma 13, because $k \geq 1$ and the blocks a' and b' are the (explicit or implicit) children of two S-connected k-blocks, it follows that $\mathrm{alph}(s[n_{a'} : n]) = \mathrm{alph}(t[n_{b'} : n'])$. This means, in particular, that $\texttt{next}(a', x) \neq \infty$ for some letter x if and only if $\texttt{next}(b', x) \neq \infty$.

Now, we can explain how to algorithmically apply Lemma 13 and find the pairs of $(k+1)$-blocks which should be split. For this, we can define, and compute in the step where the k-split pairs were obtained, a list L_k of pairs of singleton-k-blocks which were k-split and a list H_k of all the explicit k-nodes of T_s and their P-connections.

We first consider each explicit k-node a of T_s and its P-Connection, the node b of T_t (in both cases: when a and b were k-split or when they were not). For $x \in \mathrm{alph}(s[m_a - 1 : n_a - 1]) \cup \{s[m_a - 1]\}$ (note that the symbols $x \in \mathrm{alph}(s[m_a - 1 : n_a - 1])$ can be identified as the first symbols of the $(k+1)$-blocks into which $a = s[m_a : n_a]$ is split, except the rightmost one; these are the children of node a except the rightmost one) we do the following:

1. identify each $(k+1)$-block $a' = [m_{a'} : n_{a'}]$ with $\texttt{prev}(a, x) < n_{a'} < m_a$ and its pair $b' = [m_{b'} : n_{b'}]$. Then (a', b') is not in the S-Connection if $n_{b'} \leq \texttt{prev}(b, x)$ (i.e., a' and b' are $(a, k+1)$-split).
2. identify each $(k+1)$-block $a' = [m_{a'} : n_{a'}]$ with $\texttt{prev}(a, x) < n_{a'} \leq \texttt{right}(a, x)$ and its pair $b' = [m_{b'} : n_{b'}]$. Then (a', b') is not in the S-Connection if $\texttt{right}(b, x) < n_{b'} \leq n_b$.
3. if $a \not\sim_k b$, identify each $(k+1)$-block a' with $\texttt{prev}(a, x) < n_{a'} \leq \texttt{right}(a, x)$ and its pair $b' = [m_{b'} : n_{b'}]$. Then (a', b') is not in the S-Connection if $\texttt{prev}(b, x) < n_{b'} \leq \texttt{right}(b, x)$.

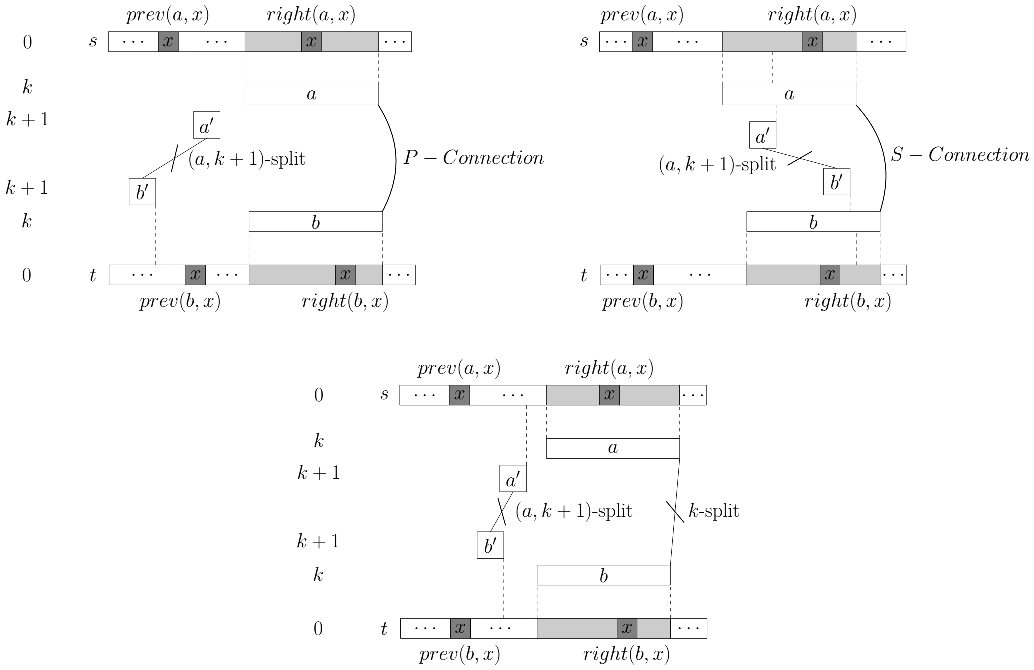


Figure 4 Illustration of the three cases of Lemma 13.

For every pair (a, b) of singleton-k-blocks which were k-split (from the list L_k), we only perform step 3 from above.

For each k-block a we considered (explicit or implicit node of T_s), we collect the singleton-$(k + 1)$-blocks that were $(a, k + 1)$-split, to be used when computing the $(k + 2)$-splits.

The next step is to implement this idea, i.e., to describe data structures allowing us to identify efficiently the $(k + 1)$-blocks a' and b' from above. We say that a pair of blocks/nodes (a', b') meets an interval-pair $([p : q], [p' : q'])$ if a' ends in $[p : q]$, and b' ends in $[p' : q']$.

Our approach is the following. We process the blocks on level k and, for each of them, get (at most) three lists of interval-pairs (one component is an interval of positions in s, the other an interval in t). On level $k + 1$, we split each pair of P-connected blocks (a', b') which meets one interval-pair from our list. A crucial property here is that, for each interval-pair, the $(k + 1)$-blocks of s which meet it, and are accordingly split from their P-Connections, are consecutive (explicit and implicit) $(k + 1)$-nodes in T_s. Thus, in order to make use of Lemma 13, we draw on the technical results given by Lemmas 14 and 15.

▶ **Lemma 14.** *Let $a = [m_a : n_a]$ and $b = [m_b : n_b]$ be P-connected blocks of T_s and T_t, respectively, and $s_a = s[m_a - 1 : n_a - 1]$. We can compute in overall $O(|alph(a)|)$ time the three lists, associated to the pair (a, b), containing:*

1. *the interval-pairs $([\texttt{prev}(a, x) + 1 : m_a - 1], [0 : \texttt{prev}(b, x)])$, for all $x \in alph(s_a)$;*
2. *the interval-pairs $([\texttt{prev}(a, x)+1 : \texttt{right}(a, x)], [\texttt{right}(b, x)+1 : n_b])$, for all $x \in alph(s_a)$;*
3. *the interval-pairs $([\texttt{prev}(a, x) + 1 : \texttt{right}(a, x)], [\texttt{prev}(b, x) + 1 : \texttt{right}(b, x)])$, for all $x \in alph(s_a)$.*

▶ **Lemma 15.** *Given two words s and t, with $|s| = n$ and $|t| = n'$, $n \geq n'$, and their Simon-Trees T_s and T_t, we can check in $O(n)$ overall time for all pairs of P-connected 1-blocks (a, b), with $a = [m_a : n_a]$ and $b = [m_b : n_b]$, whether $alph(s[n_a : n]) = alph(t[n_b : n'])$.*

Efficiently constructing the S-Connection and solving MaxSimK. Based on the previous lemmas, we can now show our main technical theorem. We use Lemma 15 to see which 1-nodes are not S-connected. This is done in $O(n)$ time. Then consider the k-nodes, for each $k \geq 2$ in increasing order. For each pair (a, b) of $(k-1)$-nodes which were split (i.e., removed from the S-Connection) in the previous step, we split the pairs of k-nodes meeting one of the interval-pairs of the three lists of (a, b), as computed in Lemma 14. To do this efficiently, we maintain an interval union-find and an interval split-find structure for each word. While the concrete algorithm can be found in the full version of this paper [24], we can now state our main result in Theorem 16.

▶ **Theorem 16.** *Given two words s and t, with $|s| = n$ and $|t| = n'$, $n \geq n'$, we can compute in $O(n)$ time the following:*

- *the S-Connection between the nodes of the two trees T_s and T_t;*
- *for each $i \in [1 : n]$, the highest level k on which the (implicit or explicit) node $[i : i]$ is k-split from its P-Connection.*

Finally, in order to solve MaxSimK, we need to compute the largest k for which the k-block $a = [1 : n_a]$ of s is S-connected to the k-block $b = [1 : n_b]$ of t. Thus, we execute the algorithm of Section 4.3 and the aforementioned level k can be easily found by checking, level by level, the blocks that contain position 1 of s on each level of T_s and the block to which they are S-connected in T_t. As a consequence of Theorem 16, we can now show our main result.

▶ **Theorem 17.** *Given two words s and t, with $|s| = n$ and $|t| = n'$, $n \geq n'$, we can solve MaxSimK and compute a distinguishing word of minimum length for s and t in $O(n)$ time.*

5 Conclusions and future work

In this paper, we presented the first algorithm solving MaxSimK in optimal time. This algorithm is based on the definition and efficient construction of a novel data-structure: the Simon-Tree associated to a word. Our algorithm constructs the respective Simon-Trees for the two input words of MaxSimK, and then establishes a connection between their nodes. While the Simon-Tree is a representation of the classes induced, for all $k \geq 1$, by the $\sim_k$-congruences on the set of suffixes of a word, this connection allows us to put together the classes induced by the respective congruences on the set of suffixes of both input word, and to obtain, as a byproduct, the answer to MaxSimK.

The work presented in this paper can be continued naturally in several directions. For instance, it seems interesting to us to compute efficiently, for two words s and t, what is the largest k such that $\mathrm{Subseq}_{\leq k}(s) \subseteq \mathrm{Subseq}(t)$. Similarly, one could consider the following pattern-matching problem: given two words s and t, and a number k, compute efficiently all factors $t[i : j]$ of t such that $t[i : j] \sim_k s$. Finally, SimK could be extended to the following setting: given a word s and regular (or a context-free) language L, and a number k, decide efficiently whether there exists a word $t \in L$ such that $s \sim_k t$. A variant of MaxSimK can be also considered in this setting: given a word s and regular (or a context-free) language L, find the maximal k for which there exists a word $t \in L$ such that $s \sim_k t$.

References

1 Ricardo A. Baeza-Yates. Searching subsequences. *Theor. Comput. Sci.*, 78(2):363–376, 1991.

2 Laura Barker, Pamela Fleischmann, Katharina Harwardt, Florin Manea, and Dirk Nowotka. Scattered factor-universality of words. In *Proc. DLT 2020*, volume 12086 of *Lecture Notes in Computer Science*, pages 14–28. Springer, 2020.

3 Karl Bringmann and Bhaskar Ray Chaudhury. Sketching, streaming, and fine-grained complexity of (weighted) LCS. In *Proc. FSTTCS 2018*, volume 122 of *LIPIcs*, pages 40:1–40:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

4 Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proc. SODA 2018*, pages 1216–1235. SIAM, 2018.

5 Maxime Crochemore, Borivoj Melichar, and Zdenek Tronícek. Directed acyclic subsequence graph - overview. *J. Discrete Algorithms*, 1(3-4):255–280, 2003.

6 Joel D. Day, Pamela Fleischmann, Florin Manea, and Dirk Nowotka. k-spectra of weakly-c-balanced words. In *Proc. DLT 2019*, volume 11647 of *Lecture Notes in Computer Science*, pages 265–277. Springer, 2019.

7 Cees H. Elzinga, Sven Rahmann, and Hui Wang. Algorithms for subsequence combinatorics. *Theor. Comput. Sci.*, 409(3):394–404, 2008.

8 Lukas Fleischer and Manfred Kufleitner. Testing Simon's congruence. In *Proc. MFCS 2018*, volume 117 of *LIPIcs*, pages 62:1–62:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

9 Dominik D. Freydenberger, Pawel Gawrychowski, Juhani Karhumäki, Florin Manea, and Wojciech Rytter. Testing k-binomial equivalence. In Multidisciplinary Creativity*, a collection of papers dedicated to G. Păun 65th birthday*, pages 239–248, 2015. available in CoRR: `arXiv:1509.00622`.

10 Harold N. Gabow and Robert Endre Tarjan. A linear-time algorithm for a special case of disjoint set union. *J. Comput. Syst. Sci.*, 30(2):209–221, 1985.

11 Emmanuelle Garel. Minimal separators of two words. In *Proc. CPM 1993*, volume 684 of *Lecture Notes in Computer Science*, pages 35–53. Springer, 1993.

12 Simon Halfon, Philippe Schnoebelen, and Georg Zetzsche. Decidability, complexity, and expressiveness of first-order logic over the subword ordering. In *Proc. LICS 2017*, pages 1–12, 2017.

13 Jean-Jacques Hebrard. An algorithm for distinguishing efficiently bit-strings by their subsequences. *Theoretical Computer Science*, 82(1):35–49, 22 May 1991.

14 Hiroshi Imai and Takao Asano. Dynamic segment intersection search with applications. In *Proc. 25th FOCS, 1984*, pages 393–402, 1984.

15 Prateek Karandikar, Manfred Kufleitner, and Philippe Schnoebelen. On the index of Simon's congruence for piecewise testability. *Inf. Process. Lett.*, 115(4):515–519, 2015.

16 Prateek Karandikar and Philippe Schnoebelen. The height of piecewise-testable languages with applications in logical complexity. In *Proc. CSL 2016*, volume 62 of *LIPIcs*, pages 37:1–37:22, 2016.

17 Prateek Karandikar and Philippe Schnoebelen. The height of piecewise-testable languages and the complexity of the logic of subwords. *Logical Methods in Computer Science*, 15(2), 2019.

18 Dietrich Kuske and Georg Zetzsche. Languages ordered by the subword order. In *Proc. FOSSACS 2019*, volume 11425 of *Lecture Notes in Computer Science*, pages 348–364. Springer, 2019.

19 Marie Lejeune, Julien Leroy, and Michel Rigo. Computing the k-binomial complexity of the Thue-Morse word. In *Proc. DLT 2019*, volume 11647 of *Lecture Notes in Computer Science*, pages 278–291, 2019.

20 Julien Leroy, Michel Rigo, and Manon Stipulanti. Generalized Pascal triangle for binomial coefficients of words. *Electron. J. Combin.*, 24(1.44):36 pp., 2017.

21 M. Lothaire. *Combinatorics on Words*. Cambridge University Press, 1997.

22 David Maier. The complexity of some problems on subsequences and supersequences. *J. ACM*, 25(2):322–336, April 1978.

23 Alexandru Mateescu, Arto Salomaa, and Sheng Yu. Subword histories and Parikh matrices. *J. of Comput. Syst. Sci.*, 68(1):1–21, 2004.

24 Paweł Gawrychowski, Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. Efficiently testing Simon's congruence. *preprint, CoRR*, 2020. `arXiv:2005.01112`.

25 Michel Rigo and Pavel Salimov. Another generalization of abelian equivalence: Binomial complexity of infinite words. *Theor. Comput. Sci.*, 601:47–57, 2015.

26 Arto Salomaa. Connections between subwords and certain matrix mappings. *Theor. Comput. Sci.*, 340(2):188–203, 2005.

27 David Sankoff and Joseph Kruskal. *Time Warps, String Edits, and Macromolecules The Theory and Practice of Sequence Comparison.* Cambridge University Press, 2000 (reprinted). originally published in 1983.

28 Shinnosuke Seki. Absoluteness of subword inequality is undecidable. *Theor. Comput. Sci.*, 418:116–120, 2012.

29 Imre Simon. An algorithm to distinguish words efficiently by their subwords.

30 Imre Simon. Piecewise testable events. In *Proc. Autom. Theor. Form. Lang., 2nd GI Conf.*, volume 33 of *LNCS*, pages 214–222. Springer, 1975.

31 Imre Simon. Words distinguished by their subwords (extended abstract). In *Proc. WORDS 2003*, volume 27 of *TUCS General Publication*, pages 6–13, 2003.

32 Zdenek Tronícek. Common subsequence automaton. In *Proc. CIAA 2002 (Revised Papers)*, volume 2608 of *Lecture Notes in Computer Science*, pages 270–275, 2002.

33 Jean Vuillemin. A unifying look at data structures. *Commun. ACM*, 23(4):229–239, 1980.

34 Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, January 1974.

35 Georg Zetzsche. The complexity of downward closure comparisons. In *Proc. ICALP 2016*, volume 55 of *LIPIcs*, pages 123:1–123:14, 2016.

Finding an Optimal Alphabet Ordering for Lyndon Factorization Is Hard

Daniel Gibney
Department of Computer Science, University of Central Florida, Orlando, FL, USA

Sharma V. Thankachan
Department of Computer Science, University of Central Florida, Orlando, FL, USA

Abstract

This work establishes several strong hardness results on the problem of finding an ordering on a string's alphabet that either minimizes or maximizes the number of factors in that string's Lyndon factorization. In doing so, we demonstrate that these ordering problems are sufficiently complex to model a wide variety of ordering constraint satisfaction problems (OCSPs). Based on this, we prove that (i) the decision versions of both the minimization and maximization problems are NP-complete, (ii) for both the minimization and maximization problems there does not exist a constant approximation algorithm running in polynomial time under the Unique Game Conjecture and (iii) there does not exist an algorithm to solve the minimization problem in time $poly(|T|) \cdot 2^{o(\sigma \log \sigma)}$ for a string T over an alphabet of size σ under the Exponential Time Hypothesis (essentially the brute force approach of trying every alphabet order is hard to improve significantly).

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

Keywords and phrases Lyndon Factorization, String Algorithms, Burrows-Wheeler Transform

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.35

Funding Supported in part by the U.S. National Science Foundation (NSF) under CCF-1703489.

1 Introduction

A Lyndon word is a string that is lexicographically strictly smallest among all of its cyclic shifts. Letting $\circ$ denote concatenation, the Lyndon factorization of a string T is the factorization of T into Lyndon words T_1, T_2, …, T_f that are lexicographically non-increasing and $T = T_1 \circ T_2 \circ \ldots \circ T_f$. For example, the Lyndon factorization of $0, 1, 0, 0, 2, 1, 1, 0, 0, 1, 0, 1, 1, 2$ is $(0, 1), (0, 0, 2, 1, 1), (0, 0, 1, 0, 1, 1, 2)$, assuming the usual ordering, $0 < 1 < 2$.

Lyndon words and Lyndon factorization are well-studied, and play an important role in string algorithms [1, 2, 10, 24, 28, 30], algebra and combinatorics [7, 17, 25], and data compression [12, 18, 20, 34, 35]. As an example, it was shown in [29] that local suffixes inside each Lyndon factor can be sorted independently and then merged to construct a string's suffix array. As another example, Lyndon factorization is used in both the construction of a string's bijective Burrows-Wheeler transform (BBWT) [13] and in performing pattern matching on indexes built from the string's BBWT [3], where the number of steps used to locate occurrences of a pattern P depends on the number of Lyndon factors within a particular suffix of P. Because of such applications, it would be beneficial to be able to control the number of factors in the Lyndon factorization of a string. Unfortunately, the Lyndon factorization of a string is unique under a fixed ordering of its alphabet [26]. However, it can vary under different alphabet orderings. For instance, if we change the alphabet ordering to $2 < 0 < 1$ in our example above, we obtain the Lyndon factorization $(0, 1)$, (0), (0), $(2, 1, 1, 0, 0, 1, 0, 1, 1)$, (2). This leads to the following problems:

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 35; pp. 35:1–35:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

▶ **Problem 1** (Lyndon Factor Minimization – Decision Version)**.** Given an integer A and text T over alphabet Σ, does there exist an ordering on Σ such that the number of Lyndon factors of T is at most A?

▶ **Problem 2** (Lyndon Factor Maximization – Decision Version)**.** Given an integer A and text T over alphabet Σ, does there exist an ordering on Σ such that the number of Lyndon factors of T is at least A?

We will also consider the *optimization variants* of these problems. The objective cost of a solution is the number of factors in its Lyndon factorization. In particular, for the minimization problem, a λ-approximation for $\lambda > 1$, is a polynomial-time algorithm that outputs an alphabet ordering where the number of factors is at most λ times the minimum possible number of factors over all possible alphabet orderings. Similarly, for the maximization problem, a λ-approximation for $\lambda < 1$, is a polynomial-time algorithm that outputs an alphabet ordering where the number of factors is at least λ times the maximum number of possible factors over all possible alphabet orderings.

These problems were first considered by Clare and Daykin, who proposed a polynomial-time greedy algorithm that can be adjusted to provide either a small number of factors or a large number of factors [8]. Through experiments, the authors showed that the number of factors can be significantly affected by their algorithm. Another approach that uses evolutionary algorithms to find alphabet orderings to optimize the number of Lyndon factors was considered in [9] and in [27]. Again, it was shown that there is often a significant effect on the number of factors, which can be controlled by the use of different fitness functions within the evolutionary algorithms. These techniques, although appearing to have a significant impact on the number of factors, do not provide any approximation guarantee.

Hardness results for the problem of ordering the alphabet of a string to minimize the number of maximal unary substrings occurring in its Burrows-Wheeler Transform (BWT) appeared in [4]. Although the Lyndon factors of a string and the structure of its BBWT are closely related, we see no clear relation between the number of Lyndon factors of a string and the number of maximal unary substrings occurring in its BWT. Moreover, the techniques applied here seem quite different from those used in [4]. We present the following results.

▶ **Theorem 3.** *The decision version of Lyndon Factor Minimization is NP-complete.*

▶ **Theorem 4.** *Under the Exponential Time Hypothesis, the optimization version of Lyndon Factor Minimization cannot be solved in time poly*$(|T|) \cdot 2^{o(|\Sigma| \log |\Sigma|)}$.

▶ **Theorem 5.** *Under the Unique Games Conjecture, the optimization version of Lyndon Factor Minimization does not admit a λ-approximation for any constant $\lambda > 1$.*

▶ **Theorem 6.** *The decision version of Lyndon Factor Maximization is NP-complete.*

▶ **Theorem 7.** *Under the Unique Games Conjecture, the optimization version of Lyndon Factor Maximization does not admit a λ-approximation for any constant $\lambda < 1$.*

We will prove these theorems in Section 3.1, Section 3.2, Section 3.3, Section 4.1, and Section 4.2, respectively. We leave open whether it is possible to have a result similar to Theorem 4 for Lyndon Factor Maximization.

Our main line of attack is to model ordering constraint satisfaction problems (OCSPs), a subject of extensive research in its own right [5, 6, 15, 16, 31, 33]. In these problems, the task is to find a linear ordering on a set of variables subject to some additional constraints. Our work shows that a solver for these Lyndon factorization problems would be powerful enough to solve difficult OCSP instances. Our results make use of strings that allow us to model different constraint satisfaction problems and thus prove our hardness results.

2 Preliminaries

We denote the concatenation of the strings u and v using the "$\circ$" symbol, writing their concatenation as $u \circ v$. However, we omit "$\circ$" where the concatenation is clear from context. Throughout this paper, we will use "$<$" and "$>$" to refer to alphabet order between symbols, the lexicographic order between strings, and the usual ordering between real numbers. Again, context will make it clear which type of order is meant. A suffix of a string T is a string v such that $T = u \circ v$ for some string u. The suffix array of a string $T[1, n]$ is a length n array where the i^{th} element is equal to the starting index of the i^{th} lexicographically smallest suffix of T. The inverse suffix array is defined as the length n array such that i^{th} element is the position of $T[i, n]$ in the suffix array, i.e., the lexicographic rank of $T[i, n]$.

The Lyndon factorization (defined in Section 1) of a string can be computed in linear time. This can be done using the well known Duval's algorithm [11], or by using the inverse suffix array, which can be constructed in linear time [22]. Lemma 8 makes it clear why the latter technique works.

▶ **Lemma 8** (Theorem 2.2 [29]). *The starting index, i, of a suffix in T that is lexicographically smaller than any suffix starting at index $j < i$ is an index where a Lyndon factor begins.*

In other words, as we scan the inverse suffix array from left-to-right, an index i where the inverse suffix array value is smaller than any seen thus far marks the start of a Lyndon factor. Moreover, if a Lyndon factor starts at index i in T, the next Lyndon word must be this factor. We aim to use this to construct strings where the number of Lyndon factors tells us something about the number of constraints satisfied within an ordering constraint satisfaction problem (OCSP). The definition of an OCSP used here is less general than the one given in [14], but still sufficient for our purposes.

▶ **Definition 9.** *An OCSP of arity k is specified by a set $\Lambda \subseteq S_k$ where S_k is the set of permutations of $\{1, 2, ..., k\}$. An instance of such an OCSP consists of a set of variables, $V = \{x_1, \ldots, x_n\}$, and m constraints, C_1, $\ldots$, C_m, each of which is an ordered k-tuple of V. The objective is to find a global ordering σ of V that maximizes $\sum_{i=1}^{m} \chi_\Lambda(\sigma_{|C_i})$, where $\sigma_{|C_i} \in S_k$ is the ordering of the k elements of C_i induced by the global ordering σ, and $\chi_\Lambda(\sigma_{|C_i}) = 1$ if $\sigma_{|C_i} \in \Lambda$ and 0 otherwise. If $\chi_\Lambda(\sigma_{|C_i}) = 1$, we say that C_i is satisfied.*

Note that $m \leq n!/(n-k)! \leq n^k$. Additionally, we will only consider OCSP instances where each variable appears in at least two constraints. Under this last assumption, we can relate the number of variables, n, to the number of clauses, m.

▶ **Lemma 10.** *For OCSPs with arity k constraints, n variables, and m constraints, where every variable appears in at least two clauses, $n \leq \frac{k}{2}m$.*

Proof. Since every variable appears in at least two constraints,

$$2n \leq \sum_{i=1}^{n} (\text{the number of times variable } x_i \text{ appears in total}) = km. \qquad ◀$$

One of the simplest OCSPs is the *Maximum Acyclic Subgraph Problem* (MAS), where $k = 2$, making constraints of the form (x_i, x_j), and where $\Lambda = \{\left(\begin{smallmatrix}1&2\\1&2\end{smallmatrix}\right)\}$ (using two-line permutation notation). That is, Λ contains only the identity permutation that orders $x_i < x_j$. For example, an instance of MAS could be $V = \{x_1, x_2, x_3, x_4, x_5\}$ and $C_1 = (x_1, x_3)$, $C_2 = (x_5, x_2)$, $C_3 = (x_3, x_4)$, $C_4 = (x_2, x_1)$. An ordering σ that puts the variables in the order $x_4 < x_5 < x_3 < x_2 < x_1$ would yield $\chi_\Lambda(\sigma_{|C_1}) = \chi_\Lambda\left(\left(\begin{smallmatrix}1&2\\2&1\end{smallmatrix}\right)\right) = 0$, $\chi_\Lambda(\sigma_{|C_2}) = \chi_\Lambda\left(\left(\begin{smallmatrix}1&2\\1&2\end{smallmatrix}\right)\right) = 1$, $\chi_\Lambda(\sigma_{|C_3}) = \chi_\Lambda\left(\left(\begin{smallmatrix}1&2\\2&1\end{smallmatrix}\right)\right) = 0$, $\chi_\Lambda(\sigma_{|C_4}) = \chi_\Lambda\left(\left(\begin{smallmatrix}1&2\\1&2\end{smallmatrix}\right)\right) = 1$, making its objective value 2.

The dual minimization problem of MAS is known as *Feedback Arc Set* (FAS). In this problem, the aim is to minimize the objective value of a solution, which is defined as the number of constraints being violated, i.e., $m - \sum_{i=1}^{m} \chi_\Lambda(\sigma_{|C_i})$. The problem is otherwise identical. The following hardness result for FAS is used when proving Theorem 5.

▶ **Lemma 11** ([14]). *Conditioned on the Unique Games Conjecture, for every constant $C > 1$, it is NP-hard to find a C-approximation for FAS.*

The Unique Games Conjecture is described in [21]. We will use the term Unique-Games-hard to refer to problems that, conditioned on the Unique Games conjecture, are NP-hard.

We can always assume that at least half of the constraints in an instance of MAS can be satisfied. To see this, take an arbitrary ordering of the variables. Either this ordering or its reversal must satisfy at least $m/2$ constraints. This is just a specific instance of a more general result. We can always assume our optimal solution satisfies at least $|\Lambda|m/k!$ constraints. Since the expected number of constraints satisfied by a random ordering on the variables is $|\Lambda|m/k!$, we know the maximum number of constraints satisfied by any ordering is bounded below by this quantity. It turns out, however, that finding a solution that does better than this expected value is computationally difficult. We give a simplified statement of the main result in [14], maintaining only the pertinent details for our problem.

▶ **Theorem 12** ([14]). *For an OCSP with arity k, for every constant $\varepsilon > 0$, it is Unique-Games-hard to find an ordering for the variables that achieves a ratio of satisfied constraints over total constraints that is at least $|\Lambda|/k! + \varepsilon$.*

Our results also make use of the OCSP known as the *Betweenness Problem.* In this problem $k = 3$ and $\Lambda = \{\left(\begin{smallmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{smallmatrix}\right)\}$. For a constraint (x_i, x_j, x_k) to be satisfied either $x_i < x_j < x_k$ or $x_k < x_j < x_i$. For example, the ordering $x_4 < x_5 < x_3 < x_2 < x_1$ satisfies the constraint (x_1, x_2, x_5), but not the constraint (x_4, x_2, x_5). By applying Theorem 12 to the Betweenness problem, we obtain that it is Unique-Games-hard to achieve a ratio of satisfied constraints to total constraints better than $2/3! = 1/3$.

For hardness under the Exponential Time Hypothesis (ETH) [19], we will use a result by Kim and Gonçalves appearing in [23]. An Arity k Permutation CSP as defined in [23] is a OCSP where Λ consists of the identity permutations, $\Lambda = \{\left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 & 2 \\ 1 & 2 \end{smallmatrix}\right), \ldots, \left(\begin{smallmatrix} 1 & 2 & \cdots & k \\ 1 & 2 & \cdots & k \end{smallmatrix}\right)\}$, and constraints *up to* arity k are allowed. This is different from our definition of OCSPs, where all constraints are of exactly arity k. The differences between these two definitions are accommodated for whenever Lemma 13 is used. In [23] the authors prove the following.

▶ **Lemma 13** ([23]). *Assuming ETH, there is no $2^{o(n \log n)}$-algorithm for Arity 4 Permutation CSP (and thus for Arity k Permutation CSP, $k \geq 4$).*

3 Hardness of Lyndon Factor Minimization

The first reduction is from the Betweenness problem to the Lyndon Factor Minimization Problem. It is used to demonstrate NP-completeness. An alternative proof can be done with a reduction from MAS. Our reasoning for choosing one over the other is we believe that the Betweenness problem provides a good initial illustration of the power of a hypothetical solver to these Lyndon factorization problems. It also provides a warm-up for the techniques used in Section 3.2. Moreover, we will use a reduction from MAS as a short proof to illustrate NP-completeness for the maximization problem, before introducing a more involved reduction to prove an inapproximability result.

3.1 NP-Completeness of Lyndon Factor Minimization

We are given as input an instance ϕ of the Betweenness problem consisting of n variables x_1, x_2, …, x_n and m constraints C_1, C_2, …, C_m. Let $F(T)$ denote the number of Lyndon factors of a string T under the alphabet ordering currently under consideration. We will use $F_T(T_1)$ to denote the number of Lyndon factors of T starting within the *first occurrence* of the substring T_1 of T. The subscript T is to remind us that the factors starting in T_1 are sensitive to the other symbols in T. By a *run* of a symbol, we mean a maximal unary substring containing that symbol.

▶ **Lemma 14.** *Let T be any string of the form $T = T_1 \circ (x_0)^\alpha \circ (x_1^\gamma\ x_2^\gamma\ \dots\ x_n^\gamma)^\beta$ where T_1 is over the alphabet $\{x_0, \dots, x_n\}$, α is greater than the length of any run of x_0 in T_1, γ is greater than the length of any run of any symbol other than x_0 in T_1, and $\beta > 1$. If x_0 is the smallest symbol in the ordering, then $F(T) \leq F_T(T_1) + 1$.*

Proof. If T_1 does not end with an x_0, then the first x_0 in the $(x_0)^\alpha$ marks the start of a new Lyndon factor in T since $(x_0)^\alpha$ is lexicographically smaller than any preceding suffix. Then this factor includes the remaining suffix of T. In this case $F(T) = F_T(T_1) + 1$. If T_1 contains a suffix consisting of only x_0's, then a new Lyndon factor must start at the first of these x_0's, and again this factor contains the remaining suffix of T. In this case, $F(T) = F_T(T_1)$. ◀

▶ **Lemma 15.** *Let T be defined as in Lemma 14. If x_0 is not the smallest symbol in the ordering, $F(T) \geq \beta - 1$.*

Proof. In this case, the smallest symbol must be one of $x_1, \dots, x_n$. Suppose the smallest is x_i. Then the first symbol in the first x_i^γ marks the beginning of a Lyndon factor. This factor is of the form $x_i^\gamma\ x_{i+1}^\gamma\ \dots\ x_n^\gamma\ x_1^\gamma\ \dots x_{i-1}^\gamma$ and is repeated at least $\beta - 1$ times. In particular, the suffix $x_{i+1}^\gamma\ \dots\ x_n^\gamma$ is preceded by $\beta - 1$ factors of the form $x_i^\gamma\ x_{i+1}^\gamma\ \dots\ x_n^\gamma\ x_1^\gamma\ \dots\ x_{i-1}^\gamma$. ◀

Lemmas 14 and 15 will be useful in proving that x_0 must be smallest in an optimal ordering. We now introduce our constraint gadgets.

▶ **Lemma 16.** *Let x_0 be the smallest symbol in T. For $i, j, k > 0$, consider the first instance of a substring S of T where*

$$S = x_0^\eta\ x_j\ x_0^\eta\ x_i\ x_0^\eta\ x_j\ x_0^\eta\ x_i\ x_0^\eta\ x_k\ x_0^\eta\ x_i\ x_0^\eta\ x_i\ x_0^\eta\ x_j\ x_0^\eta\ x_j\ x_0^\eta\ x_j$$

and η is larger than the length of any run of x_0 preceding S in T, and S is immediately followed by the run $x_0^{\eta+1}$. The symbols in this first instance of S make up three complete Lyndon factors if x_j is ordered between x_i and x_k, and four complete Lyndon factors otherwise.

Proof. Since the number of times x_0 is repeated is more than the length of any previous run, it must be the case that a new factor begins at the start of S. The six possible cases and their corresponding factorizations are:

$$\begin{aligned}
&x_0 < x_i < x_j < x_k : (x_0^\eta\ x_j), (x_0^\eta\ x_i\ x_0^\eta\ x_j\ x_0^\eta\ x_i\ x_0^\eta\ x_k), (x_0^\eta\ x_i\ x_0^\eta\ x_i\ x_0^\eta\ x_j\ x_0^\eta\ x_j\ x_0^\eta\ x_j)\\
&x_0 < x_i < x_k < x_j : (x_0^\eta\ x_j), (x_0^\eta\ x_i\ x_0^\eta\ x_j), (x_0^\eta\ x_i\ x_0^\eta\ x_k), (x_0^\eta\ x_i\ x_0^\eta\ x_i\ x_0^\eta\ x_j\ x_0^\eta\ x_j\ x_0^\eta\ x_j)\\
&x_0 < x_j < x_i < x_k : (x_0^\eta\ x_j\ x_0^\eta\ x_i\ x_0^\eta\ x_j\ x_0^\eta\ x_i\ x_0^\eta\ x_k\ x_0^\eta\ x_i\ x_0^\eta\ x_i), (x_0^\eta\ x_j), (x_0^\eta\ x_j), (x_0^\eta\ x_j)\\
&x_0 < x_k < x_i < x_j : (x_0^\eta\ x_j), (x_0^\eta\ x_i\ x_0^\eta\ x_j), (x_0^\eta\ x_i), (x_0^\eta\ x_k\ x_0^\eta\ x_i\ x_0^\eta\ x_i\ x_0^\eta\ x_j\ x_0^\eta\ x_j\ x_0^\eta\ x_j)\\
&x_0 < x_j < x_k < x_i : (x_0^\eta\ x_j\ x_0^\eta\ x_i\ x_0^\eta\ x_j\ x_0^\eta\ x_i\ x_0^\eta\ x_k\ x_0^\eta\ x_i\ x_0^\eta\ x_i), (x_0^\eta\ x_j), (x_0^\eta\ x_j), (x_0^\eta\ x_j)\\
&x_0 < x_k < x_j < x_i : (x_0^\eta\ x_j\ x_0^\eta\ x_i), (x_0^\eta\ x_j\ x_0^\eta\ x_i), (x_0^\eta\ x_k\ x_0^\eta\ x_i\ x_0^\eta\ x_i\ x_0^\eta\ x_j\ x_0^\eta\ x_j\ x_0^\eta\ x_j)
\end{aligned}$$

Notice that only in the first and last orderings where the constraint is satisfied are there three factors. The other cases have four. ◀

For each constraint $C_t = (x_i, x_j, x_k)$ in the instance ϕ of the Betweenness problem, where $1 \leq t \leq m$, we construct the gadget from Lemma 16,

$$S(C_t) \coloneqq x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_j\ x_0^t\ x_j.$$

We next define $S(\phi) \coloneqq S(C_1) \circ S(C_2) \circ \ldots \circ S(C_m) \circ (x_0)^{m+1} \circ (x_1^2\ x_2^2\ \ldots\ x_n^2)^\beta$ where $\beta = 3m + 3$.

▶ **Lemma 17.** *The string $S(\phi)$ has an alphabet ordering yielding at most $3m + 1$ Lyndon factors iff there exists a variable ordering satisfying all constraints in ϕ.*

Proof. Assuming there exists a constraint satisfying variable ordering for ϕ, make x_0 the smallest symbol and order the remaining symbols $x_1, \ldots, x_n$ according to the variable ordering. By Lemma 16, each of the substrings $S(C_t)$ for $1 \leq t \leq m$ contributes three factors, and by the analysis in Lemma 14 the remaining suffix contributes one additional factor. This creates $3m + 1$ factors in total.

Conversely, assume that no variable ordering exists that satisfies the constraints. If x_0 is the smallest symbol, then at least one $S(C_t)$ gadget contributes four factors while the others contribute at least three. The remaining suffix contributes one factor making the number of factors at least $4 + 3(m - 1) + 1 = 3m + 2$. If x_0 is not the smallest symbol, then by Lemma 15, the number of factors is at least $\beta - 1 = (3m + 3) - 1 = 3m + 2$. ◀

Since determining if there exists a variable ordering satisfying all constraints in an instance of the Betweenness problem is NP-hard [32], determining whether there exists an alphabet order where there are at most $3m + 1$ Lyndon factors is NP-hard as well. With a symbol ordering as a polynomial sized certificate, the problem is clearly in NP, proving Theorem 3.

3.2 ETH Hardness of Lyndon Factor Minimization

Here we reduce Arity 4 Permutation CSP to Lyndon Factor Minimization. Assume for the moment that x_0 is the smallest symbol, and that each substring $S(C_t)$ (yet to be defined) is followed by a run of x_0 longer than any run of x_0 that precedes it.

For an arity 2 constraint $C_t = (x_i, x_j)$, we construct a string using the symbols x_0, x_i, and x_j that has either 3 or 4 factors depending on the ordering on the variables. We will demonstrate which orderings create which factorizations. The string we construct is $S(C_t) = x_0^t\ x_i\ x_0^t\ x_i\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_i$, which has the factorizations for different orderings,

Ordering	Factorization	# factors
$x_i < x_j$:	$(x_0^t\ x_i\ x_0^t\ x_i\ x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i)(x_0^t\ x_i)$	3
$x_j < x_i$:	$(x_0^t\ x_i)(x_0^t\ x_i)(x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_i)$	4

Slightly more involved are the strings to model arity 3 constraints $C_t = (x_i, x_j, x_k)$, $S(C_t) = x_0^t\ x_i\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_i.$, where

Ordering	Factorization	# factors
$x_i < x_j < x_k$:	$(x_0^t\ x_i\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i)(x_0^t\ x_i)$	3
$x_i < x_k < x_j$:	$(x_0^t\ x_i\ x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i)(x_0^t\ x_i)$	4
$x_j < x_i < x_k$:	$(x_0^t\ x_i)(x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_i)$	4
$x_k < x_i < x_j$:	$(x_0^t\ x_i\ x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i)(x_0^t\ x_i)(x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_i)$	4
$x_j < x_k < x_i$:	$(x_0^t\ x_i)(x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_i)$	4
$x_k < x_j < x_i$:	$(x_0^t\ x_i)(x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_i)(x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_i)$	4

The most involved is the gadget for an arity 4 constraint $C_t = (x_i, x_j, x_k, x_h)$,
$S(C_t) = x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_h\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i$
which has the following factorizations depending on the ordering given to its symbols,

Ordering ('<' omitted)	Factorization	#
x_i, x_j, x_k, x_h :	$(x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_h\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k)(x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i)$	3
x_i, x_j, x_h, x_k :	$(x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k)(x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_h\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k)(x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i)$	4
x_i, x_k, x_j, x_h :	$(x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_h\ x_0^t\ x_j)(x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i)$	4
x_i, x_h, x_j, x_k :	$(x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k)(x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i\ x_0^t\ x_h\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i)$	4
x_i, x_k, x_h, x_j :	$(x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_h\ x_0^t\ x_j)(x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i)$	4
x_i, x_h, x_k, x_j :	$(x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i\ x_0^t\ x_h\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i)$	4
x_j, x_i, x_k, x_h :	$(x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_h)(x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i)$	4
x_j, x_i, x_h, x_k :	$(x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_h\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i)$	4
x_k, x_i, x_j, x_h :	$(x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i)(x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_h\ x_0^t\ x_j\ x_0^t\ x_i)(x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i)$	4
x_h, x_i, x_j, x_k :	$(x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k)(x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i)(x_0^t\ x_h\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i)$	4
x_k, x_i, x_h, x_j :	$(x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i)(x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_h\ x_0^t\ x_j\ x_0^t\ x_i)(x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i)$	4
x_h, x_i, x_k, x_j :	$(x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i)(x_0^t\ x_h\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i)$	4
x_j, x_k, x_i, x_h :	$(x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_h)(x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i)$	4
x_j, x_h, x_i, x_k :	$(x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_h\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i)$	4
x_k, x_j, x_i, x_h :	$(x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i)(x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_h\ x_0^t\ x_j\ x_0^t\ x_i)(x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i)$	4
x_h, x_j, x_i, x_k :	$(x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i)(x_0^t\ x_h\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i)$	4
x_k, x_h, x_i, x_j :	$(x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i)(x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_h\ x_0^t\ x_j\ x_0^t\ x_i)(x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i)$	4
x_h, x_k, x_i, x_j :	$(x_0^t\ x_i\ x_0^t\ x_j)(x_0^t\ x_i)(x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i)(x_0^t\ x_h\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i)$	4
x_j, x_k, x_h, x_i :	$(x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_h)(x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i)$	4
x_j, x_h, x_k, x_i :	$(x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_h\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i)$	4
x_k, x_j, x_h, x_i :	$(x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i)(x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_h\ x_0^t\ x_j\ x_0^t\ x_i)(x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i)$	4
x_h, x_j, x_k, x_i :	$(x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i)(x_0^t\ x_h\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i)$	4
x_k, x_h, x_j, x_i :	$(x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i)(x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_h\ x_0^t\ x_j\ x_0^t\ x_i)(x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i)$	4
x_h, x_k, x_j, x_i :	$(x_0^t\ x_i)(x_0^t\ x_j\ x_0^t\ x_i)(x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i)(x_0^t\ x_h\ x_0^t\ x_j\ x_0^t\ x_i\ x_0^t\ x_k\ x_0^t\ x_i\ x_0^t\ x_j\ x_0^t\ x_i)$	4

The string construction for the overall reduction is almost identical to the one for ϕ in Section 3.1. We only need to select β to be slightly different. We let $\beta = 4m + 3$. This is enough to ensure that in an optimal solution x_0 must be the smallest symbol. If x_0 is smallest, in the worst-case, when all constraints are not satisfied, there are at most $4m + 1$ Lyndon factors. If x_0 is not smallest, as shown in Lemma 15, the number of factors is at least $\beta - 1 = 4m + 2$. Then, with x_0 as the minimum, each ordering on $x_1, \ldots, x_n$ gives us $3s + 4(m - s) + 1 = 4m + 1 - s$ factors, where s is the number of satisfied constraints when using the corresponding variable ordering in ϕ. Therefore, an optimal ordering for the n variables of ϕ is obtained by an order on the $(n + 1)$ symbols which minimizes the number of Lyndon factors in the string. This combined with Lemma 13 proves Theorem 4.

3.3 Inapproximability of Lyndon Factor Minimization

We will perform an approximation preserving reduction from FAS to Lyndon Factor Minimization. Recall that for FAS the arity k of the constraints is 2, so that constraints are of the form (x_i, x_j) and Λ consists of the identity permutation. In other words, the constraint is only satisfied if $x_i < x_j$. The cost of the solution will be the number of violated constraints, which we wish to minimize. Our gadget for constraint $C_t = (x_i, x_j)$ will be

$$S(C_t) = (x_0^t\ x_i) \circ (x_0^t\ x_j)^{\alpha-1}$$

where $\alpha > 1$ will be chosen later. The whole string for our reduction will be

$$T = S(\phi) = S(C_1) \circ S(C_2) \circ \ldots \circ S(C_m) \circ (x_0)^{m+1} \circ (x_1^2\ x_2^2\ \ldots\ x_n^2)^{\beta}$$

where $\beta = \alpha m + 3$. By Lemma 15, if x_0 is not smallest, then $F(T) \geq \beta - 1$. We consider next what happens in our constraint gadgets when x_0 is smallest.

▶ **Lemma 18.** *If x_0 is smallest and $x_i < x_j$ then $F_T(S(C_t)) = 1$.*

Proof. Since x_0^t is the longest run of x_0 seen so far, the start of $S(C_t)$ marks the smallest suffix seen so far when traversing T from left to right. Then, since $x_j > x_i$, the start of all substrings of the form $x_0^t\ x_j$ do not mark the start of the smallest suffix seen so far. ◀

▶ **Lemma 19.** *If x_0 is smallest and $x_j < x_i$ then $F_T(S(C_t)) = \alpha$.*

Proof. Again, since x_0^t is the longest run of x_0 seen so far, the start of $S(C_t)$ marks the smallest suffix seen so far when traversing T from left to right. However, now the start of each substring of the form $x_0^t\ x_j$ marks the start of the smallest suffix seen so far (recall after the last $x_0^t\ x_j$ there will be a longer run of x_0 than has been seen before). Hence, there are $\alpha - 1$ additional factors created. ◀

▶ **Lemma 20.** *Any alphabet ordering where x_0 is smallest has fewer factors than an alphabet ordering where x_0 is not the smallest.*

Proof. If x_0 is smallest, $F(T) = s + \alpha(m-s) + 1$ where s is the number of satisfied constraints and the $+1$ arises from the last factor, $(x_0)^{m+1} \circ (x_1^2\ x_2^2\ \ldots\ x_n^2)^{\beta}$. Because $\alpha > 1$, this is upper bounded by the case when $s = 0$ so that $F(T) \leq \alpha m + 1$. On the other hand, if x_0 is not smallest $F(T) \geq \beta - 1 = \alpha m + 2$. ◀

Henceforth, we only need to worry about the case when x_0 is the smallest. Our aim is to show that a constant approximation algorithm for Lyndon Factor Minimization allows us to construct a constant approximation algorithm for FAS. If our hypothetical approximation algorithm for Lyndon Factor Minimization ever returned a solution where x_0 is not smallest, we add the additional step of replacing that solution with any solution where x_0 is smallest, obtaining a solution that performs even better. Then our modified algorithm maintains being an approximation algorithm for Lyndon Factor Minimization (perhaps with an even smaller approximation factor).

Let s_F^* denote the number of constraints satisfied in an optimal solution of ϕ for FAS and let s_L^* denote the number of constraints in ϕ satisfied by the variable ordering obtained from our optimal, factor minimizing, alphabet order for the corresponding instance of Lyndon Factor Minimization. Also, let s denote the actual number of constraints satisfied by the variable ordering obtained from our approximate factor minimizing alphabet order for the corresponding instance of Lyndon Factor Minimization. A λ-approximation for Lyndon Factor Minimization with $\lambda > 1$ gives the following set of inequalities:

$$s_L^* + \alpha(m - s_L^*) + 1 \leq s + \alpha(m - s) + 1 \leq \lambda(s_L^* + \alpha(m - s_L^*) + 1).$$

Which can be equivalently written as

$$(m - s_L^*) + \frac{s_L^* + 1}{\alpha} \leq (m - s) + \frac{s + 1}{\alpha} \leq \lambda(m - s_L^*) + \lambda\frac{s_L^* + 1}{\alpha}. \tag{1}$$

We will show that by taking α large enough we can ensure $s_L^* = s_F^*$.

▶ **Lemma 21.** *With $\alpha = 2(m+1)+1$, we have that $s_L^* = s_F^*$.*

Proof. The cost of an optimal solution of ϕ is $m - s_F^*$. The solution for ϕ we get from mapping our solution for Lyndon factorization back to ϕ must have at least as many violated constraints as the optimal solution for ϕ, i.e., $m - s_L^* \geq m - s_F^*$, and so $s_F^* \geq s_L^*$. Let us suppose for the sake of contradiction that $s_F^* \geq s_L^* + 1$. This implies $m - s_L^* - (m - s_F^*) \geq 1$. Then, using in addition that $\frac{s_F^*+1}{\alpha} \leq \frac{m+1}{\alpha} \leq \frac{1}{2}$, we obtain

$$\frac{s_F^* + 1}{\alpha} - \frac{s_L^* + 1}{\alpha} \leq \frac{1}{2} < 1 \leq m - s_L^* - (m - s_F^*),$$

which implies that

$$m - s_F^* + \frac{s_F^* + 1}{\alpha} < m - s_L^* + \frac{s_L^* + 1}{\alpha}.$$

Or, written more naturally as the cost of a Lyndon Factor Minimization Problem's solution,

$$s_F^* + \alpha(m - s_F^*) + 1 < s_L^* + \alpha(m - s_L^*) + 1.$$

But then this implies that the ordering on $x_1, \ldots, x_n$ that is used to obtain the optimal solution for ϕ creates fewer Lyndon factors than our supposedly optimal solution for Lyndon Factor Minimization, a contradiction. ◀

Let us now upper bound $m - s$ (our approximate solution cost when the solution is mapped back to FAS) in terms of $\lambda(m - s_F^*)$. Combining the inequalities in (1) with Lemma 21, and the fact that $s_F^* = s_L^* \leq m$ when $\alpha = 2(m+1)+1$, we get that

$$m - s \leq m - s + \frac{s+1}{\alpha} \leq \lambda(m - s_L^*) + \lambda\frac{s_L^* + 1}{\alpha} \leq \lambda\left(m - s_F^* + \frac{1}{2}\right).$$

The case where $m = s_F^*$ can easily be solved in polynomial time, so we can consider that check added to our hypothetical solution as well. Hence, we assume $m - s_F^* \geq 1 > 1/2$ and,

$$m - s_F^* \leq m - s \leq \lambda\left(m - s_F^* + \frac{1}{2}\right) < \lambda(m - s_F^* + m - s_F^*) = 2\lambda(m - s_F^*).$$

We have shown that a λ approximation for Lyndon Factor Minimization allows us to obtain, at worst, a 2λ approximation for FAS. Moreover, the α value we need to do this is polynomial in m so that the whole reduction is done in polynomial time. This polynomial time constant approximation algorithm is better then what is allowed by Lemma 11 under the Unique Games Conjecture. This completes the proof of Theorem 5.

4 Hardness of Lyndon Factor Maximization

Our approach will be similar to the one taken for minimization. First, we introduce some gadgetry for the NP-completeness proof that is later expanded upon to create an inapproximability result. As of now, the authors have not yet found gadgets to establish the same ETH hardness for the maximization problem.

4.1 NP-Completeness of Lyndon Factor Maximization

We perform a reduction from the dual of FAS, the Maximum Acyclic Subgraph Problem (MAS). Recall MAS is identical to FAS except for the cost of a solution now being the number of constraints satisfied, which we wish to maximize. For constraint $C_t = (x_i, x_j)$, we define our constraint gadget as $S(C_t) = x_0^{t+1}\ x_j\ x_0^{t+1}\ x_i$ (note the reversal of i and j). The entire string formed by our instance ϕ of FAS is

$$T = S(\phi) = (x_0\ x_1\ x_2\ \dots\ x_n) \circ S(C_1) \circ S(C_2) \circ \dots \circ S(C_m) \circ (x_0)^m.$$

▶ **Lemma 22.** *If x_0 is not the smallest symbol in the ordering, then $F(T) \leq n + m$.*

Proof. Suppose $x_i \neq x_0$ is the smallest symbol. Then the first Lyndon factor starting with x_i occurs in the prefix $(x_0\ x_1\ \dots\ x_n)$. Subsequent Lyndon factors must begin with x_i. The prefix contributes at most n factors and there are at most m remaining occurrences of x_i. ◀

▶ **Lemma 23.** *In an ordering where x_0 is smallest, $F(T) = 2s + (m - s) + 1 + m$, where s is the number of constraints satisfied in MAS by the ordering given to x_1, …, x_n.*

Proof. For a substring $S(C_t)$, if $C_t = (x_i, x_j)$ is not satisfied (i.e., $x_i > x_j$) then $F_T(S(C_t)) = 1$. If it is satisfied (i.e., $x_i < x_j$) then $F_T(S(C_t)) = 2$. The prefix $x_0\ x_1\ x_2\ \dots\ x_n$ contributes exactly one additional factor. The suffix $(x_0)^m$ contributes m factors. ◀

▶ **Lemma 24.** *Any ordering where x_0 is the smallest has more factors than an ordering where x_0 is not the smallest.*

Proof. By Lemma 10, we can assume that $n \leq m$. Then by Lemma 22, we have that if x_0 is not smallest, $F(T) \leq n + m \leq 2m$. By Lemma 23, if x_0 is smallest then $F(T) = 2s + (m - s) + 1 + m = s + 2m + 1 > 2m$. ◀

The value $F(T)$ is maximized by an alphabet order which has the largest possible number of satisfied constraints, say s^*. This gives $(s^* + 2m + 1)$ Lyndon factors. Clearly, this solution also provides an ordering satisfying the maximum number of constraints in our MAS instance. Since MAS is NP-hard, we have shown Lyndon Factor Maximization is NP-hard as well. The decision problem is in NP using the ordering on $x_1\ \dots\ x_n$ as a polynomial sized certificate, and this remains NP-hard as it could be used to solve the optimization problem. This completes the proof of Theorem 6.

4.2 Inapproximability of Lyndon Factor Maximization

First, let us describe the OCSP from which we are reducing. Let $k > 1$ be the arity of the constraints, which we will specify later. Each constraint will be satisfied iff the variables in that constraint have one of the $(k-1)!$ orderings where the last variable is ordered first, i.e., for constraint $(x_{i_1}, x_{i_2}, \dots, x_{i_{k-1}}, x_{i_k})$, the ordering over those variables will have $x_{i_k} < x_{i_j}$ for $j \in [1, k-1]$. More formally, $\Lambda = \{\left(\begin{smallmatrix} 1 & 2 & \dots & k-1 & k \\ z_1 & z_2 & \dots & z_{k-1} & 1 \end{smallmatrix}\right) \mid \cup_{i=1}^{k-1}\{z_i\} = \{2, \dots, k\}\}$. According to Theorem 12, it is Unique-Games-Hard to find an approximation which beats $|\Lambda| m/k! = (k-1)! m/k! = m/k$ constraints being satisfied.

Our constraint gadget is of the form

$$S(C_t) = (x_0^{t+1}\ x_{i_1}) \circ (x_0^{t+1}\ x_{i_2}) \circ \dots \circ (x_0^{t+1}\ x_{i_{k-1}}) \circ (x_0^{t+1}\ x_{i_k})^{\alpha}$$

and our overall string constructed from our instance ϕ of OCSP is

$$T := S(\phi) = (x_0\ x_1\ x_2\ \dots\ x_n) \circ S(C_1) \circ S(C_2) \circ \dots \circ S(C_m) \circ (x_0), \quad \text{where } \alpha = mn.$$

▶ **Lemma 25.** *If x_0 is not smallest then $F(T) \leq n+m$.*

Proof. Let $x_i \neq x_0$ be the smallest symbol instead. Then the prefix $(x_0\ x_1\ x_2\ \dots\ x_n)$ contributes at most n factors, and each remaining factor must begin with x_i. We will show that there is at most 1 factor starting in each constraint gadget. For a given constraint containing x_i, if $x_i \neq x_{i_k}$ this is immediate. On the other hand, if $x_i = x_{i_k}$ then only its first occurrence can form a smaller suffix of T than those preceding it. In more detail, since $x_0 > x_i = x_{i_k}$, we have $x_{i_k}\ (x_0^t\ x_{i_k})^{\alpha-1}x_0 < x_{i_k}\ (x_0^t\ x_{i_k})^{\alpha-2}x_0 < x_{i_k}\ (x_0^t\ x_{i_k})^{\alpha-3}x_0 < \dots$. Note that this is the reason for the final x_0 appended to T. ◀

▶ **Lemma 26.** *If x_0 is smallest, and in constraint $C_t = (x_{i_1}, \dots, x_{i_k})$ the symbol x_{i_k} is smallest among $x_{i_1}\ \dots\ x_{i_k}$, then $F_T(S(C_t)) \geq \alpha$.*

Proof. Since $x_0^{t+1}x_{i_k} < x_0^{t+1}x_{i_j}$ for $j \in [1, k-1]$, and the substring following $S(C_t)$ is either x_0^{t+2} (or the final x_0 of T), the start of **each run** of x_0 in the substring $(x_0^{t+1}x_{i_k})^\alpha$ marks the start of a suffix smaller than any of those preceding it. ◀

▶ **Lemma 27.** *If x_0 is the smallest in the ordering, then $F(T) \geq \alpha s + 1$ where s is the number of clauses in ϕ satisfied by the ordering given to $x_1, \dots, x_n$. This is larger than the number of factors from any ordering where x_0 is not the smallest.*

Proof. By Lemma 26, when x_0 is the smallest each of the satisfied constraint gadgets contributes at least α factors. In addition, the lone x_0 symbol at the end of T forms its own factor. For the second statement, we can always assume our approximate solution satisfies at least 1 constraint, hence $s \geq 1$ and $\alpha s + 1 \geq mn + 1 > m + n$, which by Lemma 25 is an upper bound on the number of factors when x_0 is not smallest. ◀

From here we only need to consider when x_0 is smallest, for the same reasoning as given in Section 3.3. Now, suppose we have a λ-approximation with $\lambda < 1$ for Lyndon Factor Maximization. Let s_L^* be the number of constraint gadgets satisfied from our optimal solution of Lyndon factor maximization, and s the number from the approximate solution. Then,

$$\lambda(\alpha s_L^* + 1 + y_L^*) \leq \alpha s + 1 + y \leq \alpha s_L^* + 1 + y_L^*$$

where y_L^* represents the number of additional factors contributed beyond $\alpha s_L^* + 1$ and y represents the number of factors beyond $\alpha s + 1$ for our approximate solution. We can equivalently write the above expression as

$$\lambda s_L^* \left(1 + \frac{1}{\alpha s_L^*} + \frac{y_L^*}{\alpha s_L^*}\right) \leq s\left(1 + \frac{1}{\alpha s} + \frac{y}{\alpha s}\right) \leq s_L^* \left(1 + \frac{1}{\alpha s_L^*} + \frac{y_L^*}{\alpha s_L^*}\right). \tag{2}$$

▶ **Lemma 28.** *For all $s \in [1, m]$, and for the corresponding y value as described above,*

$$1 \leq \left(1 + \frac{1}{\alpha s} + \frac{y}{\alpha s}\right) \leq 3.$$

Proof. We first bound y from above. Any factor in a constraint gadget begins at the start of a run x_0. In a satisfied constraint gadget, there are $k-1$ such runs outside of the $(x_0^{t+1}x_{i_k})^\alpha$ substring. Hence, each satisfied constraint gadget contributes at most $k-1$ additional factors beyond α. A constraint gadget that is not satisfied, i.e., has $x_{i_j} < x_{i_k}$ for some $j \neq k$, has the gadget's last factor beginning at the start of the substring $(x_0^{t+1}\ x_{i_j})$. This implies the substring $(x_0^{t+1}\ x_{i_k})^\alpha$ does not split into different factors. Therefore, an unsatisfied constraint gadget again contributes at most $k-1$ factors. Because of this, the m constraint gadgets contribute at most $k-1$ additional factors in total and $y \leq m(k-1)$. Finally, $\alpha = mn$, hence

$$\frac{y}{\alpha s} \leq \frac{y}{\alpha} \leq \frac{m(k-1)}{\alpha} \leq \frac{mn}{\alpha} = 1 \quad \text{and} \quad \frac{1}{\alpha s} \leq \frac{1}{\alpha} = \frac{1}{nm} \leq 1. \qquad ◀$$

Let s^*_C be the number of constraints satisfied in an optimal solution to ϕ. Like in Section 3.3, we know that $s \leq s^*_C$ and $s^*_L \leq s^*_C$, Using Lemma 28 we can easily make them differ by at most a constant factor.

▶ **Lemma 29.** *Using the definitions above, it holds that* $s^*_C \leq 3s^*_L$.

Proof. For the sake of contradiction, assume instead that $s^*_C > 3s^*_L$. Applying the ordering given by the optimal solution of ϕ to the symbols $x_1, \ldots, x_n$, and letting y^*_C be defined as above but for s^*_C, we have

$$s^*_C\left(1 + \frac{1}{\alpha s^*_C} + \frac{y^*_C}{\alpha s^*_C}\right) > s^*_C > 3s^*_L \geq s^*_L\left(1 + \frac{1}{\alpha s^*_L} + \frac{y^*_L}{\alpha s^*_L}\right).$$

However, this implies $\alpha s^*_C + 1 + y^*_C > \alpha s^*_L + 1 + y^*_L$. Thus, s^*_L couldn't have been the number of constraints satisfied in an optimal solution to our Lyndon Factor Maximization instance, since using whichever ordering was used for the solution to ϕ would have given us more factors, a contradiction. ◀

By Lemma 29, we have $\frac{1}{3}s^*_C \leq s^*_L$. Multiplying both sides by $\lambda/3$, we obtain $\frac{\lambda}{9}s^*_C \leq \frac{\lambda}{3}s^*_L$. By Lemma 28 and our starting inequality in (2) we also have that

$$\lambda s^*_L \leq \lambda s^*_L\left(1 + \frac{1}{\alpha s^*_L} + \frac{y^*_L}{\alpha s^*_L}\right) \leq s\left(1 + \frac{1}{\alpha s} + \frac{y}{\alpha s}\right) \leq 3s.$$

From which we obtain $\frac{\lambda}{3}s^*_L \leq s$. Combining these inequalities with the fact that $s \leq s^*_C$, we get $\frac{\lambda}{9}s^*_C \leq s \leq s^*_C$. That is, a λ-approximation algorithm for Lyndon Factor Maximization provides at least a $\lambda/9$ -approximation algorithm for this set of OCSP problems.

To finish the proof of Theorem 7, suppose for the sake of contradiction there exists a λ-approximation algorithm for Lyndon factor maximization for some constant $\lambda < 1$. Consider the set of OCSPs problems described in beginning of Section 4.2 with arity k such that $1/k < \lambda/9$. With our reduction, we obtain a polynomial-time algorithm that can find a solution with approximation ratio better than $|\Lambda|/k! = 1/k$, proving the Unique Games Conjecture false by Theorem 12.

5 Open Problems

We leave open the problem of establishing similar ETH hardness results for the maximization problem. We also leave open the problem of finding a (non-constant factor) approximation algorithm for either the minimization or maximization problem.

References

1 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "runs" theorem. *SIAM J. Comput.*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.

2 Hideo Bannai, Juha Kärkkäinen, Dominik Köppl, and Marcin Piatkowski. Constructing the bijective BWT. *CoRR*, abs/1911.06985, 2019. arXiv:1911.06985.

3 Hideo Bannai, Juha Kärkkäinen, Dominik Köppl, and Marcin Piatkowski. Indexing the bijective BWT. In *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019, June 18-20, 2019, Pisa, Italy*, pages 17:1–17:14, 2019. doi:10.4230/LIPIcs.CPM.2019.17.

4 Jason W. Bentley, Daniel Gibney, and Sharma V. Thankachan. On the complexity of bwt-runs minimization via alphabet reordering. In *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, pages 15:1–15:13, 2020. doi:10.4230/LIPIcs.ESA.2020.15.

5 Moses Charikar, Venkatesan Guruswami, and Rajsekar Manokaran. Every permutation CSP of arity 3 is approximation resistant. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009, Paris, France, 15-18 July 2009*, pages 62–73, 2009. doi:10.1109/CCC.2009.29.

6 Moses Charikar, Konstantin Makarychev, and Yury Makarychev. On the advantage over random for maximum acyclic subgraph. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 625–633, 2007. doi:10.1109/FOCS.2007.47.

7 Kuo Tsai Chen, Ralph H Fox, and Roger C Lyndon. Free differential calculus, iv. the quotient groups of the lower central series. *Annals of Mathematics*, pages 81–95, 1958.

8 Amanda Clare and Jacqueline W. Daykin. Enhanced string factoring from alphabet orderings. *Inf. Process. Lett.*, 143:4–7, 2019. doi:10.1016/j.ipl.2018.10.011.

9 Amanda Clare, Jacqueline W. Daykin, Thomas Mills, and Christine Zarges. Evolutionary search techniques for the lyndon factorization of biosequences. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*, pages 1543–1550, 2019. doi:10.1145/3319619.3326872.

10 Maxime Crochemore and Dominique Perrin. Two-way string matching. *J. ACM*, 38(3):651–675, 1991. doi:10.1145/116825.116845.

11 Jean-Pierre Duval. Génération d'une section des classes de conjugaison et arbre des mots de lyndon de longueur bornée. *Theor. Comput. Sci.*, 60:255–283, 1988. doi:10.1016/0304-3975(88)90113-2.

12 Isamu Furuya, Yuto Nakashima, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Lyndon factorization of grammar compressed texts revisited. In Gonzalo Navarro, David Sankoff, and Binhai Zhu, editors, *Annual Symposium on Combinatorial Pattern Matching, CPM 2018, July 2-4, 2018 - Qingdao, China*, volume 105 of *LIPIcs*, pages 24:1–24:10. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CPM.2018.24.

13 Joseph Yossi Gil and David Allen Scott. A bijective string sorting transform. *CoRR*, abs/1201.3077, 2012. arXiv:1201.3077.

14 Venkatesan Guruswami, Johan Håstad, Rajsekar Manokaran, Prasad Raghavendra, and Moses Charikar. Beating the random ordering is hard: Every ordering CSP is approximation resistant. *SIAM J. Comput.*, 40(3):878–914, 2011. doi:10.1137/090756144.

15 Venkatesan Guruswami and Yuan Zhou. Approximating bounded occurrence ordering csps. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, pages 158–169, 2012. doi:10.1007/978-3-642-32512-0_14.

16 Johan Håstad. Some optimal inapproximability results. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 1–10, 1997. doi:10.1145/258533.258536.

17 Christophe Hohlweg and Christophe Reutenauer. Lyndon words, permutations and trees. *Theor. Comput. Sci.*, 307(1):173–178, 2003. doi:10.1016/S0304-3975(03)00099-9.

18 Tomohiro I, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Faster lyndon factorization algorithms for SLP and LZ78 compressed text. *Theor. Comput. Sci.*, 656:215–224, 2016. doi:10.1016/j.tcs.2016.03.005.

19 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

20 Juha Kärkkäinen, Dominik Kempa, Yuto Nakashima, Simon J. Puglisi, and Arseny M. Shur. On the size of lempel-ziv and lyndon factorizations. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, volume 66 of *LIPIcs*, pages 45:1–45:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.45.

21 Subhash Khot. On the unique games conjecture. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, page 3, 2005. doi:10.1109/SFCS.2005.61.

22 Dong Kyue Kim, Jeong Seop Sim, Heejin Park, and Kunsoo Park. Linear-time construction of suffix arrays. In *Combinatorial Pattern Matching, 14th Annual Symposium, CPM 2003, Morelia, Michocán, Mexico, June 25-27, 2003, Proceedings*, pages 186–199, 2003. doi:10.1007/3-540-44888-8_14.

23 Eun Jung Kim and Daniel Gonçalves. On exact algorithms for the permutation CSP. *Theor. Comput. Sci.*, 511:109–116, 2013. doi:10.1016/j.tcs.2012.10.035.

24 Manfred Kufleitner. On bijective variants of the burrows-wheeler transform. In *Proceedings of the Prague Stringology Conference 2009, Prague, Czech Republic, August 31 - September 2, 2009*, pages 65–79, 2009. URL: http://www.stringology.org/event/2009/p07.html.

25 Pierre Lalonde and Arun Ram. Standard lyndon bases of lie algebras and enveloping algebras. *Transactions of the American Mathematical Society*, 347(5):1821–1830, 1995.

26 M. Lothaire. *Combinatorics on words*, volume 17. Cambridge university press, 1997.

27 Lily Major, Amanda Clare, Jacqueline W. Daykin, Benjamin Mora, Leonel Jose Peña Gamboa, and Christine Zarges. Evaluation of a permutation-based evolutionary framework for lyndon factorizations. In *Parallel Problem Solving from Nature - PPSN XVI - 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part I*, pages 390–403, 2020. doi:10.1007/978-3-030-58112-1_27.

28 Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. Sorting suffixes of a text via its lyndon factorization. In Jan Holub and Jan Zdárek, editors, *Proceedings of the Prague Stringology Conference 2013, Prague, Czech Republic, September 2-4, 2013*, pages 119–127. Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, 2013. URL: http://www.stringology.org/event/2013/p11.html.

29 Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. Suffix array and lyndon factorization of a text. *J. Discrete Algorithms*, 28:2–8, 2014. doi:10.1016/j.jda.2014.06.001.

30 Marcin Mucha. Lyndon words and short superstrings. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 958–972, 2013. doi:10.1137/1.9781611973105.69.

31 Alantha Newman. Cuts and orderings: On semidefinite relaxations for the linear ordering problem. In *Approximation, Randomization, and Combinatorial Optimization, Algorithms and Techniques, 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2004, and 8th International Workshop on Randomization and Computation, RANDOM 2004, Cambridge, MA, USA, August 22-24, 2004, Proceedings*, pages 195–206, 2004. doi:10.1007/978-3-540-27821-4_18.

32 Jaroslav Opatrny. Total ordering problem. *SIAM J. Comput.*, 8(1):111–114, 1979. doi:10.1137/0208008.

33 Prasad Raghavendra. Optimal algorithms and inapproximability results for every csp? In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 245–254, 2008. doi:10.1145/1374376.1374414.

34 Kazuya Tsuruta, Dominik Köppl, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Grammar-compressed self-index with lyndon words. *CoRR*, abs/2004.05309, 2020. `arXiv:2004.05309`.

35 Yuki Urabe, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. On the size of overlapping lempel-ziv and lyndon factorizations. In Nadia Pisanti and Solon P. Pissis, editors, *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019, June 18-20, 2019, Pisa, Italy*, volume 128 of *LIPIcs*, pages 29:1–29:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.CPM.2019.29`.

Reachability in Two-Parametric Timed Automata with One Parameter Is EXPSPACE-Complete

Stefan Göller

School of Electrical Engineering and Computer Science, Universität Kassel, Germany

Mathieu Hilaire

Université Paris-Saclay, ENS Paris-Saclay, Laboratoire Spécification et Vérification (LSV), CNRS, Gif-sur-Yvette, France

Abstract

Parametric timed automata (PTA) have been introduced by Alur, Henzinger, and Vardi as an extension of timed automata in which clocks can be compared against parameters. The reachability problem asks for the existence of an assignment of the parameters to the non-negative integers such that reachability holds in the underlying timed automaton. The reachability problem for PTA is long known to be undecidable, already over three parametric clocks.

A few years ago, Bundala and Ouaknine proved that for PTA over two parametric clocks and one parameter the reachability problem is decidable and also showed a lower bound for the complexity class $\mathsf{PSPACE}^{\mathsf{NEXP}}$. Our main result is that the reachability problem for parametric timed automata over two parametric clocks and one parameter is EXPSPACE-complete.

For the EXPSPACE lower bound we make use of deep results from complexity theory, namely a serializability characterization of EXPSPACE (in turn based on Barrington's Theorem) and a logspace translation of numbers in Chinese Remainder Representation to binary representation due to Chiu, Davida, and Litow. It is shown that with small PTA over two parametric clocks and one parameter one can simulate serializability computations.

For the EXPSPACE upper bound, we first give a careful exponential time reduction from PTA over two parametric clocks and one parameter to a (slight subclass of) parametric one-counter automata over one parameter based on a minor adjustment of a construction due to Bundala and Ouaknine. For solving the reachability problem for parametric one-counter automata with one parameter, we provide a series of techniques to partition a fictitious run into several carefully chosen subruns that allow us to prove that it is sufficient to consider a parameter value of exponential magnitude only. This allows us to show a doubly-exponential upper bound on the value of the only parameter of a PTA over two parametric clocks and one parameter. We hope that extensions of our techniques lead to finally establishing decidability of the long-standing open problem of reachability in parametric timed automata with two parametric clocks (and arbitrarily many parameters) and, if decidability holds, determining its precise computational complexity.

2012 ACM Subject Classification Theory of computation → Timed and hybrid models; Theory of computation → Automata extensions

Keywords and phrases Parametric Timed Automata, Computational Complexity, Reachability

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.36

Related Version *Full Version*: `https://arxiv.org/abs/2011.07091`

Funding *Stefan Göller*: The author was supported by the Agence Nationale de la Recherche grant no. ANR-17-CE40-0010.

Acknowledgements We thank Benedikt Bollig and Karin Quaas for discussions and feedback.

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 36; pp. 36:1–36:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Background. In the 1990's *timed automata* have been introduced by Alur and Dill [2]. They extend finite automata by clocks that can be compared against integer constants and provide a popular formalism to reason about the behavior of real-time systems with desirable algorithmic properties; for instance the reachability/emptiness problem is decidable and in fact PSPACE-complete [1].

For a more general means to specify the behavior of under-specified systems, such as embedded systems, Alur, Henzinger and Vardi [3] have introduced *parametric timed automata (PTA)* only a few years later. Here, the clocks can additionally be compared against parameters that can take unspecified non-negative integer values. Towards the verification of safety properties, or loosely speaking ruling out the existence of an execution to a bad state, the *reachability problem for PTA* in turn asks for the existence of an assignment of the parameters to the non-negative integers such that reachability holds in the resulting timed automaton.

A clock of a PTA that is being compared to at least one parameter is called *parametric*. On the negative side, it has been shown in [3] that already for PTA that contain *three parametric clocks* reachability is undecidable – even in the presence of one parameter [8]. On the positive side however, Alur, Henzinger and Vardi have shown in [3] that reachability is decidable for PTA that contain only *one parametric clock*, yet by an algorithm whose running time is non-elementary.

Reachability in PTA with two or less parametric clocks has not attracted much attention for many years, up until recently.

For PTA over *one parametric clock*, Bundala and Ouaknine have shown a first elementary complexity upper bound for the reachability problem; it is shown to be NEXP-hard and in 2NEXP [10]. The matching NEXP upper bound has been proven by Beneš et al. in [8] (also in the continuous time setting), we refer to [9] for an alternative proof by Bollig, Quaas and Sangnier using alternating two-way automata.

Bundala and Ouaknine [10] have recently advanced the decidability and complexity status of the reachability problem for PTA over *two parametric clocks* [10]: it is shown that in presence of *one parameter* the reachability problem is decidable and hard for the complexity class $\mathsf{PSPACE}^{\mathsf{NEXP}}$. To the best of our knowledge, this is in fact the largest subclass of PTA for which reachability is known to be decidable. For showing the above-mentioned decidability result [10] provides a reduction from PTA over two parametric clocks to a suitable formalism of *parametric one-counter automata*. Such an approach via parametric one-counter automata has already successfully been applied to model checking freeze-LTL as shown by Demri and Sangnier [12] and Lechner et al. [21], yet notably over a weaker model of parametric one-counter automata than the one introduced in [10]. On this note, it is worth mentioning that inter-reductions between the reachability problem of (non-parametric) timed automata involving two clocks and one-counter automata have already been established by Haase et al. [16, 17].

Decidability of reachability in PTA over two parametric clocks (without parameter restrictions) is still considered to be a challenging open problem to the best of our knowledge. For instance, as already remarked in [3], there is an easy reduction from the existential fragment of Presburger Arithmetic with divisibility to reachability in PTA over two parametric clocks.

Our contribution. Our main result (Theorem 4) states that reachability in parametric timed automata over two parametric clocks and one parameter is EXPSPACE-complete. Our contribution is two-fold.

Inspired by [13, 15], for the EXPSPACE lower bound we make use of deep results from complexity theory, namely a serializability characterization of EXPSPACE (in turn originally based on Barrington's Theorem [7]) and a logspace translation of numbers in Chinese Remainder Representation to binary representation due to Chiu, Davida, and Litow [11]. It is shown that with small PTA over two parametric clocks and one parameter one can simulate serializability computations.

For the EXPSPACE upper bound, we first give a careful exponential time reduction from PTA over two parametric clocks and one parameter to a (slight subclass of) parametric one-counter automata over one parameter based on a minor adjustment of a construction due to Bundala and Ouaknine [10]. In solving the reachability problem for parametric one-counter automata with one parameter, we provide a series of techniques to partition a fictitious run into several carefully chosen subruns that allow us to prove that it is sufficient to consider a parameter value of exponential magnitude. This allows us to show a doubly-exponential upper bound on the value of the only parameter of PTA with two parametric clocks and one parameter. We hope that extensions of our techniques lead to finally establishing decidability of the long-standing open problem of reachability in parametric timed automata with two parametric clocks (and arbitrarily many parameters) and, if decidability holds, determining its precise computational complexity.

As the results in [2], our results hold for PTA over discrete time. Indeed, for PTA with closed (i.e., non-strict) clock constraints and parameters ranging over integers, techniques [19, 22] exist that allow to reduce the reachability problem over continuous time to discrete time. There is a plethora of variants of PTA that have recently been studied, we refer to [4] for an extensive overview by André.

Overview of this paper. In Section 2 we introduce general notation, in particular PTA. Our EXPSPACE lower bound can be found in Section 3. Section 4 introduces parametric one-counter automata and states an exponential time reduction from PTA to this model. In Section 5 we introduce semiruns of parametric one-counter automata, a central notion of runs we make modifications on. Our upper bounds are the subject of Section 6. The full version of this paper is available on arXiv [14].

2 Preliminaries

By $\mathbb{N} = \{0, 1, \ldots\}$ we denote the *non-negative integers.* For every finite alphabet A we denote by A^* the set of finite words over A and the empty word by ε. For all $a \in A$ and all $w \in A^*$ let $|w|_a$ denote the number of occurrences of the letter a in w. For every finite set $M \subset \mathbb{N} \setminus \{0\}$ let $\mathsf{LCM}(M) = \min\{n \geq 1 \mid \forall m \in M \setminus \{0\} : m|n\}$ denote the least common multiple of the elements in M. For any $j \in \mathbb{N}$ let $\mathsf{LCM}(j) = \mathsf{LCM}([1, j])$ denote the least common multiple of the numbers $\{1, \ldots, j\}$.

A *guard* over a finite set of clocks Ω and a finite set of parameters P is a comparison of the form $g = \omega \bowtie e$, where $\omega \in \Omega$, $e \in P \cup \mathbb{N}$, and $\bowtie \in \{<, \leq, =, \geq, >\}$; in case $e \in P$ we call g *parametric* and *non-parametric* otherwise. We denote by $\mathcal{G}(\Omega, P)$ the *set of guards* over the set of clocks Ω and the set of parameters P. The *size* $|g|$ of a guard $g = \omega \bowtie e$ is defined as $\log(e)$ if $e \in \mathbb{N}$ and 1 otherwise. A *clock valuation* is a function from Ω to $\mathbb{N}$; we write $\vec{0}$ to denote the clock valuation $\omega \mapsto 0$. For each clock valuation v and each $t \in \mathbb{N}$ we denote by

$v+t$ the clock valuation $\omega \mapsto v(\omega)+t$. A *parameter valuation* is a function μ from P to $\mathbb{N}$. For every guard $g = \omega \bowtie p$ with $p \in P$ (resp. $g = \omega \bowtie k$ with $k \in \mathbb{N}$) we write $v \models_\mu g$ if $v(\omega) \bowtie \mu(p)$ (resp. $v(\omega) \bowtie k$).

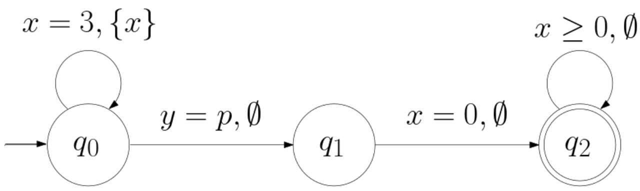


Figure 1 An example of a PTA. The automaton consists of three states, the set of clocks is $\{x, y\}$, the set of parameters is $\{p\}$. The edges are represented by arrows labeled with the corresponding guard and the set of clocks U to be reset. A parameter valuation μ witnesses that reachability holds for this PTA if, and only, if $\mu(p) \in 3\mathbb{Z}$.

A parametric timed automaton as introduced in [3] is a finite automaton extended with a finite set of parameters P and a finite set of clocks Ω that all progress at the same rate and that can be individually reset to zero. Moreover, every transition is labeled by a guard over Ω and P and by a set of clocks to be reset. Formally, a *parametric timed automaton* (*PTA* for short) is a tuple $\mathcal{A} = (Q, \Omega, P, R, q_{init}, F)$, where

- Q is a non-empty finite *set of control states*,
- Ω is a non-empty finite *set of clocks*,
- P is a finite *set of parameters*,
- $R \subseteq Q \times \mathcal{G}(\Omega, P) \times \mathcal{P}(\Omega) \times Q$ is a finite *set of rules*,
- $q_{init} \in Q$ is an *initial control state*, and
- $F \subseteq Q$ is a *set of final control states*.

A clock $\omega \in \Omega$ is called *parametric* if there exists some $(q, g, U, q') \in R$ such that the guard g is parametric. We also refer to $\mathcal{A}$ as an (m,n)-PTA if $m = |\{\omega \in \Omega \mid \omega \text{ is parametric}\}|$ is the number of parametric clocks and $n = |P|$ is the number of parameters of $\mathcal{A}$ – sometimes we also just write $(m,*)$-PTA (resp. $(*,n)$-PTA) when n (resp. m) is a priori not fixed.

The *size* of $\mathcal{A}$ is defined as $|\mathcal{A}| = |Q| + |\Omega| + |P| + |R| + \sum_{(q,g,U,q') \in R} |g|$. Let $\mathsf{Consts}(\mathcal{A})$ denote the set of constants that appear in the guards of the rules of $\mathcal{A}$. By $\mathsf{Conf}(\mathcal{A}) = Q \times \mathbb{N}^\Omega$ we denote the set of *configurations* of $\mathcal{A}$. We prefer however to denote a configuration by $q(v)$ instead of (q, v).

For each parameter valuation $\mu : P \to \mathbb{N}$ and each $(\delta, t) \in R \times \mathbb{N}$ with $\delta = (q, g, U, q')$, let $\xrightarrow{\delta,t,\mu}$ denote the binary relation on $\mathsf{Conf}(\mathcal{A})$, where $q(v) \xrightarrow{\delta,t,\mu} q'(v')$ if $v + t \models_\mu g$, $v'(\omega) = 0$ for all $\omega \in U$ and $v'(\omega) = v(\omega) + t$ for all $\omega \in \Omega \setminus U$. A *$\mu$-run* from $q_0(v_0)$ to $q_n(v_n)$ is a sequence $q_0(v_0) \xrightarrow{\delta_1,t_1,\mu} q_1(v_1) \cdots \xrightarrow{\delta_n,t_n,\mu} q_n(v_n)$. In case $P = \{p\}$ is a singleton and $\mu(p) = N$ we prefer to say *N-run* instead of μ-run and write $q(v) \xrightarrow{N} q'(v')$ to denote $q(v) \xrightarrow{\mu} q'(v')$. We say *reachability holds* for $\mathcal{A}$ if there is a μ-run from $q_{init}(\vec{0})$ to some configuration $q(v)$ for some $q \in F$, some $v \in \mathbb{N}^\Omega$, and some $\mu \in \mathbb{N}^P$. We refer to Figure 1 for an instance of a PTA for which reachability holds.

It is worth mentioning that there are further modes of time valuations and guards which exist in the literature, we refer to [5] for a recent overview.

We are interested in the following decision problem.

(m,n)-PTA-REACHABILITY

INPUT: A (m,n)-PTA $\mathcal{A}$.
QUESTION: Does reachability hold for $\mathcal{A}$?

Alur et al. have already shown in their seminal paper that PTA-Reachability is in general undecidable, already in the presence of only three parametric clocks [3], Beneš et al. strengthened this when only one parameter is present [8].

▶ **Theorem 1** ([8]). $(3,1)$-PTA-Reachability *is undecidable.*

To the contrary, $(1,*)$-PTA-Reachability has recently been shown to be complete for NEXP, where a non-elementary upper bound was initially given by Alur et al. [3].

▶ **Theorem 2** ([10, 8, 9]). $(1,*)$-PTA-Reachability *is* NEXP*-complete.*

On the other end, decidability of $(2,*)$-PTA-Reachability is still open to the best of our knowledge. In presence of precisely one parameter the following is known.

▶ **Theorem 3** ([10]). $(2,1)$-PTA-Reachability *is decidable and* $\mathsf{PSPACE}^{\mathsf{NEXP}}$*-hard.*

The following theorem states our main result.

▶ **Theorem 4.** $(2,1)$-PTA-Reachability *is* EXPSPACE*-complete.*

3 An EXPSPACE lower bound via serializability

In this section we show an EXPSPACE lower bound for $(2,1)$-PTA-Reachability. We show that on small PTA with two parametric clocks x and y and one parameter p one can perform both (i) PSPACE computations and (ii) compute $x-y \bmod p$ modulo numbers that are dynamically given in binary. Building upon these auxiliary gadgets we show how to implement bottleneck computations in a leaf language characterization of EXPSPACE [13]. We assume the reader is familiar with Turing machines and standard complexity classes such as LOGSPACE, PSPACE and EXPSPACE. We refer to [23, 6] for further details on complexity. We also assume the reader is familiar with (deterministic) finite automata and regular languages, we refer to [18] for more details on this.

For each $a,b \in \mathbb{Z}$ we define $[a,b] = \{k \in \mathbb{Z} \mid a \leq k \leq b\}$. For each $i,n \in \mathbb{N}$ let $\textsc{Bit}_i(n)$ denote the i-th least significant bit of the binary presentation of n, i.e. $n = \sum_{i\in\mathbb{N}} 2^i \cdot \textsc{Bit}_i(n)$. For each $m \geq 1$, by $\textsc{Bin}_m(n) = \textsc{Bit}_0(n)\cdots\textsc{Bit}_{m-1}(n)$ we denote the sequence of the first m least significant bits of the binary presentation of n, i.e. the least significant bit is on the left. Conversely, given a binary string $w = w_0 \cdots w_{n-1} \in \{0,1\}^n$ of length n we denote by $\textsc{Val}(w) = \sum_{i=0}^{n-1} 2^i \cdot w_i \in [0, 2^n - 1]$ the value of w interpreted as a non-negative integer.

Let $\mathcal{A}$ be a parametric timed automaton over a set of clocks Ω with two parametric clocks x and y. We say a valuation $v : \Omega \to \mathbb{N}$ is *bit-compatible* if $v(\omega) \in \{0,1\}$ for all non-parametric clocks $\omega \in \Omega$. Assume moreover that Ω contains *non-parametric clocks* $\Theta_+ \cup \Theta_-$, where Θ is some set and $\Theta_+ = \{\vartheta^+ \mid \vartheta \in \Theta\}$ and $\Theta_- = \{\vartheta^- \mid \vartheta \in \Theta\}$ are two disjoint corresponding copies of Θ; in this case, for any valuation $v : \Omega \to \mathbb{N}$ we define the mapping $\widehat{v} : \Theta \to \{0,1\}$ as $\widehat{v}(\vartheta) = 0$ if $v(\vartheta^+) = v(\vartheta^-)$ and $\widehat{v}(\vartheta) = 1$ otherwise. In the following we call such non-parametric clocks $\{\vartheta^+, \vartheta^- \mid \vartheta \in \Theta\}$, appearing as implicit pairs, *bit clocks* since they are used to encode bits.

▶ **Definition 5.** *A* $(2,1)$*-PTA* $\mathcal{A} = (Q, \Omega, \{p\}, R, q_{init}, \{q_{fin}\})$ *whose parametric clocks are* x *and* y *and whose one parameter is* p computes a function $f : \mathbb{N} \times \{0,1\}^n \to \{0,1\}^m$ *if its set of clocks* Ω *contains two disjoint sets of*

- *non-parametric "input" bit clocks* $\{{in_0}^+, {in_0}^-, \ldots, in_{n-1}^+, in_{n-1}^-\}$ *and*
- *non-parametric "output" bit clocks* $\{{out_0}^+, {out_0}^-, \ldots, {out_{m-1}}^+, {out_{m-1}}^-\}$

such that for all $N \in \mathbb{N}$ *and all bit-compatible* $v_0 : \Omega \to [0, N-1]$ *we have*

1. *$q_{init}(v_0) \xrightarrow{N}{}^{*} q_{fin}(v')$ for some bit-compatible $v' : \Omega \to [0, N-1]$ and*
2. *for all $v' : \Omega \to \mathbb{N}$ for which $q_{init}(v_0) \xrightarrow{N}{}^{*} q_{fin}(v')$ we have*
   - *$v' \in [0, N-1]^{\Omega}$ is bit-compatible,*
   - *$\widehat{v'}(in_i) = \widehat{v_0}(in_i)$ for all $i \in [0, n-1]$,*
   - *$v'(x) - v'(y) \equiv v_0(x) - v_0(y)$ mod N, and*
   - *$\prod_{j=0}^{m-1} \widehat{v'}(out_j) = f(v_0(x) - v_0(y) \text{ mod } N, \prod_{i=0}^{n-1} \widehat{v_0}(in_i))$, where $\prod$ denotes concatenation.*

The following lemma essentially has its roots in the PSPACE-hardness proof for the emptiness problem for timed automata (without parameters) introduced by Alur and Dill [2], however constructed to satisfy the carefully chosen interface given by Definition 5.

▶ **Lemma 6.** *For every* PSPACE*-computable function $g : \{0,1\}^n \to \{0,1\}^m$ one can compute in polynomial time in $n+m$ a $(2,1)$-PTA computing the function $f : \mathbb{N} \times \{0,1\}^n \to \{0,1\}^m$, where $f(k,w) = g(w)$ for all $(k,w) \in \mathbb{N} \times \{0,1\}^n$.*

The following lemma shows that PTA with two parametric clocks and one parameter can compute modulo dynamically given numbers in binary.

▶ **Lemma 7.** *One can compute in polynomial time in $n+m$ a $(2,1)$-PTA with two parametric clocks and one parameter that computes the function $f : \mathbb{N} \times \{0,1\}^n \to \{0,1\}^m$, where $f(k,w) = \textsc{Bin}_m(k \text{ mod } \textsc{Val}(w))$.*

We are now ready to state the main result of this section.

▶ **Theorem 8.** $(2,1)$-PTA-Reachability *is* EXPSPACE*-hard.*

For each language $L \subseteq A^*$ let $\chi_L : A^* \to \{0,1\}$ denote the characteristic function of L. By $\preceq_n$ we denote the lexicographic order on n-bit strings, thus $w \preceq_n v$ if $\textsc{Val}(w) \leq \textsc{Val}(v)$, e.g. $0101 \preceq_4 0011$.

Our EXPSPACE lower bound proof makes use of the following characterization of EXPSPACE, which is a slight padded adjustment of the leaf-language characterization of PSPACE from [20], which in turn has its roots in Barrington's Theorem [7].

▶ **Theorem 9** (Theorem 2 in [13]). *For every language $L \subseteq \{0,1\}^*$ in* EXPSPACE *there exists a polynomial $s : \mathbb{N} \to \mathbb{N}$, a regular language $R \subseteq \{0,1\}^*$, and a language $U \in$* LOGSPACE *such that for all $w \in \{0,1\}^n$ we have*

$$w \in L \quad \iff \quad \prod_{m=0}^{2^{2^{s(n)}}-1} \chi_U(w \cdot \textsc{Bin}_{2^{s(n)}}(m)) \in R, \tag{1}$$

where $\cdot$ and $\prod$ denote string concatenation.

Let us fix any language L in EXPSPACE and assume $L \subseteq \{0,1\}^*$ without loss of generality.

Applying Theorem 9, let us fix the regular language $R \subseteq \{0,1\}^*$ along with some fixed deterministic finite automaton (DFA for short) $D = (Q_D, \{0,1\}, q_0, \delta_D, F_D)$ with $L(D) = R$, the fixed polynomial s and the fixed language $U \in$ LOGSPACE.

Let us moreover fix an input $w \in \{0,1\}^n$ of length n for L. Figure 2 rephrases the characterization (1) in Theorem 9 in terms of an execution of a program that returns 1 if, and only if, $w \in L$.

The following lemma gives us a helpful initial gadget PTA that allows us to enforce that the parameter p can only be evaluated to numbers that are larger than $2^{2^{s(n)}}$, thus being helpful for storing variables up to the value 2^{2^n}.

```
(1)   var $q \in Q_D$
(2)   var $b \in \{0,1\}$
(3)   var $B \in \mathbb{N}$
(4)   $q := q_0$;
(5)   $B := 0$;
(6)   while $B < 2^{2^n}$ loop
(7)       $b := \chi_U(w \cdot \textsc{Bin}_{2^{s(n)}}(B))$
(8)       $q := \delta_D(q,b)$
(9)       $B := B + 1$
(10)  end loop
(11)  return $q \in F$
```

Figure 2 A program returning 1 if, and only if, $w \in L$ (using the charactization in Theorem 9), where $D = (Q_D, \{0,1\}, q_0, \delta_D, F_D)$ is some deterministic finite automaton such that $L(D) = R$.

▶ **Lemma 10.** *One can compute in polynomial time in n some parametric timed automaton $\mathcal{A}_{big} = (Q_{big}, \Omega_{big}, \{p\}, R_{big}, q_{big,init}, \{q_{big,fin}\})$ with two parametric clocks $x, y \in \Omega_{big}$ and one parameter p such that*

1. *$q_{big,init}(\vec{0}) \xrightarrow{N} q_{big,fin}(v')$ for some $v' : \Omega_{big} \to \mathbb{N}$ for some $N \in \mathbb{N}$, and*
2. *for all $N \in \mathbb{N}$ and all $v' : \Omega_{big} \to \mathbb{N}$ we have $q_{big,init}(\vec{0}) \xrightarrow{N}^{*} q_{big,fin}(v')$ implies $N > 2^{2^{s(n)}}$.*

Using the above gadgets one can show that the program in Figure 2 can indeed be simulated by small $(2,1)$-PTA, whose proof we sketch below.

▶ **Lemma 11.** *One can compute in polynomial time in n a $(2,1)$-PTA $\mathcal{A}$ for which reachability holds, if and only if, the execution of the program depicted in Figure 2 returns* 1.

Proof (sketch). The initial part of $\mathcal{A}$ will consist of the gadget PTA $\mathcal{A}_{big}$ from Lemma 10 and allow us to enforce an assignment of $\mathcal{A}$'s only parameter p to some value $N > 2^{2^{s(n)}}$. We store the variable B of the program in Figure 2 as the difference between $\mathcal{A}$'s two parametric clocks x and y modulo N. We only sketch the most crucial program line (7), namely computing the bit $\chi_U(w \cdot \textsc{Bin}_{2^{s(n)}}(B))$, where we recall that U is in LOGSPACE.

For simulating a suitable logspace Turing machine on this exponentially large input our PTA $\mathcal{A}$ will use $O(\log(n + 2^{s(n)})) = \text{poly}(n)$ auxiliary bit clocks, say $\mathcal{J}$, to store in binary the position of the input head and further $O(\log(n + 2^{s(n)})) = \text{poly}(n)$ auxiliary bit clocks for storing the working tape. Reading and writing on the working tape as well as updating the position of the input head can be done quite straightforwardly using polynomially many bit clocks. The main challenge is to access the cell content $\textsc{Bit}_j(w \cdot \textsc{Bin}_{2^{s(n)}}(B))$, where the address j can directly (in binary) be stored using the above-mentioned bit clocks $\mathcal{J}$.

To compute $\textsc{Bit}_j(w \cdot \textsc{Bin}_{2^{s(n)}}(B))$ we access B on the fly via its Chinese Remainder Representation $\mathsf{CRR}(B)$ that we define next: Let p_i denote the i-th prime number and assume $\prod_{i=1}^{m} p_i > B$ for some $m \in \mathbb{N}$, then $\mathsf{CRR}_m(B)$ denotes the bit tuple $(b_{i,r})_{i \in [1,m], r \in [0,p_i - 1]}$, where $b_{i,r} = 1$ if $B \bmod p_i = r$ and $b_{i,r} = 0$ otherwise. The individual input bits to $\mathsf{CRR}(B)$ can be sub-computed via our modulo gadget from Lemma 7. The individual input bits to $\textsc{Bin}_{2^{s(n)}}(B)$ can be obtained by a composition of the latter access to $\mathsf{CRR}(B)$ and simulating a logspace Turing machine that computes $\textsc{Bin}_{2^{s(n)}}(B)$ from $\mathsf{CRR}(B)$ by a result by Chiu, Davida, and Litow [11]. ◀

4 From two-parametric timed automata with one parameter to parametric one-counter automata

Being introduced by Bundala and Ouaknine in [10], we define parametric one-counter automata. These are automata that can manipulate a counter that can be incremented or decremented, parametrically or not, compared against constants or parameters, and with divisibility tests modulo constants. It is worth mentioning that the notion of parametric one-counter automata from [10] is slightly more expressive than ours, allowing more operations.

After introducing parametric one-counter automata we mention Theorem 13, proven essentially already in [10] – again, however for a slightly more expressive model of parametric one-counter automata – that states that $(2,1)$-PTA-Reachability can be reduced in exponential time to the reachability problem of parametric one-counter automata over one parameter.

Given a set of parameters P we denote by $\mathsf{Op}(P)$ the *set of operations*, namely $\mathsf{Op}(P) = \mathsf{Op}_{\pm} \cup \mathsf{Op}_{\pm P} \cup \mathsf{Op}_{\mathsf{mod}\mathbb{N}} \cup \mathsf{Op}_{\bowtie\mathbb{N}} \cup \mathsf{Op}_{\bowtie P}$, where

- $\mathsf{Op}_{\pm} = \{-1, 0, +1\}$, $\mathsf{Op}_{\pm P} = \{+p, -p \mid p \in P\}$,
- $\mathsf{Op}_{\mathsf{mod}\mathbb{N}} = \{\mathsf{mod}\ c \mid c \in \mathbb{N}\}$,
- $\mathsf{Op}_{\bowtie\mathbb{N}} = \{\bowtie c \mid \bowtie \in \{<, \leq, =, \geq, >\}, c \in \mathbb{N}\}$, and $\mathsf{Op}_{\bowtie P} = \{\bowtie p \mid \bowtie \in \{<, \leq, =, \geq, >\}, p \in P\}$.

The *size* $|op|$ of an operation op is defined as $|op| = \log(c)$ if $op = \mathsf{mod}\ c$ or $op = \bowtie c$ with $c \in \mathbb{N}$ and $|op| = 1$ otherwise. We denote by *updates* those operations that lie in $\mathsf{Op}_{\pm} \cup \mathsf{Op}_{\pm P}$ and by *tests* those operations that lie in $\mathsf{Op}_{\mathsf{mod}\mathbb{N}} \cup \mathsf{Op}_{\bowtie\mathbb{N}} \cup \mathsf{Op}_{\bowtie P}$.

A *parametric one-counter automaton* (POCA for short) is a tuple $\mathcal{C} = (Q, P, R, q_{init}, F)$, where Q is a non-empty finite *set of control states*, P is a non-empty finite *set of parameters* that can take non-negative integer values, $R \subseteq Q \times \mathsf{Op}(P) \times Q$ is a finite *set of rules*, q_{init} is an *initial control state*, and $F \subseteq Q$ is a *set of final control states*. The *size* of $\mathcal{C}$ is defined as $|\mathcal{C}| = |Q| + |P| + |R| + \sum_{(q,op,q') \in R} |op|$. Let $\mathsf{Consts}(\mathcal{C})$ denote the constants that appear in the operations $op \in \mathsf{Op}_{\mathsf{mod}\mathbb{N}} \cup \mathsf{Op}_{\bowtie\mathbb{N}}$ for some rule (q, op, q') in R. By $\mathsf{Conf}(\mathcal{C}) = Q \times \mathbb{Z}$ we denote the set of *configurations* of $\mathcal{C}$. We prefer however to denote a configuration of $\mathsf{Conf}(\mathcal{C})$ by $q(z)$ instead of (q, z).

Being slightly non-standard we define configurations to take counter values over $\mathbb{Z}$ rather than over $\mathbb{N}$ for notational convenience. This does not cause any loss of generality as we allow guards that enable us to test if the value of the counter is greater or equal to zero.

▶ **Definition 12** (transition). *For every $op \in \mathsf{Op}(P)$, for every parameter valuation $\mu : P \to \mathbb{N}$, for every POCA $\mathcal{C}$, and for every two configurations $q(z)$ and $q'(z')$ in $\mathsf{Conf}(\mathcal{C})$ we define the* transition *$q(z) \xrightarrow{op,\mu} q'(z')$ if there exists some $(q, op, q') \in R$ such that either of the following holds*

(1) *$op = c \in \mathsf{Op}_{\pm}$ and $z' = z + c$,*

(2) *$op \in \mathsf{Op}_{\pm P}$, and either $op = +p$ and $z' = z + \mu(p)$, or $op = -p$ and $z' = z - \mu(p)$.*

(3) *$op = \mathsf{mod}\ c \in \mathsf{Op}_{\mathsf{mod}\mathbb{N}}$, $z = z'$ and $z' \equiv 0\ \mathsf{mod}\ c$,*

(4) *$op = \bowtie c \in \mathsf{Op}_{\bowtie\mathbb{N}}$, $z = z'$ and $z' \bowtie c$, or*

(5) *$op = \bowtie p \in \mathsf{Op}_{\bowtie P}$, $z = z'$ and $z' \bowtie \mu(p)$.*

Let $\mu : P \to \mathbb{N}$ be a parameter valuation. We just write $q(z) \xrightarrow{\mu} q'(z')$ if $q(z) \xrightarrow{op,\mu} q'(z')$ for some operation op. A *μ-run (or just run)* in $\mathcal{C}$ (from $q_0(z_0)$ to $q_n(z_n)$) is a sequence, possibly empty (i.e. $n = 0$), of the form $\pi = q_0(z_0) \xrightarrow{op_0,\mu} q_1(z_1) \cdots \xrightarrow{op_{n-1},\mu} q_n(z_n)$.

We say π is *accepting* if $q_0 = q_{init}$, $z_0 = 0$, and $q_n \in F$. We say *reachability holds* (for the POCA $\mathcal{C}$) if there exists an accepting μ-run for some $\mu \in \mathbb{N}^P$. We refer to Figure 3 for an

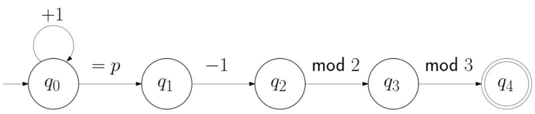


Figure 3 An example of a POCA. The automaton consists of five states and the set of parameters is $\{p\}$. The edges are represented by arrows labeled with the corresponding operations. A parameter valuation $\mu : \{p\} \to \mathbb{N}$ witnesses that reachability holds for the above POCA if, and only, if $\mu(p) \equiv 1 \bmod 6$.

instance of a POCA for which reachability holds. For any two $c, d \in [0, n]$ we define the *subrun* $\pi[c, d]$ *from* $q_c(z_c)$ *to* $q_d(z_d)$ *of* π as the μ-run $q_c(z_c) \xrightarrow{\pi_c,\mu} q_{c+1}(z_{c+1}) \cdots \xrightarrow{\pi_{d-1},\mu} q_d(z_d)$.

As expected, a *prefix* (resp. *suffix*) of π is an μ-run of the form $\pi[0, d]$ (resp. $\pi[d, n]$). In the particular case where $P = \{p\}$ is a singleton for some parameter p and $\mu(p) = N$, we write $q(z) \xrightarrow{op,N} q'(z')$ to denote $q(z) \xrightarrow{op,\mu} q'(z')$ and prefer to call a μ-run an N-*run*.

We define $\Delta(\pi) = z_n - z_0$ as the *counter effect* of the run π and for each $i \in [0, n-1]$ we define $\Delta(\pi, i) = \Delta(\pi[i, i+1])$ to denote the counter effect of the i-th transition of π. Its *length* is defined as $|\pi| = n$. As expected, let $\textsc{Values}(\pi) = \{z_i \mid i \in [0, n]\}$ denote the set of counter values of the configurations of the run π. The run π's *maximum* is defined as $\max(\pi) = \max(\textsc{Values}(\pi))$ and its *minimum* as $\min(\pi) = \min(\textsc{Values}(\pi))$.

The next theorem states an exponential time reduction from $(2, 1)$-PTA-Reachability to the reachability problem of POCA over one parameter whose counter values are bound by a linear function in the parameter value and its size. It has already been proven in the more general setting over an arbitrary number of parameters in [10], however using a POCA model allowing more operations.

▶ **Theorem 13.** *The following is computable in exponential time:*
INPUT: A $(2, 1)$-PTA $\mathcal{A}$.
OUTPUT: A POCA $\mathcal{C}$ over one parameter p such that

1. *for all $N \in \mathbb{N}$ all accepting N-runs π in $\mathcal{C}$ satisfy $\textsc{Values}(\pi) \subseteq [0, 4 \cdot \max(N, |\mathcal{C}|)]$, and*
2. *reachability holds for $\mathcal{A}$ if, and only if, reachability holds for $\mathcal{C}$.*

5 Semiruns, their bracket projection, and embeddings

In this section we motivate and introduce the notion of semiruns by loosening the conditions on runs, and define basic operations on them. These basic operations possibly change their counter values, length, or counter effect. We finally introduce the notion of embeddings, which provide a formal means to express when a semirun can structurally be found as a subsequence of another.

▶ **Definition 14** (semitransition). *Let $\mathcal{C} = (Q, P, R, q_{init}, F)$ be a POCA. For every operation $op \in \mathsf{Op}(P)$ and every $N \in \mathbb{N}$ and for every two configurations $q(z)$ and $q'(z')$ in $\mathsf{Conf}(\mathcal{C})$ we define the* semitransition $q(z) \xRightarrow{op,\mu} q'(z')$ *if there exists some $(q, op, q') \in R$ such that conditions (1),(2), and (3) hold and where conditions (4) and (5) are loosened by*

(4') *$op = \bowtie c \in \mathsf{Op}_{\bowtie\mathbb{N}}$ and $z = z'$, and*
(5') *$op = \bowtie p \in \mathsf{Op}_{\bowtie P}$, and $z = z'$.*

Thus, in a nutshell, when writing $q(z) \xRightarrow{op,\mu} q'(z')$ we do not require that the comparison tests against parameters or against constants hold; however the updates and the modulo tests against constants must be respected.

This naturally gives rise to the definition of a *μ-semirun*, which is defined as expected. Note that in particular every μ-run is a μ-semirun. The notion of an N-semirun, the relation $q(z) \xRightarrow{op,N} q'(z')$, the counter effect Δ, Values, min, max, subsemirun, prefix, suffix are defined as for runs.

Importantly, note also that semitransitions involving comparison tests are still syntactically present in semiruns. By a careful analysis, one can therefore possibly perform operations on N-semiruns in order to show that they are in fact N-runs.

Let Γ be any integer that is divisible by all constants in $\mathsf{Consts}(\mathcal{C})$ in some POCA $\mathcal{C}$. We define the *shifting* of an N-semirun π by Γ as $\pi + \Gamma = q_0(z_0 + \Gamma) \xRightarrow{\pi_0,N} q_1(z_1 + \Gamma) \cdots \xRightarrow{\pi_{n-1},N} q_n(z_n + \Gamma)$. Since there are no effective comparison tests and Γ is an integer that is divisible by all constants appearing in modulo tests in $\mathcal{C}$, it is clear that $\pi + \Gamma$ is again an N-semirun.

For two configurations $q_i(z_i)$ and $q_j(z_j)$ with $0 \leq i < j \leq n$, $q_i = q_j$, and $z_j - z_i = \Gamma$ we define the *gluing* of the configurations as

$$\pi - [i,j] \;=\; q_0(z_0) \;\cdots\; \xRightarrow{\pi_{i-1},N} q_i(z_i) \xRightarrow{\pi_j,N} q_{j+1}(z_{j+1} - \Gamma) \;\cdots\; \xRightarrow{\pi_{n-1},N} q_n(z_n - \Gamma).$$

When gluing the leftmost and rightmost configurations of pairwise non-intersecting intervals $I_1 = [a_1, b_1], \ldots, I_k = [a_k, b_k] \subseteq [0, n]$, assuming $b_i < a_{i+1}$ for all $1 \leq i < k$, and $q_{a_i} = q_{b_i}$ and $z_{b_i} - z_{a_i}$ is divisible by all constants in $\mathsf{LCM}(\mathsf{Consts}(\mathcal{C}))$ for all $1 \leq i \leq k$, we will use $\pi - I_1 - I_2 \cdots - I_k$ to denote the result corresponding to gluing each interval successively while shifting the others accordingly, formally $\pi^{(k)}$, where $\pi^{(1)} = \pi - [a_1, b_1]$ and inductively, $\pi^{(s)} = \pi^{(s-1)} - [a_s - \Sigma_{1 \leq j < s}(|I_j| - 1), b_s - \Sigma_{1 \leq j < s}(|I_j| - 1)]$ for all $s \in [2, k]$.

We define the projection ϕ of a semitransition $\tau = q(z) \xRightarrow{op,N} q'(z')$ to a word over the binary alphabet $\{[,]\}$, where transitions with $op = +p$ are mapped to [, transitions with $op = -p$ are mapped to], and all other transitions are mapped to the empty word ε. The mapping ϕ is naturally extended to a morphism on semiruns.

We are particularly interested in N-semiruns whose projection by ϕ contains as many opening as closing brackets and only a few pending (when read from left to right) opening or closing brackets. To make this formal, for all $k \in \mathbb{N}$ we define regular language

$$\Lambda_k = \left\{ w \in \{[,]\}^* : |w|_{[} = |w|_{]}, \forall u, v \in \{[,]\}^*.\ uv = w \implies |u|_{[} - |u|_{]} \in [-k, k] \right\}.$$

We will often prefer to view N-runs as N-semiruns. Indeed, in case N is sufficiently large we first view any N-run as an N-semirun, apply certain of the above-mentioned operations on them to obtain some $(N - \Gamma)$-semirun, where Γ is divisible by all constants appearing in the underlying POCA. However, we would then like to claim that the resulting $(N - \Gamma)$-semirun is in fact an $(N - \Gamma)$-run as desired, in particular the comparison tests need to hold. To do so, we introduce a notion when an N-semirun can be embedded into an M-semirun (possibly $N \neq M$) in the sense that operations are being preserved, source and target control states are being preserved and that with respect to some line $\ell \in \mathbb{Z}$ the counter value of each configuration of the embedding has the same orientation with respect to ℓ as the counter value of the configuration it corresponds to.

▶ **Definition 15** (ℓ-embedding). *Let $\ell \in \mathbb{Z}$. An N-semirun $\sigma = s_0(y_0) \cdots \xRightarrow{\sigma_{n-1},N} s_n(y_n)$ is an ℓ-*embedding *of an M-semirun $\pi = q_0(z_0) \cdots \xRightarrow{\pi_{m-1},M} q_m(z_m)$ if $s_0 = q_0$, $s_n = q_m$ and there exists an order-preserving injective mapping $\psi : [0, n] \to [0, m]$ such that*

- *$\sigma_i = \pi_{\psi(i)}$ for all $i \in [0, n-1]$, and*
- *$\ell \bowtie y_i$ if, and only if, $\ell \bowtie z_{\psi(i)}$ for all $\bowtie \in \{<, =, >\}$ and all $i \in [0, n]$.*

Moreover we say σ is max-falling, *resp.* min-rising, *w.r.t. π if $\max(\sigma) \leq \max(\pi)$, resp. if $\min(\sigma) \geq \min(\pi)$.*

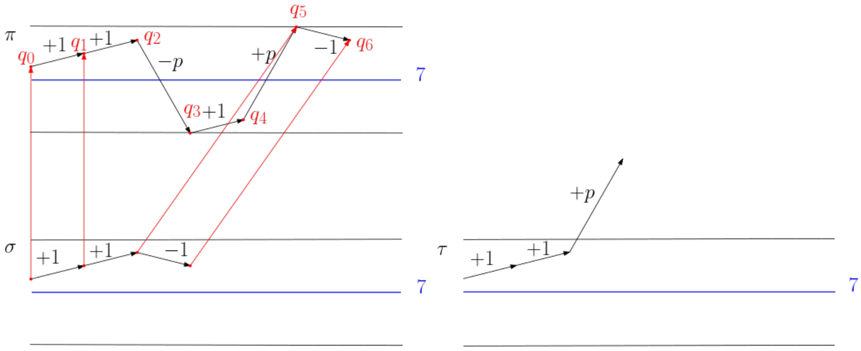


Figure 4 Example of a semirun σ that could possibly be a 7-embedding of the semirun π and a semirun τ that cannot.

Consider the semiruns π, σ and τ in Figure 4, where neither concrete counter values nor the control states of σ nor τ are mentioned. The semirun σ can possibly be an 7-embedding of π (if its source control control is q_0 and its target control state is q_6). However, τ cannot be a 7-embedding of π. Indeed, for every possible injection ψ such that $\tau_2 = +p = \pi_{\psi(2)}$, the counter value of τ at position 2 is strictly larger than 7, whereas the counter value of π at position $\psi(2)$ is strictly below 7.

It is immediate that an ℓ-embedding of an ℓ-embedding is again an ℓ-embedding. Moreover, if the target configuration of σ equals the source configuration of τ and σ and τ are ℓ-embeddings of σ' and τ' respectively, and $\sigma'\tau'$ is a semirun, then so is their concatenation $\sigma\tau$ an ℓ-embedding of $\sigma'\tau'$. Such basic properties will be used extensively in our proofs.

6 Upper bounds

In this section we state the Small Parameter Theorem which states that every POCA over one parameter and every sufficiently large parameter value N, accepting N-runs with counter values all in $[0, 4N]$ can be turned into accepting N'-runs for some smaller N'. After having stated the theorem one can show that together with Theorem 13 it implies an EXPSPACE upper bound for $(2,1)$-PTA-Reachability.

For each POCA $\mathcal{C} = (Q, P, R, q_{init}, F)$ we define the following constants:

$$Z_{\mathcal{C}} = \mathsf{LCM}(\mathsf{Consts}(\mathcal{C})) \qquad \Gamma_{\mathcal{C}} = \mathsf{LCM}(17 \cdot |Q|) \cdot Z_{\mathcal{C}}$$

$$\Upsilon_{\mathcal{C}} = 17 \cdot |Q| \cdot \mathsf{LCM}(17 \cdot |Q|) \cdot (17 \cdot |Q| \cdot Z_{\mathcal{C}} + 2) \qquad M_{\mathcal{C}} = 30 \cdot (\Upsilon_{\mathcal{C}} + \Gamma_{\mathcal{C}} + 1)$$

Since for every non-empty finite set $U \subseteq \mathbb{N} \setminus \{0\}$ we have $\mathsf{LCM}(U) \leq \max(U)^{|U|}$, all of the above constants are asymptotically bounded by $2^{\mathsf{poly}(|\mathcal{C}|)}$.

The main result of this section is the following theorem.

▶ **Theorem 16** (Small Parameter Theorem). *Let $\mathcal{C} = (Q, \{p\}, R, q_{init}, F)$ be a POCA with one parameter. If there exists an accepting N-run in $\mathcal{C}$ with values in $[0, 4N]$ for some $N > M_{\mathcal{C}}$, then there exists an accepting $(N - \Gamma_{\mathcal{C}})$-run in $\mathcal{C}$.*

The Small Parameter Theorem has the following consequence for $(2,1)$-PTA-Reachability.

▶ **Corollary 17.** $(2,1)$-PTA-Reachability *is in* EXPSPACE.

Overview of the proof of the Small Parameter Theorem

The Small Parameter Theorem (Theorem 16) states that, in case N is sufficiently large, accepting N-runs whose configurations have counter values all inside $[0, 4N]$ can be turned into accepting $(N - \Gamma_{\mathcal{C}})$-runs. For its proof we proceed as follows. As mentioned already in Section 5 we prefer to view N-runs as N-semiruns.

Manipulating only N-semiruns, the following Depumping Lemma can turn N-semiruns whose Δ is either sufficiently large (resp. sufficiently small) again into N-semiruns whose Δ is less large (resp. small). It requires however an N-run whose ϕ-projection has a nice bracketing property, namely a ϕ-projection that lies in the regular language Λ_8.

▶ **Lemma 18** (Depumping Lemma). *For all N-semiruns π satisfying $\phi(\pi) \in \Lambda_8$ and $|\Delta(\pi)| > \Upsilon_{\mathcal{C}}$ there exists an N-semirun π' such that either*

- $\Delta(\pi) > \Upsilon_{\mathcal{C}}$ *and* $\Delta(\pi') = \Delta(\pi) - \Gamma_{\mathcal{C}}$*, or*
- $\Delta(\pi) < -\Upsilon_{\mathcal{C}}$ *and* $\Delta(\pi') = \Delta(\pi) + \Gamma_{\mathcal{C}}$.

Moreover, $\pi' = \pi - I_1 - I_2 \ \cdots \ - I_k$ for pairwise disjoint intervals $I_1, \ldots, I_k \subseteq [0, |\pi|]$ such that we have $\phi(\pi[I_i]) \in \Lambda_{16}$ for all $i \in [1, k]$, and either $\Delta(\pi[I_i]) > 0$ for all $i \in [1, k]$ or $\Delta(\pi[I_i]) < 0$ for all $i \in [1, k]$.

Proof. Let $\pi = q_0(z_0) \xrightarrow{\pi_0, N} q_1(z_1) \xrightarrow{\pi_1, N} \quad \cdots \quad \xrightarrow{\pi_{n-1}, N} q_n(z_n)$ be an N-semirun such that $\phi(\pi) \in \Lambda_8$. We will assume without loss of generality that $\Delta(\pi) > \Upsilon_{\mathcal{C}}$. The dual case when $\Delta(\pi) < -\Upsilon_{\mathcal{C}}$ can be proven analogously.
For every position $i \in [0, n]$ let us define

$$\lambda(i) = |\phi(\pi[0,i])|_{[} - |\phi(\pi[0,i])|_{]} \qquad \text{and} \qquad \mathsf{pot}(i) = z_i - z_0 - \lambda(i) \cdot N \qquad .$$

Note that since $\phi(\pi) \in \Lambda_8$ by assumption we have for all $i \in [0, n]$,

$$\lambda(i) \in [-8, 8], \tag{2}$$

and moreover

$$\phi(\pi[0,i]) \in \Lambda_8 \iff \lambda(i) = 0. \tag{3}$$

We note the following important properties of pot,

1. $|\mathsf{pot}(i-1) - \mathsf{pot}(i)| \leq 1$ for all $i \in [1, n]$,
2. $\mathsf{pot}(0) = 0$,
3. for all $0 \leq i < j \leq n$, if $\lambda(i) = \lambda(j)$, then $\mathsf{pot}(j) - \mathsf{pot}(i) = z_j - z_i$, and
4. $\mathsf{pot}(n) = z_n - z_0 = \Delta(\pi)$ since $\lambda(0) = \lambda(n) = 0$.

The following claim states that if in a subsemirun pot increases sufficiently, one can find a subsemirun therein that can potentially be glued.

▷ Claim 19. For each subsemirun $\pi[a, b]$ that satisfies $\mathsf{pot}(b) - \mathsf{pot}(a) > 17 \cdot |Q| \cdot Z_{\mathcal{C}}$ there exist positions $a \leq s < t \leq b$, such that

- $q_s = q_t$,
- $\lambda(s) = \lambda(t)$, and
- $z_t - z_s = dZ_{\mathcal{C}}$ for some $d \in [1, 17 \cdot |Q|]$.

Proof of the Claim. Since by assumption $\mathsf{pot}(b) - \mathsf{pot}(a) > 17 \cdot |Q| \cdot Z_{\mathcal{C}}$, by the pigeonhole principle and Point 1 above, there exist two indices $a \leq s < t \leq b$ such that $q_s = q_t$, $\lambda(s) \in [-8, 8]$ and $\lambda(t) \in [-8, 8]$ are equal, and $\mathsf{pot}(t) - \mathsf{pot}(s) = dZ_{\mathcal{C}}$ for some $d \in [1, 17 \cdot |Q|]$. By Point 3 above, from $\lambda(t) = \lambda(s)$, it follows $z_t - z_s = \mathsf{pot}(t) - \mathsf{pot}(s) = dZ_{\mathcal{C}}$. ◁

Since $\mathsf{pot}(i) - \mathsf{pot}(i-1) \leq 1$ for all $i \in [1, n]$ by Point 1 above and

$$\begin{aligned} \mathsf{pot}(n) - \mathsf{pot}(0) &= z_n - z_0 \\ &= \Delta(\pi) \\ &> \Upsilon_{\mathcal{C}} \\ &\overset{\text{page } 11}{=} 17 \cdot |Q| \cdot \mathsf{LCM}(17 \cdot |Q|) \cdot (17 \cdot |Q| \cdot Z_{\mathcal{C}} + 2)\,, \end{aligned}$$

by the pigeonhole principle, there exist at least

$$17 \cdot |Q| \cdot \mathsf{LCM}(17 \cdot |Q|)$$

pairwise disjoint subsemiruns $\pi[a, b]$ satisfying $\mathsf{pot}(b) - \mathsf{pot}(a) > 17 \cdot |Q| \cdot Z_{\mathcal{C}}$. Let

$$L = \mathsf{LCM}(17 \cdot |Q|),$$

and let $\pi[a_1, b_1], \ldots, \pi[a_{17 \cdot |Q| \cdot L}, b_{17 \cdot |Q| \cdot L}]$ be an enumeration of these latter subsemiruns. We apply the above Claim to all of these $\pi[a_i, b_i]$: there exist positions $a_i \leq s_i \leq t_i \leq b_i$ such that $\lambda(s_i) = \lambda(t_i)$, $q_{s_i} = q_{t_i}$, and $z_{t_i} = z_{s_i} + d_i Z_{\mathcal{C}}$ for some $d_i \in [1, 17 \cdot |Q|]$. From $\lambda(s_i) = \lambda(t_i)$ and (2) it follows $\phi(\pi[s_i, t_i]) \in \Lambda_{16}$. Recall that $\Gamma_{\mathcal{C}} = \mathsf{LCM}(17 \cdot |Q|) \cdot Z_{\mathcal{C}} = L \cdot Z_{\mathcal{C}}$ by definition on page 11. By the pigeonhole principle, among these $17 \cdot |Q| \cdot L$ pairwise disjoint subsemiruns $\pi[a_i, b_i]$, there exists some $d \in [1, 17 \cdot |Q|]$ such that there are L/d many different $\pi[a_i, b_i]$ all satisfying $d_i = d$. Let $\pi[a_{i_1}, b_{i_1}], \ldots, \pi[a_{i_{L/d}}, b_{i_{L/d}}]$ be an enumeration of these latter $\pi[a_i, b_i]$. Note that for all of these $\pi[a_i, b_i]$ we have $\Delta(\pi[s_{i_j}, t_{i_j}]) = d \cdot Z_{\mathcal{C}}$. Since moreover $q_{s_{i_j}} = q_{t_{i_j}}$ we know that, for all $j \in [1, L/d]$, the gluing $\pi - [s_{i_j}, t_{i_j}]$ is an N-semirun with $\Delta(\pi - [s_{i_j}, t_{i_j}]) = \Delta(\pi) - dZ_{\mathcal{C}}$. Thus,

$$\pi' \quad = \quad \pi - [s_{i_1}, t_{i_1}] - \ldots - [s_{i_{L/d}}, t_{i_{L/d}}]$$

is an N-semirun satisfying $\Delta(\pi') = \Delta(\pi) - d \cdot (L/d) \cdot Z_{\mathcal{C}} = \Delta(\pi) - \Gamma_{\mathcal{C}}$ as required. ◀

Assuming N to be sufficiently large the Bracket Lemma shows that for every $(N - \Gamma_{\mathcal{C}})$-semirun whose Δ is again sufficiently large (resp. sufficiently small) and whose ϕ-projection satisfies a majority condition, the existence of a subsemirun whose Δ is again sufficiently large (resp. sufficiently small) but which has a ϕ-projection that lies in the regular language Λ_8. Since the resulting subsemiruns have a desirable ϕ-projection, the Depumping Lemma can be applied to these. Combining these remarks allows us to construct a new semirun whose Δ is slightly smaller (resp. bigger) than the Δ of the original semirun the Bracket Lemma was applied to.

▶ **Lemma 20** (Bracket Lemma). *For all $N > M_{\mathcal{C}}$ and for all $(N - \Gamma_{\mathcal{C}})$-semiruns π satisfying* $\textsc{Values}(\pi) \subseteq [0, 4N]$*, $\Delta(\pi) < -\Upsilon_{\mathcal{C}}$ (resp. $\Delta(\pi) > \Upsilon_{\mathcal{C}}$) and where $\phi(\pi)$ contains at least as many occurrences of $[$ as occurrences of $]$ (resp. at least as many occurrences of $]$ as occurrences of $[$) there exists a subsemirun $\pi[c, d]$ satisfying $\phi(\pi[c, d]) \in \Lambda_8$ and $\Delta(\pi[c, d]) < -\Upsilon_{\mathcal{C}}$ (resp. $\Delta(\pi[c, d]) > \Upsilon_{\mathcal{C}}$).*

Note that trivially, every N-semirun ρ with $\phi(\rho) = \varepsilon$ is already an $(N - \Gamma_{\mathcal{C}})$-semirun. Let us exemplify an interplay between the Bracket Lemma and the Depumping Lemma. For instance, assume we are to turn the following N-semirun into an $(N - \Gamma_{\mathcal{C}})$-semirun with the same source and target configuration, namely an N-semirun of the form $\tau\rho$, where τ is a $+p$-transition (thus a length one N-semirun), $\phi(\rho) = \varepsilon$, and $\Delta(\rho) < -\Upsilon_{\mathcal{C}}$: indeed, firstly one can explicitly turn the $+p$-transition τ into the $+p$-transition $\widehat{\tau}$ with $\Delta(\widehat{\tau}) = N - \Gamma_{\mathcal{C}}$ (thus a length one $(N - \Gamma_{\mathcal{C}})$-semirun) and secondly apply (by observing that ρ is already

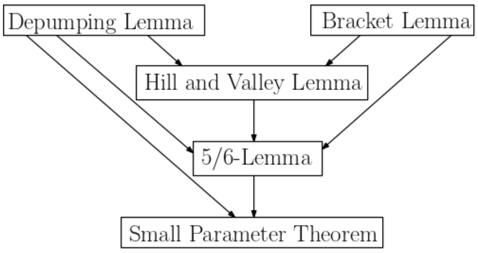


Figure 5 Illustration of the dependencies between the lemmas. The presence of an arrow going from a lemma to another means that the lemma in question is used inside the proof of the lemma the arrow points to.

an $(N - \Gamma_{\mathcal{C}})$-semirun) the Bracket Lemma to ρ. Using the interplay between the Bracket Lemma and the Depumping Lemma one can obtain an $(N - \Gamma_{\mathcal{C}})$-semirun $\widehat{\rho}$ (obtained by gluing and shifting) such that $\widehat{\tau}\,\widehat{\rho}$ is an $(N - \Gamma_{\mathcal{C}})$-semirun with the same source and target configuration as $\tau\rho$.

The following notion of hills and valleys provides a more general class of semiruns to which the above-mentioned reasoning in the previous paragraph can be applied. B-hills are semiruns that start and end in configurations with low counter values but where all intermediate configurations have counter values above these source and target configurations, and where moreover $+p$-transitions (resp. $-p$-transitions) are followed (resp. preceded) by semiruns with counter effect strictly smaller than $-\Upsilon_{\mathcal{C}}$ (resp. strictly larger than $\Upsilon_{\mathcal{C}}$). B-valleys are defined dually.

Formally let $q_0(z_0) \xrightarrow{\pi_0,N} q_1(z_1) \cdots \xrightarrow{\pi_{n-1},N} q_n(z_n)$ be an N-semirun. It is a *B-hill* if $z_0, z_n < B$, $z_i \geq B$ for all $i \in [1, n-1]$, $\pi_i = -p$ implies $z_i > z_0 + \Upsilon_{\mathcal{C}}$ for all $i \in [0, n-1]$, and $\pi_i = +p$ implies $z_{i+1} > z_n + \Upsilon_{\mathcal{C}}$ for all $i \in [0, n-1]$. Dually, it is a *B-valley* if $z_0, z_n > B$, $z_i \leq B$ for all $i \in [1, n-1]$, $\pi_i = -p$ implies $z_{i+1} < z_n - \Upsilon_{\mathcal{C}}$ for all $i \in [0, n-1]$, and $\pi_i = +p$ implies $z_i < z_0 - \Upsilon_{\mathcal{C}}$ for all $i \in [0, n-1]$.

The Hill and Valley Lemma (Lemma 21) allows us to transform N-semiruns that are B-hills (resp. B-valleys) into $(N - \Gamma_{\mathcal{C}})$-semiruns with the same source and target configurations.

▶ **Lemma 21** (Hill and Valley Lemma). *For all $N, B \in \mathbb{N}$ and all N-semiruns π from $q_0(z_0)$ to $q_n(z_n)$ with $N > M_{\mathcal{C}}$ and $\textsc{Values}(\pi) \subseteq [0, 4N]$ such that moreover π is either a B-hill or a B-valley, there exists an $(N - \Gamma_{\mathcal{C}})$-semirun from $q_0(z_0)$ to $q_n(z_n)$ that is both a min-rising and max-falling $(B - \Upsilon_{\mathcal{C}} - \Gamma_{\mathcal{C}} - 1)$-embedding of π (in case π is a B-hill), or both a min-rising and max-falling $(B + \Upsilon_{\mathcal{C}} + \Gamma_{\mathcal{C}} + 1)$-embedding of π (in case π is a B-valley).*

By carefully factorizing N-semiruns with a Δ smaller than $5/6 \cdot N$ into suitably chosen hills and valleys, one can turn them into $(N - \Gamma_{\mathcal{C}})$-semiruns that are moreover ℓ-embeddings for every ℓ that is not far away from the counter values of both the source and target configuration. The following lemma makes this more formal.

▶ **Lemma 22** (5/6-Lemma). *For all $N > M_{\mathcal{C}}$ and all $\ell \in \mathbb{Z}$ and all N-semiruns π from $q_0(z_0)$ to $q_n(z_n)$ with $\textsc{Values}(\pi) \subseteq [0, 4N]$ satisfying $\max(z_0, z_n, \ell) - \min(z_0, z_n, \ell) \leq 5/6 \cdot N$ there exists an $(N - \Gamma_{\mathcal{C}})$-semirun π' from $q_0(z_0)$ to $q_n(z_n)$ that is an ℓ-embedding of π such that $\textsc{Values}(\pi') \subseteq [\min(\pi) - \Gamma_{\mathcal{C}}, \max(\pi) + \Gamma_{\mathcal{C}}]$.*

Figure 5 provides an overview of the dependencies of the above-mentioned lemmas.

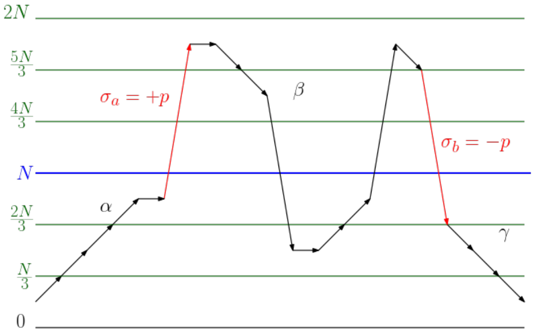


Figure 6 Application of the 5/6-Lemma to the subrun $\sigma[a, b+1]$.

Let us exemplify how the 5/6-Lemma is used in proving the Small Parameter Theorem. For this let us fix some POCA $\mathcal{C}$ over a parameter p, some $N > M_{\mathcal{C}}$ and an accepting N-run π with $\textsc{Values}(\pi) \subseteq [0, 4N]$, where π is of the form $\pi = r_0(x_0) \xrightarrow{\pi_0, N} r_1(x_1) \cdots \xrightarrow{\pi_{n-1}, N} r_n(x_n)$ and where $r_n \in F$. We need to show the existence of an accepting $(N - \Gamma_{\mathcal{C}})$-run. We may assume $x_n = 0$ w.l.o.g. (by simply requiring a final zero test in a new PTA).

Since $\frac{N}{3} < N - \Gamma_{\mathcal{C}}$, by definition of the constants on page 11, every subrun ρ of π with $\textsc{Values}(\rho) \subseteq [0, \frac{N}{3}[$ is already an $(N - \Gamma_{\mathcal{C}})$-run. One can therefore uniquely factorize π as $\pi = \rho^{(0)}\sigma^{(1)}\rho^{(1)} \cdots \sigma^{(m)}\rho^{(m)}$, where each $\rho^{(j)}$ satisfies $\textsc{Values}(\rho^{(j)}) \subseteq [0, \frac{N}{3}[$ and each $\sigma^{(j)}$ is some subrun $\pi[c, d]$ with $x_c < \frac{N}{3}$, $x_d < \frac{N}{3}$ and $x_e \geq \frac{N}{3}$ for all $e \in [c+1, d-1]$, where moreover $[c+1, d-1] \neq \emptyset$.

Thus, it suffices to show that for every N-run $\sigma = q_0(z_0) \cdots \xrightarrow{\sigma_{m-1}, N} q_m(z_m)$ satisfying $\textsc{Values}(\sigma) \subseteq [0, 4N]$, $z_0, z_m < \frac{N}{3}$ and $z_i \geq \frac{N}{3}$ for all $i \in [1, m-1]$, there exists an $(N - \Gamma_{\mathcal{C}})$-run from $q_0(z_0)$ to $q_m(z_m)$. Let us assume $\max(\sigma) \geq N$ (the case $\max(\sigma) < N$ is even easier) and let $a \in [0, m]$ be minimal such that $z_a < N$ and $z_{a+1} \geq N$ and let $b \in [0, m]$ be maximal such that $z_b \geq N$ and $z_{b+1} < N$. That is, one can factorize σ as $\sigma = \alpha\, \sigma[a, a+1]\, \beta\, \sigma[b, b+1]\, \gamma$, where $\alpha = \sigma[0, a]$, $\beta = \sigma[a+1, b]$ and $\gamma = \sigma[b+1, m]$. The situation is depicted in Figure 6.

For $i \in [1, 5]$ let $\mathcal{I}_i = \left\{ z \in [0, 4N] \mid \frac{iN}{3} \leq z < \frac{(i+1)N}{3} \right\}$. Our proof involves a careful case distinction on which of the $\mathcal{I}_i$ the counter values z_a, z_{a+1}, z_b and z_{b+1} lie in, respectively. Let us here only treat the case $z_{a+1}, z_b \in \mathcal{I}_5$; thus σ_a (resp. σ_b) is a $+p$-transition (resp. $-p$-transition) and therefore $z_a, z_{b+1} \in \mathcal{I}_2$. We apply the 5/6-Lemma to the subrun $\sigma[a, b+1]$ for $\ell = N$, hereby obtaining an $(N - \Gamma_{\mathcal{C}})$-semirun $\sigma\widehat{[a, b+1]}$ that is an N-embedding of $\sigma[a, b+1]$ also from $q_a(z_a)$ to $q_{b+1}(z_{b+1})$. It follows from $\textsc{Values}(\sigma\widehat{[a, b+1]}) \subseteq [\min(\sigma[a, b+1]) - \Gamma_{\mathcal{C}}, \max(\sigma[a, b+1]) + \Gamma_{\mathcal{C}}]$ that $\sigma\widehat{[a, b+1]} - \Gamma_{\mathcal{C}}$ is in fact an $(N - \Gamma_{\mathcal{C}})$-run from $q_a(z_a - \Gamma_{\mathcal{C}})$ to $q_{b+1}(z_{b+1} - \Gamma_{\mathcal{C}})$. By definition of a and b it follows that $\phi(\alpha) = \phi(\gamma) = \varepsilon$. Thus, by exploiting that $z_a, z_{b+1} \in \mathcal{I}_2$ one can, by suitably applying the Depumping Lemma (possibly several times), obtain an $(N - \Gamma_{\mathcal{C}})$-run $\widehat{\alpha}$ from $q_0(z_0)$ to $q_a(z_a - \Gamma_{\mathcal{C}})$, and dually an $(N - \Gamma_{\mathcal{C}})$-run $\widehat{\gamma}$ from $q_{b+1}(z_{b+1} - \Gamma_{\mathcal{C}})$ to $q_m(z_m)$. The concatenation of $\widehat{\alpha}$, $\sigma\widehat{[a, b+1]} - \Gamma_{\mathcal{C}}$ and $\widehat{\gamma}$ is the desired $(N - \Gamma_{\mathcal{C}})$-run from $q_0(z_0)$ to $q_m(z_m)$.

7 Conclusion

In this paper we have shown that the reachability problem for parameteric timed automata with two parametric clocks and one parameter is complete for exponential space.

For the lower bound proof, inspired by [13, 15], we made use of two results from complexity theory. First, we made use of a serializability characterization of EXPSPACE from [13] which is a padded version of the serializability characterization of PSPACE from [20], which in turn has its roots in Barrington's Theorem [7]. Second, we made use of a result of Chiu, Davida, Litow that states that numbers in Chinese Remainder Representation can be translated into binary representation in NC^1 (and thus in logarithmic space). We are convinced that it is worthwhile to develop a suitable programming language that serves as a unifying framework in that it provides an interface for proving lower bounds for various problems involving automata. In a sense, we have developed the corresponding interface "by hand" when defining how parametric timed automata can compute functions (Definition 5).

For the EXPSPACE upper bound we first followed the approach of Bundala and Ouaknine [10] by providing an exponential time translation from reachability in parametric timed automata with two parametric clocks and one parameter (i.e. $(2,1)$-PTA) to reachability in parametric one-counter automata (POCA) over one parameter, yet on a slightly less expressive POCA model as introduced in [10]. We then studied the reachability in POCA with one parameter p. A repeated application of our Small Parameter Theorem (Theorem 16) allows to conclude that such a POCA has an accepting N-run all of whose counter values lie in $[0,4N]$ if, and only if, there exists such an accepting N-run for some N that is at most exponential in the size of the POCA. Since the translation from $(2,1)$-PTA to POCA is computable in exponential time, this gives a doubly exponential upper bound on the parameter value of the original $(2,1)$-PTA and hence an EXPSPACE upper bound for $(2,1)$-PTA-Reachability (Corollary 17).

In proving the Small Parameter Theorem we introduced the notion of semiruns and gave several techniques for manipulating them. The Depumping Lemma (Lemma 18) allowed us to construct from semiruns with large absolute counter effect new semiruns with a smaller absolute counter effect. The Bracket Lemma (Lemma 20) allowed us to find in semiruns having a sufficiently large absolute counter effect and satisfying some majority condition on the number of occurrences of $+p$-transitions and $-p$-transitions some subsemirun that has again a large absolute counter effect and moreover some bracketing properties. Our Hill and Valley Lemma (Lemma 21) allowed to turn, for sufficiently large N, any N-semirun that is either a hill or a valley into an N'-semirun for some $N' < N$. Our 5/6-Lemma (Lemma 22) allowed to turn for sufficiently large N any N-semirun with an absolute counter effect of at most $5/6 \cdot N$ into an N'-semirun for some $N' < N$.

We hope that extensions of our techniques provide a line of attack for finally showing decidability (and the precise complexity) of $(2,*)$-PTA-Reachability. For these however, it seems that the reduction to POCA indeed requires the presence of so-called $+[0,p]$-transitions. When analyzing runs in the corresponding more general POCA model that in turn also involves an arbitrary number of parameters, it will become necessary to "de-scale" semiruns in the following sense. Already in the presence of two parameters one can see that it becomes necessary to decrease the value of both parameters simultaneously proportionally: for instance one can build a $(2,2)$-PTA for which reachability holds only if the first parameter is a multiple of the second parameter. How our techniques can be extended to handle such obstacles remains yet to be explored.

References

1 Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking for real-time systems. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90), Philadelphia, Pennsylvania, USA, June 4-7, 1990*, pages 414–425. IEEE Computer Society, 1990. doi:10.1109/LICS.1990.113766.

2 Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

3 Rajeev Alur, Thomas A Henzinger, and Moshe Y Vardi. Parametric real-time reasoning. In *Proc. STOC'93*, pages 592–601. ACM, 1993.

4 Étienne André. What's decidable about parametric timed automata? *International Journal on Software Tools for Technology Transfer*, 21(2):203–219, 2019.

5 Étienne André. What's decidable about parametric timed automata? *Int. J. Softw. Tools Technol. Transf.*, 21(2):203–219, 2019.

6 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambrdige University Press, 2009.

7 D. A. M. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

8 Nikola Benes, Peter Bezdek, Kim Guldstrand Larsen, and Jirí Srba. Language emptiness of continuous-time parametric timed automata. 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*, volume 9135 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2015.

9 Benedikt Bollig, Karin Quaas, and Arnaud Sangnier. The complexity of flat freeze ltl. *Logical Methods in Computer Science*, 15(3):32:1–32:26, 2019. `arXiv:1609.06124`.

10 Daniel Bundala and Joel Ouaknine. On parametric timed automata and one-counter machines. *Information and Computation*, 253:272–303, 2017.

11 Andrew Chiu, George Davida, and Bruce Litow. Division in logspace-uniform NC^1. *Theoretical Informatics and Applications. Informatique Théorique et Applications*, 35(3):259–275, 2001.

12 Stéphane Demri and Arnaud Sangnier. When model-checking freeze LTL over counter machines becomes decidable. In C.-H. Luke Ong, editor, *Foundations of Software Science and Computational Structures, 13th International Conference, FOSSACS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6014 of *Lecture Notes in Computer Science*, pages 176–190. Springer, 2010.

13 Stefan Göller, Christoph Haase, Joël Ouaknine, and James Worrell. Model Checking Succinct and Parametric One-Counter Automata. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 575–586. Springer, 2010.

14 Stefan Göller and Mathieu Hilaire. Reachability in two-parametric timed automata with one parameter is EXPSPACE-complete. *CoRR*, abs/2011.07091, 2020. `arXiv:2011.07091`.

15 Stefan Göller and Markus Lohrey. Branching-time model checking of one-counter processes and timed automata. *SIAM J. Comput.*, 42(3):884–923, 2013.

16 Christoph Haase. *On the complexity of model checking counter automata*. PhD thesis, Oxford University, 2012.

17 Christoph Haase, Joël Ouaknine, and James Worrell. Relating reachability problems in timed and counter automata. *Fundam. Informaticae*, 143(3-4):317–338, 2016.

18 Michael A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, 1978.

19 Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. What good are digital clocks? In Werner Kuich, editor, *Automata, Languages and Programming, 19th International Colloquium, ICALP92, Vienna, Austria, July 13-17, 1992, Proceedings*, volume 623 of *Lecture Notes in Computer Science*, pages 545–558. Springer, 1992.

20 Ulrich Hertrampf, Clemens Lautemann, Thomas Schwentick, Heribert Vollmer, and Klaus W. Wagner. On the power of polynomial time bit-reductions. In *Proceedings of the Eighth Annual Structure in Complexity Theory Conference*, pages 200–207. IEEE Computer Society Press, 1993.

21 Antonia Lechner, Richard Mayr, Joël Ouaknine, Amaury Pouly, and James Worrell. Model checking flat freeze LTL on one-counter automata. *Log. Methods Comput. Sci.*, 14(4), 2018.

22 Joël Ouaknine and James Worrell. Universality and Language Inclusion for Open and Closed Timed Automata. In Oded Maler and Amir Pnueli, editors, *Hybrid Systems: Computation and Control, 6th International Workshop, HSCC 2003 Prague, Czech Republic, April 3-5, 2003, Proceedings*, volume 2623 of *Lecture Notes in Computer Science*, pages 375–388. Springer, 2003. doi:10.1007/3-540-36580-X_28.

23 C. H. Papadimitriou. *Computational Complexity.* Addison Wesley, 1994.

Refined Notions of Parameterized Enumeration Kernels with Applications to Matching Cut Enumeration

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

Christian Komusiewicz
Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, Germany

Dieter Kratsch
LGIMP, Université de Lorraine, Metz, France

Van Bang Le
Institut für Informatik, Universität Rostock, Germany

Abstract

An enumeration kernel as defined by Creignou et al. [Theory Comput. Syst. 2017] for a parameterized enumeration problem consists of an algorithm that transforms each instance into one whose size is bounded by the parameter plus a solution-lifting algorithm that efficiently enumerates all solutions from the set of the solutions of the kernel. We propose to consider two new versions of enumeration kernels by asking that the solutions of the original instance can be enumerated in polynomial time or with polynomial delay from the kernel solutions. Using the NP-hard Matching Cut problem parameterized by structural parameters such as the vertex cover number or the cyclomatic number of the input graph, we show that the new enumeration kernels present a useful notion of data reduction for enumeration problems which allows to compactly represent the set of feasible solutions.

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

Keywords and phrases enumeration problems, polynomial delay, output-sensitive algorithms, kernelization, structural parameterizations, matching cuts

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.37

Related Version *Full Version*: `https://arxiv.org/abs/2101.03800`

Funding *Petr A. Golovach*: Supported by the Research Council of Norway via the project "MULTIVAL" (grant no. 263317).

Acknowledgements We dedicate this paper to the memory of our coauthor and friend Dieter Kratsch who recently passed away. Without Dieter, this paper would have never been written.

1 Introduction

The enumeration of all feasible solutions of a computational problem is a fundamental task in computer science. For the majority of enumeration problems, the number of feasible solutions can be exponential in the input size in the worst-case. The running time of enumeration algorithms is thus measured not only in terms of the input size n but also in terms of the output size. The two most-widely used definitions of efficient algorithms are polynomial output-sensitive algorithms where the running time is polynomial in terms of input and output size and polynomial-delay algorithms, where the algorithm spends only a polynomial running time between the output of consecutive solutions. Since in some enumeration problems, even the problem of deciding the existence of one solution is not solvable in polynomial time, it

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 37; pp. 37:1–37:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

was proposed to allow FPT algorithms that have running time or delay $f(k) \cdot n^{O(1)}$ for some problem-specific parameter k [9, 11, 12, 14, 31]. Naturally, FPT-enumeration algorithms are based on extensions of standard techniques in FPT algorithms such as bounded-depth search trees [11, 12, 14] or color coding [31].

An important technique for obtaining FPT algorithms for decision problems is *kernelization* [10, 15, 28], where the idea is to shrink the input instance in polynomial time to an equivalent instance whose size depends only on the parameter k. In fact, a parameterized problem admits an FPT algorithm if and only if it admits a kernelization. It seems particularly intriguing to use kernelization for enumeration problems as a small kernel can be seen as a compact representation of the set of feasible solutions. The first notion of kernelization in the context of enumeration problems were the *full kernels* defined by Damaschke [11]. Informally, a full kernel for an instance of an enumeration problem is a subinstance that contains all minimal solutions of size at most k. This definition is somewhat restrictive since it is tied to subset minimization problems parameterized by the solution size parameter k. Nevertheless, full kernels have been obtained for some problems [12, 16, 26, 35].

To overcome the restrictions of full kernels, Creignou et al. [9] proposed *enumeration kernels*. Informally, an enumeration kernel for a parameterized enumeration problem is an algorithm that replaces the input instance by one whose size is bounded by the parameter and which has the property that the solutions of the original instance can be computed by listing the solutions of the kernel and using an efficient *solution-lifting algorithm* that outputs for each solution of the kernel a set of solutions of the original instance. In the definition of Creignou et al. [9], the solution-lifting algorithm may be an FPT-delay algorithm, that is, an algorithm with $f(k) \cdot n^{O(1)}$ delay where n is the overall input size. We find that this time bound is too weak, because it essentially implies that every enumeration problem that can be solved with FPT-delay admits an enumeration kernel of *constant* size. Essentially, this means that the solution-lifting algorithm is so powerful that it can enumerate all solutions while ignoring the kernel. Motivated by this observation and the view of kernels as compact representations of the solution set, we modify the original definition of enumeration kernels [9].

Our results. We present two new notions of efficient enumeration kernels by replacing the demand for FPT-delay algorithms by a demand for polynomial-time enumeration algorithms or polynomial-delay algorithms, respectively. We call the two resulting notions of enumeration kernelization *fully-polynomial enumeration kernels* and *polynomial-delay enumeration kernels*. Our paper aims at showing that these two new definitions present a sweet spot between the notion of full kernels, which is too strict for some applications, and enumeration kernels, which are too lenient in some sense. We first show that the two new definitions capture the class of efficiently enumerable problems in the sense that a problem has a fully-polynomial (a polynomial-delay) enumeration kernel if and only if it has an FPT-enumeration algorithm (an FPT-delay enumeration algorithm). Moreover, the kernels have constant size if and only if the problems have polynomial-time (polynomial-delay) enumeration algorithms. Thus, the new definitions correspond to the case of problem kernels for decision problems, which are in FPT if and only if they have kernels and which can be solved in polynomial time if and only if they have kernels of constant size (see, e.g. [10, Chapter 2] or [15, Chapter 1]).

We then apply both types of kernelizations to the enumeration of matching cuts. A matching cut of a graph G is the set of edges $M = E(A, B)$ for a partition $\{A, B\}$ of $V(G)$ forming a matching. We investigate the problems of enumerating all minimal, all maximal, or all matching cuts of a graph. We refer to these problems as Enum Minimal MC, Enum Maximal MC, and Enum MC, respectively. These matching cut problems constitute a very suitable study case for enumeration kernels, since it is NP-hard to decide whether a graph has a matching cut [7] and therefore, they do not admit polynomial output-sensitive algorithms. We consider all three problems with respect to structural parameterizations such as the vertex cover number, the modular width, or the cyclomatic number of the input graph. The choice of these parameters is motivated by the fact that neither problem admits an enumeration kernel of polynomial size for the more general structural parameterizations by the treewidth or cliquewidth up to some natural complexity assumptions (see Proposition 5). Table 1 summarizes the results. Due to space constraints some results and proofs are either omitted or just sketched. We refer to the full version [20] for the details.

Table 1 An overview of our results. Herein, "kernel" means fully-polynomial enumeration kernel, "del-kernel" means polynomial-delay enumeration kernel and "bi" means bijective enumeration kernel (a slight generalization of full kernels), a ($\star$) means that the lower bound assumes NP $\not\subseteq$ coNP/ poly, "?" means open status. We use ($*$) for statements whose proofs are omitted in this extended abstract (see [20] for the proofs). The cyclomatic number is also known as the feedback edge number.

Parameter k	Enum MC	Enum Minimal MC	Enum Maximal MC
treewidth & cliquewidth	No poly-size del-kernel ($\star$) (Prop. 5)	No poly-size del-kernel ($\star$) (Prop. 5)	No poly-size del-kernel ($\star$) (Prop. 5)
vertex cover & twin-cover number	size-$\mathcal{O}(k^2)$ del-kernel (Theorems 11 & 12) No kernel	size-$\mathcal{O}(k^2)$ kernel (Theorems 11 & 12)	size-$\mathcal{O}(k^2)$ del-kernel (Theorems 11 & 12) No kernel
neighborhood diversity	size-$\mathcal{O}(k)$ del-kernel ($*$) No kernel ($*$)	size-$\mathcal{O}(k)$ kernel ($*$)	size-$\mathcal{O}(k)$ del-kernel ($*$) No kernel ($*$)
modular width	?	$\mathcal{O}(k)$-kernel ($*$)	?
cyclomatic number	size-$\mathcal{O}(k)$ del-kernel (Theorem 13) No kernel	size-$\mathcal{O}(k)$ del-kernel (Theorem 13)	?
clique partition number	size-$\mathcal{O}(k^3)$ bi kernel ($*$)	size-$\mathcal{O}(k^3)$ bi kernel ($*$)	size-$\mathcal{O}(k^3)$ bi kernel ($*$)

To discuss some of our results and their implication for enumeration kernels in general more precisely, consider Enum MC, Enum Minimal MC, and Enum Maximal MC parameterized by the vertex cover number. We show that Enum Minimal MC admits a fully-polynomial enumeration kernel of polynomial size. As it can be seen that the problem has no full kernel, we obtain that there are natural enumeration problems with a fully-polynomial enumeration kernel that have no full kernel (not even one of super-polynomial size). Then, we show that Enum MC and Enum Maximal MC admit polynomial-delay enumeration kernels but have no fully-polynomial enumeration kernels. Thus, there are natural enumeration problems with polynomial-delay enumeration kernels that do not admit fully-polynomial enumeration kernels (not even one of super-polynomial size).

We also prove a tight upper bound $F(n+1)-1$ for the maximum number of matching cuts of an n-vertex graph, where $F(n)$ is the n-th Fibonacci number and show that all matching cuts can be enumerated in $\mathcal{O}^*(F(n)) = \mathcal{O}^*(1.6181^n)$ time (Theorem 6).

Related work. The current-best exact decision algorithm for Matching Cut, the problem of deciding whether a given graph G has a matching cut, has a running time of $O(1.328^n)$ where n is the number of vertices in G [25]. Faster exact algorithms can be obtained for the case when the minimum degree is large [23]. Matching Cut has FPT-algorithms for the maximum cut size k [21], the vertex cover number of G [27], and weaker parameters such as the twin-cover number [1] or the cluster vertex deletion number [25].

For an overview of enumeration algorithms, refer to the survey of Wasa [37]. A broader discussion of parameterized enumeration is given by Meier [32]. A different extension of enumeration kernels are *advice enumeration kernels* [2]. In these kernels, the solution-lifting algorithm does not need the whole input but only a possibly smaller advice. A further loosely connected extension of standard kernelization are *lossy kernels* which are used for optimization problems [29]; the common thread is that both definitions use a solution-lifting algorithm for recovering solutions of the original instance.

Graph notation. All graphs considered in this paper are finite undirected graphs without loops or multiple edges. We follow the standard graph-theoretic notation and terminology and refer to the book of Diestel [13] for basic definitions. For each of the graph problems considered in this paper, we let $n = |V(G)|$ and $m = |E(G)|$ denote the number of vertices and edges, respectively, of the input graph G if it does not create confusion. For a graph G and a subset $X \subseteq V(G)$ of vertices, we write $G[X]$ to denote the subgraph of G induced by X. For a set of vertices X, $G - X$ denotes the graph obtained by deleting the vertices of X, that is, $G - X = G[V(G) \setminus X]$; for a vertex v, we write $G - v$ instead of $G - \{v\}$. Similarly, for a set of edges A (an edge e, respectively), $G - A$ ($G - e$, respectively) denotes the graph obtained by the deletion of the edges of A (the edge e, respectively). For a vertex v, we denote by $N_G(v)$ the *(open) neighborhood* of v, i.e., the set of vertices that are adjacent to v in G. We use $N_G[v]$ to denote the *closed neighborhood* $N_G(v) \cup \{v\}$ of v. For $X \subseteq V(G)$, $N_G[X] = \bigcup_{v \in X} N_G[v]$ and $N_G(X) = N_G[X] \setminus X$. For disjoint sets of vertices A and B of a graph G, $E_G(A, B) = \{uv \mid u \in A,\ v \in B\}$. We may omit subscripts in the above notation if it does not create confusion. We use P_n, C_n, and K_n to denote the n-vertex path, cycle, and complete graph, respectively. We write $G + H$ to denote the *disjoint union* of G and H, and we use kG to denote the disjoint union of k copies of G.

In a graph G, a *cut* is a partition $\{A, B\}$ of $V(G)$, and we say that $E_G(A, B)$ is an *edge cut*. A *matching* is an edge set in which no two of the edges have a common end-vertex; note that we allow empty matchings. A *matching cut* is a (possibly empty) edge set being an edge cut and a matching. We underline that by our definition, a matching cut is a set of edges, as sometimes in the literature (see, e.g., [7, 22]) a matching cut is defined as a partition $\{A, B\}$ of the vertex set such that $E(A, B)$ is a matching. While the two variants of the definitions are equivalent, say when the decision variant of the matching cut problem is considered, this is not the case in enumeration and counting when we deal with disconnected graphs. For example, the empty graph on n vertices has $2^{n-1} - 1$ partitions $\{A, B\}$ which all correspond to exactly one matching cut in the sense of our definition, namely $M = \emptyset$. A matching cut M of G is *(inclusion) minimal* (*maximal*, respectively) if G has no matching cut $M' \subset M$ ($M' \supset M$, respectively). Notice that a disconnected graph has exactly one minimal matching cut which is the empty set.

2 Parameterized Enumeration and Enumeration Kernels

We use the framework for parameterized enumeration proposed by Creignou et al. [9]. An *enumeration problem* (over a finite alphabet Σ) is a tuple $\Pi = (L, \mathsf{Sol})$ such that

(i) $L \subseteq \Sigma^*$ is a decidable language,

(ii) $\mathsf{Sol}\colon \Sigma^* \to \mathcal{P}(\Sigma^*)$ is a computable function such that for every $x \in \Sigma^*$, $\mathsf{Sol}(x)$ is a finite set and $\mathsf{Sol}(x) \neq \emptyset$ if and only if $x \in L$.

Here, $\mathcal{P}(A)$ is used to denote the powerset of a set A. A string $x \in \Sigma^*$ is an *instance*, and $\mathsf{Sol}(x)$ is the set of solutions to instance x. A *parameterized enumeration problem* is defined as a triple $\Pi = (L, \mathsf{Sol}, \kappa)$ such that (L, Sol) satisfy (i) and (ii) of the above definition, and

(iii) $\kappa\colon \Sigma^* \to \mathbb{N}$ is a parameterization.

We say that $k = \kappa(x)$ is a *parameter*. We define the parameterization as a function of an instance but it is standard to assume that the value of $\kappa(x)$ is either simply given in x or can be computed in polynomial time from x. We follow this convention throughout the paper.

An *enumeration algorithm* $\mathcal{A}$ for a parameterized enumeration problem Π is a deterministic algorithm that for every instance x, outputs exactly the elements of $\mathsf{Sol}(x)$ without duplicates, and terminates after a finite number of steps on every instance. The algorithm $\mathcal{A}$ is an FPT *enumeration* algorithm if it outputs all solutions in at most $f(\kappa(x))p(|x|)$ steps for a computable function $f(\cdot)$ that depends only on the parameter and a polynomial $p(\cdot)$.

We also consider output-sensitive enumerations, and for this, we define delays. Let $\mathcal{A}$ be an enumeration algorithm for Π. For $x \in L$ and $1 \leq i \leq |\mathsf{Sol}(x)|$, the *i-th delay* of $\mathcal{A}$ is the time between outputting the i-th and $(i+1)$-th solutions in $\mathsf{Sol}(x)$. The 0-th delay is the *precalculation* time which is the time from the start of the computation until the output of the fist solution, and the $|\mathsf{Sol}(x)|$-th delay is the *postcalculation* time which is the time after the last output and the termination of $\mathcal{A}$ (if $\mathsf{Sol}(x) = \emptyset$, then the precalculation and postcalculation times are the same). It is said that $\mathcal{A}$ is a *polynomial-delay* algorithm, if all the delays are upper-bounded by $p(|x|)$ for a polynomial $p(\cdot)$. For a parameterized enumeration problem Π, $\mathcal{A}$ is an FPT-*delay algorithm*, if the delays are at most $f(\kappa(x))p(|x|)$, where $f(\cdot)$ is a computable function and $p(\cdot)$ is a polynomial. Notice that every FPT enumeration algorithm $\mathcal{A}$ is also an FPT delay algorithm.

The key definition for us is the generalization of the standard notion of a kernel in Parameterized Complexity (see, e.g, [15]) for enumeration problems.

▶ **Definition 1.** *Let $\Pi = (L, \mathsf{Sol}, \kappa)$ be a parameterized enumeration problem. A* fully-polynomial enumeration kernel(ization) *for Π is a pair of algorithms $\mathcal{A}$ and $\mathcal{A}'$ with the following properties:*

(i) *For every instance x of Π, $\mathcal{A}$ computes in time polynomial in $|x| + \kappa(x)$ an instance y of Π such that $|y| + \kappa(y) \leq f(\kappa(x))$ for a computable function $f(\cdot)$.*

(ii) *For every $s \in \mathsf{Sol}(y)$, $\mathcal{A}'$ computes in time polynomial in $|x| + |y| + \kappa(x) + \kappa(y)$ a nonempty set of solutions $S_s \subseteq \mathsf{Sol}(x)$ such that $\{S_s \mid s \in \mathsf{Sol}(y)\}$ is a partition of $\mathsf{Sol}(x)$.*

Notice that by (ii), $x \in L$ if and only if $y \in L$.

We say that $\mathcal{A}$ is a *kernelization* algorithm and $\mathcal{A}'$ is a *solution-lifting* algorithm. Informally, a solution-lifting algorithm takes as its input a solution for a “small” instance constructed by the kernelization algorithm and, having an access to the original input instance, outputs polynomially many solutions for the original instance, and by going over all the solutions to the small instance, we can generate all the solutions of the original instance without repetitions. We say that an enumeration kernel is *bijective* if $\mathcal{A}'$ produces a unique solution to x, that is, it establishes a bijection between $\mathsf{Sol}(y)$ and $\mathsf{Sol}(x)$, that is, the compressed instance essentially has the same solutions as the input instance. In particular, full kernels [11] are the special case of bijective kernels where $\mathcal{A}'$ is the identity. As it is standard, $f(\cdot)$ is the *size* of a kernel, and the kernel has *polynomial size* if $f(\cdot)$ is a polynomial.

We define *polynomial-delay enumeration kernel(ization)* in a similar way. The only difference is that (ii) is replaced by the condition

(ii*) For every $s \in \mathsf{Sol}(y)$, $\mathcal{A}'$ computes with delay polynomial in $|x| + |y| + \kappa(x) + \kappa(y)$ a set of solutions $S_s \subseteq \mathsf{Sol}(x)$ such that $\{S_s \mid s \in \mathsf{Sol}(y)\}$ is a partition of $\mathsf{Sol}(x)$.

It is straightforward to make the following observation.

▶ **Observation 2.** *Every bijective enumeration kernel is a fully-polynomial enumeration kernel; every fully-polynomial enumeration kernel is a polynomial-delay enumeration kernel.*

Notice also that our definition of polynomial-delay enumeration kernel is different from the definition given by Creignou et al. [9]. In their definition, Creignou et al. [9] require that the solution-lifting algorithm $\mathcal{A}'$ should list all the solutions in S_s with FPT delay for the parameter $\kappa(x)$. We believe that this condition is too weak. In particular, with this requirement, every parameterized enumeration problem, that has an FPT enumeration algorithm $\mathcal{A}^*$ and such that the existence of at least one solution can be verified in polynomial time, has a trivial kernel of constant size. The kernelization algorithm can output any instance satisfying (i) and then we can use $\mathcal{A}^*$ as a solution-lifting algorithm that essentially ignores the output of the kernelization algorithm. Note that for enumeration problems, we typically face the situation where the existence of at least one solution is not an issue. We argue that our definitions are natural by showing the following theorem.

▶ **Theorem 3.** *A parameterized enumeration problem* Π *has an* FPT *enumeration algorithm (an* FPT *delay algorithm) if and only if* Π *admits a fully-polynomial enumeration kernel (polynomial-delay enumeration kernel). Moreover,* Π *can be solved in polynomial time (with polynomial delay) if and only if* Π *admits a fully-polynomial enumeration kernel (a polynomial-delay enumeration kernel) of constant size.*

Proof. The proof of the first claim is similar to the standard arguments for showing the equivalence between fixed-parameter tractability and the existence of a kernel (see, e.g. [10, Chapter 2] or [15, Chapter 1]). However dealing with enumeration problems requires some specific arguments. Let $\Pi = (L, \mathsf{Sol}, \kappa)$ be a parameterized enumeration problem.

In the forward direction, the claim is trivial. Recall that L is decidable and $\mathsf{Sol}(\cdot)$ is a computable function by the definition. If Π admits a fully-polynomial enumeration kernel (a polynomial-delay enumeration kernel respectively), then we apply an arbitrary enumeration algorithm, which is known to exist since $\mathsf{Sol}(\cdot)$ is computable, to the instance y produced by the kernelization algorithm. Then, for each $s \in \mathsf{Sol}(y)$, use the solution-lifting algorithm to list the solutions to the input instance.

For the opposite direction, assume that Π can be solved in $f(\kappa(x)) \cdot |x|^c$ time (with $f(\kappa(x)) \cdot |x|^c$ delay, respectively) for an instance x, where $f(\cdot)$ is a computable function and c is a positive constant. Since $f(\cdot)$ is computable, we assume that we have an algorithm $\mathcal{F}$ computing $f(k)$ in $g(k)$ time. We define $h(k) = \max\{f(k), g(k)\}$.

We say that an instance x of Π is a *trivial no-instance* if x is an instance of minimum size with $\mathsf{Sol}(x) = \emptyset$. We call x a *minimum yes-instance* if x is an instance of minimum size that has a solution. Notice that if Π has instances without solutions, then the size of a trivial no-instance is a constant that depends on Π only and such an instance can be computed in constant time. Similarly, if the problem has instances with solutions, then the size of a minimum yes-instance is constant and such an instance can be computed in constant time. We say that x is a *trivial yes-instance* if x is an instance with minimum size of $\mathsf{Sol}(x)$ that, subject to the first condition, has minimum size. Clearly, the size of a trivial yes-instance is a constant that depends only on Π. However, we may be unable to compute a trivial yes-instance.

Let x be an instance of Π and $k = \kappa(x)$. We run the algorithm $\mathcal{F}$ to compute $f(k)$ for at most $n = |x|$ steps. If the algorithm failed to compute $f(k)$ in n steps, we conclude that $g(k) \geq n$. In this case, the kernelization algorithm outputs x. Then the solution-lifting algorithm just trivially outputs its input solutions. Notice that $|x| \leq g(k) \leq h(k)$ in this case. Assume from now that $\mathcal{F}$ computed $f(k)$ in at most n steps.

If $|x| \leq f(k)$, then the kernelization algorithm outputs the original instance x, and the solution-lifting algorithm trivially outputs its input solutions. Note that $|x| \leq f(k) \leq h(k)$.

Finally, we suppose that $f(k) < |x|$. Observe that the enumeration algorithm runs in $|x|^{c+1}$ time (with $|x|^{c+1}$ delay, respectively) in this case, that is, the running time is polynomial. We use the enumeration algorithm to verify whether x has a solution. For this, notice that a polynomial-delay algorithm can be used to solve the decision problem; we just run it until it outputs a first solution (or reports that there are no solutions). If x has no solution, then Π has a trivial no-instance and the kernelization algorithm computes and outputs it. If x has a solution, then the kernelization algorithm computes a minimum yes-instance y in constant time. We use the enumeration algorithm to check whether $|\mathsf{Sol}(y)| \leq |\mathsf{Sol}(x)|$. If this holds, then we set $z = y$. Otherwise, if $|\mathsf{Sol}(x)| < |\mathsf{Sol}(y)|$, we find an instance z of minimum size such that $|\mathsf{Sol}(z)| \leq |\mathsf{Sol}(x)|$. Notice that this can be done in constant time, because the size of z is upper-bounded by the size of a trivial yes-instance. Then we list the solutions of z in constant time and order them. For the i-th solution of z, the solution-lifting algorithm outputs the i-th solution of x produced by the enumeration algorithm, and for the last solution of z, the solution-lifting algorithm further runs the enumeration algorithm to output the remaining solutions. Since $|\mathsf{Sol}(z)| \leq |\mathsf{Sol}(x)|$, the solution-lifting algorithm outputs a nonempty set of solutions for x for every solution of z.

It is easy to see that we obtain a fully-polynomial enumeration kernel of size $\mathcal{O}(h(\kappa(x))$ (a polynomial-delay enumeration kernel, respectively).

For the second claim, the arguments are the same. If a problem admits a fully-polynomial (a polynomial-delay) enumeration kernel of constant size, then the solutions of the original instance can be listed in polynomial time (or with polynomial delay, respectively) by the solution-lifting algorithm called for the constant number of the solutions of the kernel. Conversely, if a problem can be solved in polynomial time (with polynomial delay, respectively), we can apply the above arguments assuming that $f(k)$ (and, therefore, $g(k)$) is a constant. ◀

In our paper, we consider structural parameterizations of Enum Minimal MC, Enum Maximal MC, and Enum MC by several graph parameters, and the majority of these parameterizations are stronger than the parameterization either by the *treewidth* or the *cliquewidth* of the input graph. Defining the treewidth (denoted by $\mathsf{tw}(G)$) and cliquewidth (denoted by $\mathsf{cw}(G)$) goes beyond of the scope of the current paper and we refer to [8] (see also, e.g., [10]). By the celebrated result of Bodlaender [3] (see also [10]), it is FPT in t to decide whether $\mathsf{tw}(G) \leq t$ and to construct the corresponding tree-decomposition. No such algorithm is known for cliquewidth. However, for algorithmic purposes, it is usually sufficient to use the approximation algorithm of Oum and Seymour [34] (see also [33, 10]). Observe that the property that a set of edges M of a graph G is a matching cut of G can be expressed in *monadic second-order logic* (MSOL); we refer to [8, 10] for the definition of MSOL on graphs. Then the matching cuts (the minimal or maximal matching cuts) of a graph of treewidth at most t can be enumerated with FPT delay with respect to the parameter t by the celebrated meta theorem of Courcelle [8]. The same holds for the weaker parameterization by the cliquewidth of the input graph, because we can use MSOL formulas without quantifications over (sets of) edges: For a graph G, we pick a vertex in each connected component of G and label it. Let R be the set of labeled vertices. Then the enumeration of nonempty matching cuts is equivalent to the enumeration of all partitions $\{A, B\}$ of $V(G)$ such that (i) $R \subseteq A$ and (ii) $E(A, B)$ is a matching. Notice that condition (ii) can be written as follows: for every $u_1, u_2 \in A$ and $v_1, v_2 \in B$, if u_1 is adjacent to v_1 and u_2 is adjacent to v_2, then either $u_1 = u_2$ and $v_1 = v_2$ or $u_1 \neq u_2$ and $v_1 \neq v_2$. Since the empty matching cut can be listed separately if it exists, we obtain that we can use MSOL formulations of the enumeration problems, where only quantifications over vertices and sets of vertices are used. Then the result of Courcelle [8] implies that Enum Minimal MC, Enum Maximal MC, and Enum MC can be solved with FPT delay when parameterized by the cliquewidth of the input graph.

We summarize these observations in the following proposition.

▶ **Proposition 4.** *Enum MC, Enum Minimal MC, and Enum Maximal MC on graphs of treewidth (cliquewidth) at most t can be solved with* FPT *delay when parameterized by t.*

This proposition implies that Enum MC, Enum Minimal MC and Enum Maximal MC can be solved with FPT delay for all structural parameters whose values can be bounded from below by an increasing function of treewidth or cliquewidth. However, we are mainly interested in fully-polynomial or polynomial-delay enumeration kernelization. We conclude this section by pointing out that it is unlikely that Enum Minimal MC, Enum Maximal MC, and Enum MC admit polynomial-delay enumeration kernels of polynomial size for the treewidth or cliquewidth parameterizations. It was pointed out by Komusiewicz, Kratsch, and Le [25] that the decision version of the matching cut problem (that is, the problem asking whether a given graph G has a matching cut) does not admit a polynomial kernel when parameterized by the treewidth of the input graph unless $\mathrm{NP} \subseteq \mathrm{coNP}/\mathrm{poly}$. By the definition of a polynomial-delay enumeration kernel, this gives the following statement.

▶ **Proposition 5.** *Enum Minimal MC, Enum Maximal MC and Enum MC do not admit polynomial-delay enumeration kernels of polynomial size when parameterized by the treewidth (cliquewidth, respectively) of the input graph unless $\mathrm{NP} \subseteq \mathrm{coNP}/\mathrm{poly}$.*

3 A Tight Upper Bound for the Maximum Number of Matching Cuts

In this section we provide a tight upper bound for the maximum number of matching cuts of an n-vertex graph. We complement this result by giving an exact enumeration algorithm for (minimal, maximal) matching cuts. Finally, we give some lower bounds for the maximum number of minimal and maximal matching cuts, respectively. Throughout this section, we use $\#_{\mathrm{mc}}(G)$ to denote the number of matching cuts of a graph G.

To give the upper bound, we use the classical Fibonacci numbers. For a positive integer n, we denote by $F(n)$ the n-th Fibonacci number. Recall that $F(1) = F(2) = 1$, and for $n \geq 3$, the Fibonacci numbers satisfy the recurrence $F(n) = F(n-1) + F(n-2)$. Recall also that the n-th Fibonacci number can be expressed by the following closed formula:

$$F(n) = \frac{1}{\sqrt{5}}\Big(\Big(\frac{1+\sqrt{5}}{2}\Big)^n + \Big(\frac{1-\sqrt{5}}{2}\Big)^n\Big)$$

for every $n \geq 1$. In particular, $F(n) = \mathcal{O}(1.6181^n)$.

▶ **Theorem 6** $(*)$. [1] *An n-vertex graph has at most $F(n+1)-1$ matching cuts. The bound is tight and is achieved for paths. Moreover, if $n \geq 5$, then an n-vertex graph G has $F(n+1)-1$ matching cuts if and only if G is a path. Furthermore, the matching cuts can be enumerated in $\mathcal{O}^*(F(n))$ time.*

Let us remark that if $n \leq 4$, then besides paths P_n, the graphs $K_p + K_q$ for $1 \leq p, q \leq 2$ such that $n = p + q$ have $F(n+1) - 1$ matching cuts.

Clearly, the upper bound for the maximum number of matching cuts given in Theorem 6 is an upper bound for the maximum number of minimal and maximal matching cuts. However, the number of minimal or maximal matching cuts may be significantly less than the number of all matching cuts. We conclude this section by stating the best lower bounds we know for the maximum number of maximal matching cuts and minimal matching cuts, respectively.

[1] The proofs of the statements labeled $(*)$ are omitted in this extended abstract.

▶ **Proposition 7** (∗). *The graph $G = kC_7$ with $n = 7k$ vertices has $14^k = 14^{n/7} \geq 1.4579^n$ maximal matching cuts.*

To achieve a lower bound for the maximum number of minimal matching cuts, we consider the graphs H_k constructed as follows for a positive integer k.

- For every $i \in \{1, \ldots, k\}$, construct two vertices u_i and v_i and a (u_i, v_i)-path of length 4.
- Make the vertices $u_1, \ldots, u_k$ pairwise adjacent, and do the same for $v_1, \ldots, v_k$.

▶ **Proposition 8** (∗). *The number of minimal matching cuts of H_k with $n = 5k$ vertices is at least $4^k = 4^{n/5} \geq 1.3195^n$.*

4 Enumeration Kernels for the Vertex Cover Number Parameterization

In this section, we consider the parameterization of the matching cut problems by the vertex cover number of the input graph. Notice that this parameterization is one of the most thoroughly investigated with respect to classical kernelization (see, e.g., the recent paper of Bougeret, Jansen, and Sau [6] for the currently most general results of this type). However, we are interested in enumeration kernels.

Recall that a set of vertices $X \subseteq V(G)$ is a *vertex cover* of G if for every edge $uv \in E(G)$, at least one of its end-vertices is in X, that is, $V(G) \setminus X$ is an independent set. The *vertex cover number* of G, denoted by $\tau(G)$, is the minimum size of a vertex cover of G. Computing $\tau(G)$ is NP-hard but one can find a 2-approximation by taking the end-vertices of a maximal matching of G [19] (see also [24] for a better approximation) and this suffices for our purposes. Throughout this section, we assume that the parameter $k = \tau(G)$ is given together with the input graph. Note that for every graph G, $\mathsf{tw}(G) \leq \tau(G)$. Therefore, Enum MC, Enum Minimal MC, and Enum Maximal MC can be solved with FPT delay when parameterized by the vertex cover number by Proposition 4.

First, we describe the basic kernelization algorithm that is exploited for all the kernels in this subsection. Let G be a graph that has a vertex cover of size k. The case when G has no edges is trivial and will be considered separately. Assume from now that G has at least one edge and $k \geq 1$.

We use the above-mentioned 2-approximation algorithm to find a vertex cover X of size at most $2k$. Let $I = V(G) \setminus X$. Recall that I is an independent set. Denote by I_0, I_1, and $I_{\geq 2}$ the subsets of vertices of I of degree 0, 1, and at least 2, respectively. We use the following *marking* procedure to label some vertices of I.

(i) *Mark* an arbitrary vertex of I_0 (if it exists).

(ii) For every $x \in X$, *mark* an arbitrary vertex of $N_G(x) \cap I_1$ (if it exists).

(iii) For every two distinct vertices $x, y \in X$, select an arbitrary set of $\min\{3, |(N_G(x) \cap N_G(y)) \cap I_{\geq 2}|\}$ vertices in $I_{\geq 2}$ that are adjacent to both x and y, and *mark* them for the pair $\{x, y\}$.

Note that a vertex of $I_{\geq 2}$ can be marked for distinct pairs of vertices of X. Denote by Z the set of marked vertices of I. Clearly, $|Z| \leq 1 + |X| + 3\binom{|X|}{2}$. We define $H = G[X \cup Z]$. Notice that $|V(H)| \leq |X| + |Z| \leq 1 + 2|X| + 3\binom{|X|}{2} \leq 6k^2 + k + 1$. This completes the description of the basic kernelization algorithm that returns H. It is straightforward to see that H can be constructed in polynomial time.

It is easy to see that H does not keep the information about all matching cuts in G due to the deleted vertices. However, the crucial property is that H keeps all matching cuts of $G' = G - (I_0 \cup I_1)$. Formally, we define $H' = H - (I_0 \cup I_1)$ and show the following lemma.

▶ **Lemma 9** (∗). *A set of edges $M \subseteq E(G')$ is a matching cut of G' if and only if $M \subseteq E(H')$ and M is a matching cut of H'.*

To see the relations between matching cuts of G and H, we define a special equivalence relation for the subsets of edges of G. For a vertex $x \in X$, let $L_x = \{xy \in E(G) \mid y \in I_1\}$, that is, L_x is the set of pendant edges of G with exactly one end-vertex in the vertex cover. Observe that if $L_x \neq \emptyset$, then there is $\ell_x \in L_x$ such that $\ell_x \in E(H)$, because for every $x \in X$, a neighbor in I_1 is marked if it exists. We define $L = \bigcup_{x \in X} L_x$. Notice that each matching cut of G contains at most one edge of every L_x. We say that two sets of edges M_1 and M_2 are *equivalent* if $M_1 \setminus L = M_2 \setminus L$ and for every $x \in X$, $|M_1 \cap L_x| = |M_2 \cap L_x|$. It is straightforward to verify that the introduced relation is indeed an equivalence relation. It is also easy to see that if M is a matching cut of G, then every $M' \subseteq E(G)$ equivalent to M is a matching cut. We show the following lemma.

▶ **Lemma 10** (∗). *A set of edges $M \subseteq E(G)$ is a matching cut (minimal or maximal matching cut, respectively) of G if and only if H has a matching cut (minimal or maximal matching cut, respectively) M' equivalent to M.*

We use Lemma 10 to obtain our kernelization results. For Enum Minimal MC, we show that the problem admits a fully-polynomial enumeration kernel, and we prove that Enum Maximal MC and Enum MC have polynomial-delay enumeration kernels.

▶ **Theorem 11.** *Enum Minimal MC admits a fully-polynomial enumeration kernel and Enum MC and Enum Maximal MC admit polynomial-delay enumeration kernels with $\mathcal{O}(k^2)$ vertices when parameterized by the vertex cover number k of the input graph.*

Proof. Let G be a graph with $\tau(G) = k$. If $G = K_1$, then the kernelization algorithm returns $H = G_1$ and the solution-lifting algorithm is trivial as G has no matching cuts. Assume that G has at least 2 vertices. If G has no edges, then the empty set is the unique matching cut of G. Then the kernelization algorithm returns $H = 2K_1$, and the solution-lifting algorithm outputs the empty set for the empty matching cut of H. Thus, we can assume without loss of generality that G has at least one edge and $k \geq 1$.

We use the same basic kernelization algorithm that constructs H as described above and output H for all the problems. Recall that $|V(H)| \leq 6k^2 + k + 1$. The kernels differ only in the solution-lifting algorithms. These algorithms exploit Lemma 10 and for every matching cut (minimal or maximal matching cut, respectively) M of H, they list the equivalent matching cuts of G. Lemma 10 guarantees that the families of matching cuts (minimal or maximal matching cuts, respectively) constructed for every matching cut of H compose the partition of the sets of matching cuts (minimal or maximal matching cuts, respectively) of G. This is exactly the property that is required by the definition of a fully-polynomial (polynomial-delay) enumeration kernel. To describe the algorithm, we use the notation defined in this section.

First, we consider Enum Minimal MC. Let M be a minimal matching cut of H. If $M \cap L = \emptyset$, then M is the unique matching cut of G that is equivalent to M, and our algorithm outputs M. Suppose that $M \cap L \neq \emptyset$. Then by the minimality of M, $M = \{\ell_x\}$ for some $x \in X$, because every edge of L is a matching cut. Then the sets $\{e\}$ for every $e \in L_x$ are exactly the matching cuts equivalent to M. Clearly, we have at most n such matching cuts and they can be listed in linear time. This implies that condition (ii) of the definition of a fully-polynomial enumeration kernel is fulfilled. Thus, Enum Minimal MC has a fully-polynomial enumeration kernel with at most $6k^2 + k + 1$ vertices.

Next, we consider Enum Maximal MC and Enum MC. The solution-lifting algorithms for these problems are the same. Let M be a (maximal) matching cut of H. Let also $M_1 = M \cap L$ and $M_2 = M \setminus M_1$. If $M_1 = \emptyset$, then M is the unique matching cut of G that is equivalent to M, and our algorithm outputs M. Assume from now that $M_1 \neq \emptyset$. Then there is $Y \subseteq X$ such that $M_1 = \{\ell_x \mid x \in Y\}$. We use the recursive algorithm Enum Equivalent (see Algorithm 1) that takes as an input a matching S of G and $W \subseteq Y$ and outputs the equivalent matching cuts M' of G such that (i) $S \subseteq M'$, (ii) M' is equivalent to M, and (iii) the constructed matchings M' differ only by some edges of the sets L_x for $x \in W$. Initially, $S = M_2$ and $W = Y$.

Algorithm 1 Enum Equivalent(S, W).

```
if $W = \emptyset$ then
    output $S$
end
else if $S \neq \emptyset$ then
    select arbitrary $x \in W$;
    foreach $e \in L_x$ do
        Enum Equivalent$(S \cup \{e\}, W \setminus \{x\})$
    end
end
```

To enumerate the matching cuts equivalent to M, we call Enum Equivalent(M_2, Y). We claim that Enum Equivalent(M_2, Y) enumerates the matching cuts of G that are equivalent to M with $\mathcal{O}(n)$ delay.

By the definition of the equivalence and Lemma 10, every matching cut M' of G that is equivalent to M can be written as $M' = M_2 \cup \{e_x \mid x \in Y\}$, where e_x is an edge of L_x for $x \in Y$. Then to see the correctness of Enum Equivalent, observe the following. If $W \neq \emptyset$, then the algorithm picks a vertex $x \in W$. Then for every edge $e \in L_x$, it enumerates the matching cuts containing S and e. This means that our algorithm is, in fact, a standard *backtracking* enumeration algorithm (see [30]) and immediately implies that the algorithm lists all the required matching cuts exactly once. Since the depth of the recursion is at most n and the algorithm always outputs a matching cut for each leaf of the search tree, the delay is $\mathcal{O}(n)$. This completes the proof of the polynomial-delay enumeration kernel for Enum Maximal MC and Enum MC.

To conclude the proof of the theorem, let us remark that, formally, the solution-lifting algorithms described in the proof require X. However, in fact, we use only sets L_x that can be computed in polynomial time for given G and H. ◀

Notice that Theorem 11 is tight in the sense that Enum Maximal MC and Enum MC do not admit fully-polynomial enumeration kernels for the parameterization by the vertex cover number. To see this, let k be a positive integer and consider the n-vertex graph G, where $n > k$ is divisible by k, that is the union of k stars $K_{1,p}$ for $p = n/k - 1$. Clearly, $\tau(G) = k$. We observe that G has $p^k = (n/k - 1)^k$ maximal matching cut that are formed by picking one edge from each of the k stars. Similarly, G has $(p+1)^k = (n/k)^k$ matching cuts obtained by picking at most one edge from each star. In both cases, this means that the (maximal) matching cuts cannot be enumerated by an FPT algorithm. By Theorem 3, this rules out the existence of a fully-polynomial enumeration kernel.

We conclude this section by showing that Theorem 11 can be generalized to the weaker parameterization by the *twin-cover* number, introduced by Ganian [17, 18] as a generalization of a vertex cover. Recall that two vertices u and v of a graph G are *true twins* if $N[u] = N[v]$. A set of vertices X of a graph G is said to be a *twin-cover* of G if for every edge uv of G, at least one of the following holds: (i) $u \in X$ or $v \in X$ or (ii) u and v are true twins. The *twin-cover* number, denoted by $\mathsf{tc}(G)$, is the minimum size of a twin-cover. Notice that $\mathsf{tc}(G) \leq \tau(G)$ and $\mathsf{tc}(G) \geq \mathsf{cw}(G) + 2$ for every G [17, 18]. Let $\mathcal{X} = \{X_1, \ldots, X_r\}$ be the partition of $V(G)$ into the classes of true twins. Note that $\mathcal{X}$ can be computed in linear time using an algorithm for computing a modular decomposition [36]. Then we can define the *true-twin quotient* graph $\mathcal{G}$ with respect to $\mathcal{X}$, that is, the graph with the node set $\mathcal{X}$ such that two classes of true twins X_i and X_j are adjacent in $\mathcal{G}$ if and only if the vertices of X_i are adjacent to the vertices of X_j in G. Then it can be seen that $\mathsf{tc}(G) \geq \tau(\mathcal{G})$. We prove the following.

▶ **Theorem 12** (∗). *Enum Minimal MC admits a fully-polynomial enumeration kernel and Enum MC and Enum Maximal MC admit polynomial-delay enumeration kernels with $\mathcal{O}(k^2)$ vertices when parameterized by the vertex cover number of the true-twin quotient graph of the input graph.*

5 Enumeration Kernels for the Parameterization by the Feedback Edge Number

A set of edges X of a graph G is said to be a *feedback* edge set if $G - S$ has no cycle, that is, $G - S$ is a forest. The minimum size of a feedback edge set is called the *feedback edge* number or the *cyclomatic* number. We use $\mathsf{fn}(G)$ to denote the feedback edge number of a graph G. It is well-known (see, e.g., [13]) that if G is a graph with n vertices, m edges and r connected components, then $\mathsf{fn}(G) = m - n + r$ and a feedback edge set of minimum size can be found in linear time. Throughout this section, we assume that the input graph in an instance of Enum Minimal MC or Enum MC is given together with a feedback edge set. Equivalently, we may assume that kernelization and solution-lifting algorithms are supplied by the same algorithm computing a minimum feedback edge set. Then this algorithm computes exactly the same set for the given input graph.

In contrast to vertex cover number and neighborhood diversity, Enum Minimal MC does not admit a fully-polynomial enumeration kernel in case of the feedback edge number: let ℓ and k be positive integers and consider the graph $H_{k,\ell}$ that is constructed as follows.

- For every $i \in \{1, \ldots, k\}$, construct two vertices u_i and v_i and a (u_i, v_i)-path of length ℓ.
- Add edges to make each of $u_1, \cdots, u_k$ and $v_1, \cdots, v_k$ a path of length $k - 1$.

Observe that $H_{k,\ell}$ has at least ℓ^k minimal matching cuts composed by taking one edge from every (u_i, v_i)-path. Since $H_{k,\ell}$ has $n = k(\ell + 1)$ vertices and $\mathsf{fn}(H_{k,\ell}) = k - 1$, the number of minimal matching cuts is at least $\left(\frac{n}{\mathsf{fn}(H_{k,\ell})-1} - 1\right)^{\mathsf{fn}(H_{k,\ell})}$. This immediately implies that the minimal matching cuts cannot be enumerated in FPT time. In particular, Enum Minimal MC cannot have a fully-polynomial enumeration kernel by Theorem 3. However, this problem and Enum MC admit polynomial-delay enumeration kernels.

▶ **Theorem 13.** *Enum Minimal MC and Enum MC admit a polynomial-delay enumeration kernel with $\mathcal{O}(k)$ vertices when parameterized by the feedback edge number k of the input graph.*

The kernels for Enum Minimal MC and Enum MC are similar but the kernel for Enum MC requires some technical details that do not appear in the kernel for Enum Minimal MC. For Enum MC, we need the following observation that follows from the results of Courcelle [8] in the same way as Proposition 4 using a *Counting* MSOL formulation of the enumeration problem.

▶ **Observation 14.** *Let F be a forest and let $A, B, C \subseteq E(F)$ be disjoint edge sets. Then all matchings M of F such that $A \subseteq M$, $B \cap M = \emptyset$, and either $C \subseteq M$ or $C \cap M = \emptyset$ can be enumerated with polynomial delay. Moreover, if u, v are distinct vertices of the same connected component of F and $h \in \{0, 1\}$, then all such (nonempty) matchings with the additional property that $|E(P) \cap A| \mod 2 = h$, where P is the (u, v)-path of F, also can be enumerated with polynomial delay.*

Proof of Theorem 13. We sketch the proof for Enum MC. Let G be a graph with $\mathsf{fn}(G) = k$ and a feedback edge set S of size k. The case where G is a forest can be settled by using Observation 14 (or Proposition 4). We assume from now that G is not a forest. In particular, $S \neq \emptyset$. If G has one or more connected component that are trees, we select an arbitrary vertex v^* of these components. If G has a connected component that contains a vertex of degree one and is not a tree, then arbitrary select such a vertex u^* of degree one and denote by e^* be the edge incident to u^*. Then we iteratively delete vertices of degree at most one distinct from u^* and v^*. Denote by G' the obtained graph. Notice that G' has at most one isolated vertex (the vertex v^*) and at most one vertex of degree one (the vertex u^*). Observe also that S is a minimum feedback edge set of G'. Let $T = G' - S$. Notice that T is a forest and has at most $2|S| + 2 \leq 2k + 2$ vertices of degree at most one. It can be shown that T has at most $2k$ vertices of degree at least three. Denote by X the set of vertices of T that either are end-vertices of the edges of S, or have degree one, or have degree at least three. Then $|X| \leq 4k + 2$, and every vertex v of G' of degree two is an inner vertex of an (x, y)-path P such that $x, y \in X$ and the inner vertices of P are outside X. Moreover, for every two distinct $x, y \in X$, G' has at most one (x, y)-path P_{xy} with all its inner vertices outside X. We denote by $\mathcal{P}$ the set of all such paths. We say that an edge of P_{xy} is the *x-edge* if it is incident to x and is the *y-edge* if it is incident to y. We say that an edge e of P_{xy} is a *second x-edge* (a *second y-edge*, respectively) if e has a common end-vertex with the x-edge (with the y-edge, respectively). The edges that are distinct from the x-edge, the second x-edge, the y-edge and the second y-edge are called *middle edges*. We say that P_{xy} is *long* if P_{xy} has length at least six; otherwise, P_{xy} is *short*. Let $F = G - E(G')$. Since $S \subseteq E(G')$, F is a forest. Moreover, each connected component T of F has at most one vertex in $V(G')$.

We exhaustively apply the following reduction rule.

▶ **Reduction Rule.** *If there is a long path $P_{xy} \in \mathcal{P}$ for some $x, y \in X$, then contract an arbitrary middle edge of P_{xy}.*

Let H be the graph obtained from G' by the exhaustive application of the reduction rule. We also denote by $\mathcal{P}'$ the set of paths obtained from the paths of $\mathcal{P}$; we use P'_{xy} to denote the path obtained from $P_{xy} \in \mathcal{P}$. Our kernelization algorithm returns H together with S. It can be seen that $|V(H)| \leq 20k + 1$.

For the construction of the solution-lifting algorithm, recall that by our assumption the input graph is given together with S and $S \subseteq E(H)$. Then we can identify v^*, u^* and e^* in G and H, and then we can recompute the set X. Next, we can compute the sets of paths $\mathcal{P}$ and $\mathcal{P}'$ of G and H, respectively, in polynomial time. This allows us to assume that the solution-lifting algorithm has access to these sets.

To construct the solution-lifting algorithm, denote by $\mathcal{M}$ and $\mathcal{M}'$ the sets of matching cuts of G and H, respectively. Define $\mathcal{M}_1 = \{M \in \mathcal{M} \mid M \cap E(G') = \emptyset\}$ and $\mathcal{M}_2 = \{M \in \mathcal{M} \mid M \cap E(G') \neq \emptyset\}$. Notice that $M \in \mathcal{M}_1$ is nonempty if and only if M is a nonempty matching of $F = G - E(G')$. First, we deal with the matching cuts of $\mathcal{M}_1$. Observe that G is connected if and only if H is connected. This means that the empty set is a matching cut of G if and only if the empty set is a matching cut of H.

Suppose that H has the empty matching cut. Then the solution-lifting algorithm, given this matching cut of H, outputs the matching cuts of $\mathcal{M}_1$. Notice that $\mathcal{M}_1 \neq \emptyset$, because $\mathcal{M}_1$ contains the empty matching cut. The solution-lifting algorithm outputs the empty matching cut and all nonempty matchings of F using Observation 14.

Assume now that H is connected. Then G is connected as well and $\mathcal{M}_1 \neq \emptyset$ if and only if $F \neq \emptyset$. By the construction of G', if F is not empty, then G has a vertex of degree one. In particular, the kernelization algorithm selects u^* and e^* in this case. Notice that e^* is a bridge of G, and it holds that $\{e^*\}$ is a matching cut of both G and H. Observe also that $\{e^*\} \in \mathcal{M}_2$. This matching cut is generated by the solution-lifting algorithm for the cut $\{e^*\}$ of H: when the algorithm finishes listing the matching cuts of $\mathcal{M}_2$ for $\{e^*\}$, it switches to the listing of all nonempty matchings of F. This can be done with polynomial delay by Observation 14.

Next, we analyze the matching cuts of $\mathcal{M}_2$. By definition, a matching cut M of G is in $\mathcal{M}_2$ if $M \cap E(G') \neq \emptyset$. This means that $M \cap E(G')$ is a matching cut of G', and for a nonempty matching M of G, $M \in \mathcal{M}_2$ if and only if $M \cap E(G')$ is a nonempty matching cut of G'. We exploit this property and the solution-lifting algorithm lists nonempty matching cuts of G' and then for each matching cut of G', it outputs all its possible extensions by matchings of F. For this, we define the following relation between matching cuts of H and G'. Let M be a nonempty matching cut of H and let M' be a nonempty matching of G' (note that we do not require M' to be a matching cut). We say that M' is *equivalent* to M if the following holds:

(i) $M \cap E(H[X]) = M' \cap E(G[X])$ (note that $H[X] = G[X]$).

(ii) For every $P_{xy} \in \mathcal{P}$ such that P_{xy} is short, $M \cap E(P'_{xy}) = M' \cap E(P_{xy})$ (note that $P_{xy} = P'_{xy}$ in this case).

(iii) For every long $P_{xy} \in \mathcal{P}$,

  (a) $M \cap E(P'_{xy}) \neq \emptyset$ if and only if $M' \cap E(P_{xy}) \neq \emptyset$,

  (b) $|M \cap E(P'_{xy})| \mod 2 = |M' \cap E(P_{xy})| \mod 2$,

  (c) the x-edge (y-edge, respectively) of P'_{xy} is in M' if and only if the x-edge (y-edge, respectively) of P_{xy} is in M,

  (d) if for the second x-edge e_x, the second y-edge e_y and the middle edge e of P'_{xy}, $|M \cap \{e_x, e_y, e\}| = 1$, then

  - $e_x \in M$ ($e_y \in M$, respectively) if and only if $e_x \in M'$ and $e_y \notin M'$ ($e_x \notin M'$ and $e_y \in M'$, respectively),
  - $e \in M$ if and only if either $e_x, e_y \in M'$ or $e_x, e_y \notin M'$.

  (note that e_x, e_y are the second x-edge and y-edge of P_{xy}, because P'_{xy} is constructed by contracting of some middle edges of P_{xy}).

We use the properties of the relation summarized in the following claim.

▷ Claim 15.

(i) For every nonempty matching cut M of H, there is a nonempty matching M' of G' that is equivalent to M.

(ii) For every nonempty matching cut M of H and every nonempty matching M' of G' equivalent to M, M' is a matching cut of G'.

(iii) Every nonempty matching cut M' of G' is equivalent to at most one matching cut of H.

(iv) For every nonempty matching cut M' of G', there is a nonempty matching cut of M such that M' is equivalent to M.

Claim 15 allows us to construct the solution-lifting algorithm for nonempty matching cuts of H that outputs nonempty matching cuts from $\mathcal{M}_2$. For each nonempty matching cut M of H, the algorithm lists the matching cuts M' of G' such that M' is equivalent to M. Then for each M', we extend M' to matching cuts of G by adding matchings of $F = G - E(G')$. For this, we consider the algorithm $\textsc{EnumPath}(P_{x,y}, A, B, C, h)$ that given a path $P_{xy} \in \mathcal{P}$, disjoint sets $A, B, C \subseteq E(P_{xy})$, and an integer $h \in \{0, 1\}$, enumerates with polynomial delay all nonempty matchings M of P_{xy} such that $A \subseteq M$, $B \cap M = \emptyset$, either $C \subseteq M$ or $C \cap M = \emptyset$, and $|M| \mod 2 = h$. Such an algorithm exists by Observation 14. We also use the algorithm $\textsc{EnumMatchF}(M)$ that, given a matching cut M of G', lists all matching cuts of G of the form $M \cup M'$, where M' is a matching of F. $\textsc{EnumMatchF}(M)$ is constructed as follows. Let A be the set of edges of F incident to the end-vertices of F (recall that each connected component of F contains at most one vertex of $V(G')$). Then we enumerate the matchings M' of F such that $M' \cap A = \emptyset$. This can be done with polynomial delay by Observation 14.

Algorithm 2 $\textsc{EnumEquivalent}(L, \mathcal{R})$.

```
if $\mathcal{R} = \emptyset$ then
    call $\textsc{EnumMatchF}(M)$;
    return every matching cut $M'$ generated by the algorithm and quit
end
else if $\mathcal{R} \neq \emptyset$ then
    select arbitrary $P_{xy} \in \mathcal{R}$;
    set $A := \emptyset$; $B := \emptyset$; $C := \emptyset$; $h := |M \cap E(P'_{xy})| \mod 2$;
    if $e_x \in M$ then set $A := A \cup \{e_x\}$;
    if $e_y \in M$ then set $A := A \cup \{e_y\}$;
    if $e'_x \in M$ and $e, e'_y \notin M$ then set $A := A \cup \{e'_x\}$ and $B := B \cup \{e'_y\}$;
    if $e'_y \in M$ and $e, e'_x \notin M$ then set $A := A \cup \{e'_y\}$ and $B := B \cup \{e'_x\}$;
    if $e \in M$ and $e'_x, e'_y \notin M$ then set $C := C \cup \{e'_x, e'_y\}$;
    call $\textsc{EnumPath}(P_{x,y}, A, B, C, h)$;
    foreach nonempty matching $Z$ generated by $\textsc{EnumPath}(P_{x,y}, A, B, C, h)$ do
        $\textsc{EnumEquivalent}(L \cup Z, \mathcal{R} \setminus \{P_{xy}\})$
    end
end
```

We use $\textsc{EnumPath}$ and $\textsc{EnumMatchF}$ as subroutines of the recursive branching algorithm $\textsc{EnumEquivalent}$ (see Algorithm 2) that, given a matching M of H, takes as an input a matching L of G and $\mathcal{R} \subseteq \mathcal{P}$ and outputs the matching cuts M' of G such that (i) $L \subseteq M'$, (ii) M' is equivalent to M, and (iii) the constructed matchings M' differ only by some edges of the paths $P_{xy} \in \mathcal{R}$. To initiate the computations, we construct the initial matching L' of G and the initial set of paths $\mathcal{R}' \subseteq \mathcal{P}$ as follows. We define $\mathcal{R}' \subseteq \mathcal{P}$ to be the set of long paths $P_{xy} \subseteq \mathcal{P}$ such that $P'_{xy} \cap M \neq \emptyset$. Then $L' \subseteq M$ is the set of edges of M that are not in the paths of $\mathcal{R}'$. Recall that as an intermediate step, we enumerate nonempty matching cuts of G' that are equivalent to M. Then it can be noted that to do this, we have to enumerate all possible extensions of M to M' satisfying condition (iii) of the equivalence definition. Therefore, we call $\textsc{EnumEquivalent}(L', \mathcal{R}')$ to solve the enumeration problem. It can be seen that $\textsc{EnumEquivalent}(L', \mathcal{R}')$ enumerates with polynomial delay all nonempty matching cuts $M \in \mathcal{M}_2$ such that $M' \cap E(G')$ is a nonempty matching cut of G' equivalent to M.

To summarize, recall that if H is connected and has a vertex of degree one, we used the matching cut $\{e^*\}$ to list the matching cuts formed by the edges of $F = G - E(G')$. Clearly, $\{e^*\}$ is generated by $\textsc{EnumEquivalent}(L', \mathcal{R}')$ for L' and $\mathcal{R}'$ constructed for $M = \{e^*\}$. Therefore, we conclude that the solution-lifting algorithm satisfies condition (ii*) of the definition of a polynomial-delay enumeration kernel. ◀

6 Conclusion

We initiated the systematic study of enumeration kernelization for several variants of the matching cut problem. We obtained fully-polynomial (polynomial-delay) enumeration kernels for the parameterizations by the vertex cover number, twin-cover number, neighborhood diversity, modular width, and feedback edge number. Since the solution-lifting algorithms are simple branching algorithms, these kernels give a condensed view of the solution sets which may be interesting in applications where one may want to inspect all solutions manually. Restricting to polynomial-time and polynomial-delay solution-lifting algorithms seems helpful in the sense that they will usually be easier to understand.

There are many topics for further research in enumeration kernelization. For Matching Cut, it would be interesting to investigate other structural parameters, like the feedback vertex number (see [10] for the definition). More generally, the area of enumeration kernelization seems still somewhat unexplored. It would be interesting to see applications of the various kernel types to other enumeration problems. For this, it seems to be important to develop general tools for enumeration kernelizations. For example, is it possible to establish a framework for enumeration kernelization lower bounds similar to the techniques used for classical kernels [4, 5] (see also [10, 15])?

Concerning the counting and enumeration of matching cuts, we also proved the upper bound $F(n+1) - 1$ for the maximum number of matching cuts of an n-vertex graph and showed that the bound is tight. What can be said about the maximum number of minimal and maximal matching cuts? It is not clear whether our lower bounds given in Propositions 7 and 8 are tight. Finally, it seems promising to study enumeration kernels for d-Cut [21], a generalization of Matching Cut that has recently received some attention.

References

1 N. R. Aravind, Subrahmanyam Kalyanasundaram, and Anjeneya Swami Kare. On structural parameterizations of the matching cut problem. In Xiaofeng Gao, Hongwei Du, and Meng Han, editors, *Combinatorial Optimization and Applications - 11th International Conference, COCOA 2017, Shanghai, China, December 16-18, 2017, Proceedings, Part II*, volume 10628 of *Lecture Notes in Computer Science*, pages 475–482, 2017.

2 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.

3 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

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

5 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discret. Math.*, 28(1):277–305, 2014. doi:10.1137/120880240.

6 Marin Bougeret, Bart M. P. Jansen, and Ignasi Sau. Bridge-depth characterizes which structural parameterizations of vertex cover admit a polynomial kernel. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 16:1–16:19, 2020. doi:10.4230/LIPIcs.ICALP.2020.16.

7 Vasek Chvátal. Recognizing decomposable graphs. *Journal of Graph Theory*, 8(1):51–53, 1984. doi:10.1002/jgt.3190080106.

8 Bruno Courcelle. Linear delay enumeration and monadic second-order logic. *Discret. Appl. Math.*, 157(12):2675–2700, 2009. doi:10.1016/j.dam.2008.08.021.

9 Nadia Creignou, Arne Meier, Julian-Steffen Müller, Johannes Schmidt, and Heribert Vollmer. Paradigms for parameterized enumeration. *Theory Comput. Syst.*, 60(4):737–758, 2017. doi:10.1007/s00224-016-9702-4.

10 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.

11 Peter Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. *Theor. Comput. Sci.*, 351(3):337–350, 2006. doi:10.1016/j.tcs.2005.10.004.

12 Peter Damaschke. Fixed-parameter enumerability of cluster editing and related problems. *Theory Comput. Syst.*, 46(2):261–283, 2010.

13 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

14 Henning Fernau. On parameterized enumeration. In *Computing and Combinatorics, 8th Annual International Conference, COCOON 2002, Singapore, August 15-17, 2002, Proceedings*, volume 2387 of *Lecture Notes in Computer Science*, pages 564–573. Springer, 2002. doi:10.1007/3-540-45655-4_60.

15 Fedor V. Fomin, Daniel Lokshtanove, Saket Saurabh, and Meirav Zehavi. *Kernelization. Theory of Parameterized Preprocessing*. Cambridge University Press, Cambridge, 2019. doi:10.1017/9781107415157.

16 Fedor V. Fomin, Saket Saurabh, and Yngve Villanger. A polynomial kernel for proper interval vertex deletion. *SIAM J. Discret. Math.*, 27(4):1964–1976, 2013. doi:10.1137/12089051X.

17 Robert Ganian. Twin-cover: Beyond vertex cover in parameterized algorithmics. In Dániel Marx and Peter Rossmanith, editors, *Parameterized and Exact Computation - 6th International Symposium, IPEC 2011, Saarbrücken, Germany, September 6-8, 2011. Revised Selected Papers*, volume 7112 of *Lecture Notes in Computer Science*, pages 259–271. Springer, 2011. doi:10.1007/978-3-642-28050-4_21.

18 Robert Ganian. Improving vertex cover as a graph parameter. *Discret. Math. Theor. Comput. Sci.*, 17(2):77–100, 2015. URL: http://dmtcs.episciences.org/2136.

19 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

20 Petr Golovach, Christian Komusiewicz, Dieter Kratsch, and Van Bang Le. Refined notions of parameterized enumeration kernels with applications to matching cut enumerations. *CoRR*, abs2101.03800, 2021. arXiv:2101.03800.

21 Guilherme C. M. Gomes and Ignasi Sau. Finding cuts of bounded degree: Complexity, FPT and exact algorithms, and kernelization. In Bart M. P. Jansen and Jan Arne Telle, editors, *14th International Symposium on Parameterized and Exact Computation, IPEC 2019, September 11-13, 2019, Munich, Germany*, volume 148 of *LIPIcs*, pages 19:1–19:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.IPEC.2019.19.

22 R. L. Graham. On primitive graphs and optimal vertex assignments. *Ann. New York Acad. Sci.*, 175:170–186, 1970.

23 Sun-Yuan Hsieh, Hoàng-Oanh Le, Van Bang Le, and Sheng-Lung Peng. Matching cut in graphs with large minimum degree. In Ding-Zhu Du, Zhenhua Duan, and Cong Tian, editors, *Computing and Combinatorics - 25th International Conference, COCOON 2019, Xi'an, China, July 29-31, 2019, Proceedings*, volume 11653 of *Lecture Notes in Computer Science*, pages 301–312. Springer, 2019. doi:10.1007/978-3-030-26176-4_25.

24 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.

25 Christian Komusiewicz, Dieter Kratsch, and Van Bang Le. Matching cut: Kernelization, single-exponential time fpt, and exact exponential algorithms. *Discret. Appl. Math.*, 283:44–58, 2020. doi:10.1016/j.dam.2019.12.010.

26 Christian Komusiewicz and Johannes Uhlmann. A cubic-vertex kernel for flip consensus tree. *Algorithmica*, 68(1):81–108, 2014.

27 Dieter Kratsch and Van Bang Le. Algorithms solving the matching cut problem. *Theor. Comput. Sci.*, 609:328–335, 2016. doi:10.1016/j.tcs.2015.10.016.

28 Stefan Kratsch. Recent developments in kernelization: A survey. *Bull. EATCS*, 113, 2014.

29 Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. 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 224–237. ACM, 2017. doi:10.1145/3055399.3055456.

30 Andrea Marino. *Analysis and enumeration*, volume 6 of *Atlantis Studies in Computing*. Atlantis Press, Paris, 2015. Algorithms for biological graphs, With forewords by Tiziana Calamoneri and Pierluigi Crescenzi.

31 Kitty Meeks. Randomised enumeration of small witnesses using a decision oracle. *Algorithmica*, 81(2):519–540, 2019.

32 Arne Meier. *Parametrised enumeration.* Habilitation thesis, Leibniz Universit´at Hannover, 2020. doi:10.15488/9427.

33 Sang-il Oum. Approximating rank-width and clique-width quickly. *ACM Trans. Algorithms*, 5(1):10:1–10:20, 2008. doi:10.1145/1435375.1435385.

34 Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

35 Marko Samer and Stefan Szeider. Backdoor trees. In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 363–368. AAAI Press, 2008. URL: http://www.aaai.org/Library/AAAI/2008/aaai08-057.php.

36 Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In *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.

37 Kunihiro Wasa. Enumeration of enumeration algorithms. *CoRR*, abs/1605.05102, 2016. arXiv:1605.05102.

Average-Case Algorithms for Testing Isomorphism of Polynomials, Algebras, and Multilinear Forms

Joshua A. Grochow
Departments of Computer Science and Mathematics, University of Colorado Boulder,
Boulder, CO, USA

Youming Qiao
Center for Quantum Software and Information, University of Technology Sydney,
Ultimo, NSW 2007, Australia

Gang Tang
Center for Quantum Software and Information, University of Technology Sydney,
Ultimo, NSW 2007, Australia

Abstract

We study the problems of testing isomorphism of polynomials, algebras, and multilinear forms. Our first main results are average-case algorithms for these problems. For example, we develop an algorithm that takes two cubic forms $f, g \in \mathbb{F}_q[x_1, \dots, x_n]$, and decides whether f and g are isomorphic in time $q^{O(n)}$ *for most* f. This average-case setting has direct practical implications, having been studied in multivariate cryptography since the 1990s. Our second result concerns the complexity of testing equivalence of alternating trilinear forms. This problem is of interest in both mathematics and cryptography. We show that this problem is polynomial-time equivalent to testing equivalence of symmetric trilinear forms, by showing that they are both Tensor Isomorphism-complete (Grochow & Qiao, *ITCS* 2021), therefore is equivalent to testing isomorphism of cubic forms over most fields.

2012 ACM Subject Classification Computing methodologies → Algebraic algorithms; Computing methodologies → Combinatorial algorithms

Keywords and phrases polynomial isomorphism, trilinear form equivalence, algebra isomorphism, average-case algorithms, tensor isomorphism complete, symmetric and alternating bilinear maps

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.38

Related Version *Full Version*: https://arxiv.org/abs/2012.01085

Funding *Joshua A. Grochow*: Partially supported by NSF grants DMS-1750319 and DMS-1622390.
Youming Qiao: Partially supported by the Australian Research Council DP200100950.
Gang Tang: Partially supported by the Australian Research Council DP160101652.

Acknowledgements We thank the anonymous reviewers for their careful reading and helpful suggestions.

1 Introduction

In this paper, we study isomorphism testing problems for polynomials, algebras, and multilinear forms. Our first set of results is algorithmic, namely average-case algorithms for these problems (Section 1.1). Our second result is complexity-theoretic, concerning the problems of testing equivalence of symmetric and alternating trilinear forms (Section 1.2).

1.1 Average-case algorithms for polynomial isomorphism and more

The polynomial isomorphism problem. Let $\mathbb{F}$ be a field, and let $X = \{x_1, \dots, x_n\}$ be a set of variables. Let $\mathrm{GL}(n, \mathbb{F})$ be the general linear group consisting of $n \times n$ invertible matrices over $\mathbb{F}$. A natural group action of $A = (a_{i,j}) \in \mathrm{GL}(n, \mathbb{F})$ on the polynomial ring $\mathbb{F}[X]$ sends $f(x_1, \dots, x_n)$ to $f \circ A := f(\sum_{j=1}^{n} a_{1,j}x_j, \dots, \sum_{j=1}^{n} a_{n,j}x_j)$. The *polynomial isomorphism*

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 38; pp. 38:1–38:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

problem (PI) asks, given $f, g \in \mathbb{F}[X]$, whether there exists $A \in \mathrm{GL}(n, \mathbb{F})$ such that $f = g \circ A$. In the literature, this problem was also called the polynomial equivalence problem [1].

An important subcase of PI is when the input polynomials are required to be homogeneous of degree d. In this case, this problem is referred to as the homogeneous polynomial isomorphism problem, denoted as d-HPI. Homogeneous degree-3 (resp. degree-2) polynomials are also known as cubic (resp. quadratic) forms.

In this article, we assume that a polynomial is represented in algorithms by its list of coefficients of the monomials, though other representations like algebraic circuits are also possible in this context [13]. Furthermore, we shall mostly restrict our attention to the case when the polynomial degrees are constant.

Motivations to study polynomial isomorphism. The polynomial isomorphism problem has been studied in both multivariate cryptography and computational complexity. In 1996, inspired by the celebrated zero-knowledge protocol for graph isomorphism [9], Patarin proposed to use PI as the security basis of authentication and signature protocols [17]. This lead to a series of works on practical algorithms for PI; see [3,4,12] and references therein. In the early 2000s, Agrawal, Kayal and Saxena studied PI from the computational complexity perspective. They related PI with graph isomorphism and algebra isomorphism [1,2], and studied some special instances of PI [13] as well as several related algorithmic tasks [18].

Despite these works, little progress has been made on algorithms *with rigorous analysis* for the *general* PI. More specifically, Kayal's algorithm [13] runs in randomized polynomial time, works for the degree $d \geq 4$, and doesn't require the field to be finite. However, it only works in the *multilinear* setting, namely when f and g are isomorphic to a common multilinear polynomial h. The algorithms from multivariate cryptography [4] either are heuristic, or need unproven assumptions. While these works contain several nice ideas and insights, and their implementations show practical improvements, they are nonetheless heuristic in nature, and rigorously analyzing them seems difficult. Indeed, if any of these algorithms had worst-case analysis matching their heuristic performance, it would lead to significant progress on the long-open Group Isomorphism problem (see, e.g., [11,15]).

Our result on polynomial isomorphism. Our first result is an average-case algorithm with rigorous analysis for PI over a finite field $\mathbb{F}_q$. As far as we know, this is the first non-trivial algorithm with rigorous analysis for PI over finite fields. (The natural brute-force algorithm, namely enumerating all invertible matrices, runs in time $q^{n^2} \cdot \mathrm{poly}(n, \log q)$.) Furthermore, the average-case setting is quite natural, as it is precisely the one studied multivariate cryptography. We elaborate on this further after stating our result.

To state the result, let us define what a random polynomial means in this setting. Since we represent polynomials by their lists of coefficients, a random polynomial of degree d is naturally the one whose coefficients of the monomials of degree $\leq d$ are independently randomly drawn from $\mathbb{F}_q$. We also consider the homogeneous setting where only monomials of degree $= d$ are of interest.

▶ **Theorem 1.** *Let $d \geq 3$ be a constant. Let $f, g \in \mathbb{F}_q[x_1, \dots, x_n]$ be (resp. homogeneous) polynomials of degree $\leq d$ (resp. $= d$). There exists an $q^{O(n)}$-time algorithm that decides whether f and g are isomorphic, for all but at most $\frac{1}{q^{\Omega(n)}}$ fraction of f.*

Furthermore, if f and g are isomorphic, then this algorithm also computes an invertible matrix A which sends f to g.

Let us briefly indicate the use of this average-case setting in multivariate cryptography. In the authentication scheme described in [17], the public key consists of two polynomials $f, g \in \mathbb{F}_q[x_1, \dots, x_n]$, where f is a random polynomial, and g is obtained by applying a random invertible matrix to f. Then f and g are public keys, and any isomorphism from f to g can serve as the private key. Therefore, the algorithm in Theorem 1 can be used to recover a private key for most f.

Adapting the algorithm strategy to more isomorphism problems. In [1, 2], the algebra isomorphism problem (AI) was studied and shown to be polynomial-time equivalent to PI over most fields. In [11], many more problems are demonstrated to be polynomial-time equivalent to PI, including the trilinear form equivalence problem (TFE). In these reductions, due to the blow-up of the parameters, the $q^{O(n)}$-time algorithm in Theorem 1 does not translate to moderately exponential-time, average-case algorithms for these problems. The algorithm design idea, however, does translate to give moderately exponential-time, average-case algorithms for AI and TFE. This will be shown in Section 3.2.

1.2 Complexity of symmetric and alternating trilinear form equivalence

From cubic forms to symmetric and alternating trilinear forms. In the context of polynomial isomorphism, cubic forms are of particular interest. In complexity theory, it was shown that d-HPI reduces to cubic form isomorphism over fields with dth roots of unity [1,2], or over fields of characteristic 0 or $> d$ [11]. In multivariate cryptography, cubic form isomorphism also received special attention, since using higher degree forms results in less efficiency in the cryptographic protocols.

Just as quadratic forms are closely related with symmetric bilinear forms, cubic forms are closely related with symmetric trilinear forms. Let $\mathbb{F}$ be a field of characteristic not 2 or 3, and let $f = \sum_{1 \leq i \leq j \leq k \leq n} a_{i,j,k} x_i x_j x_k \in \mathbb{F}[x_1, \dots, x_n]$ be a cubic form. For any $i, j, k \in [n]$, let $1 \leq i' \leq j' \leq k' \leq n$ be the result of sorting i, j, k in the increasing order, and set $a_{i,j,k} = a_{i',j',k'}$. Then we can define a symmetric[1] trilinear form $\phi_f : \mathbb{F}^n \times \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F}$ by

$$\phi_f(u, v, w) = \sum_{i \in [n]} a_{i,i,i} u_i v_i w_i + \frac{1}{3} \cdot \sum_{\substack{i,j,k \in [n] \\ |\{i,j,k\}|=2}} a_{i,j,k} u_i v_j w_k + \frac{1}{6} \cdot \sum_{\substack{i,j,k \in [n] \\ i,j,k \text{ all different}}} a_{i,j,k} u_i v_j w_k.$$

It can be seen easily that for any $v = (v_1, \dots, v_n)^{\mathrm{t}} \in \mathbb{F}^n$, $f(v_1, \dots, v_n) = \phi_f(v, v, v)$.

In the theory of bilinear forms, symmetric and skew-symmetric bilinear forms are two important special subclasses. For example, they are critical in the classifications of classical groups [19] and finite simple groups [21]. For trilinear forms, we also have skew-symmetric trilinear forms. In fact, to avoid some complications over fields of characteristics 2 or 3, we shall consider alternating trilinear forms which are closely related to skew-symmetric ones. For trilinear forms, the exceptional groups of type E_6 can be constructed as the stabilizer of certain symmetric trilinear forms, and those of type G_2 can be constructed as the stabilizer of certain alternating trilinear forms.

We say that a trilinear form $\phi : \mathbb{F}^n \times \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F}$ is *alternating*, if whenever two arguments of ϕ are equal, ϕ evaluates to zero. Note that this implies skew-symmetry, namely for any $u_1, u_2, u_3 \in \mathbb{F}^n$ and any $\sigma \in \mathrm{S}_3$, $\phi(u_1, u_2, u_3) = \mathrm{sgn}(\sigma) \cdot \phi(u_{\sigma(1)}, u_{\sigma(2)}, u_{\sigma(3)})$. Over fields of characteristic zero or > 3, this is equivalent to skew-symmetry.

[1] That is, for any permutation $\sigma \in \mathrm{S}_3$, $\phi(u_1, u_2, u_3) = \phi(u_{\sigma(1)}, u_{\sigma(2)}, u_{\sigma(3)})$.

The trilinear form equivalence problem. Given a trilinear form $\phi : \mathbb{F}^n \times \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F}$, $A \in \mathrm{GL}(n, \mathbb{F})$ naturally acts on ϕ by sending it to $\phi \circ A := \phi(A^{-1}(u), A^{-1}(v), A^{-1}(w))$. The *trilinear form equivalence problem* then asks, given two trilinear forms $\phi, \psi : \mathbb{F}^n \times \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F}$, whether there exists $A \in \mathrm{GL}(n, \mathbb{F})$, such that $\phi = \psi \circ A$. Over fields of characteristic not 2 or 3, two cubic forms f and g are isomorphic if and only if ϕ_f and ϕ_g are equivalent, so cubic form isomorphism is polynomial-time equivalent to symmetric trilinear form equivalence over such fields. Note that for clarity, we reserve the term "isomorphism" for polynomials (and cubic forms), and use "equivalence" for multilinear forms.

Motivations to study alternating trilinear form equivalence. Our main interest is to study the complexity of alternating trilinear form equivalence, with the following motivations.

The first motivation comes from cryptography. To store a symmetric trilinear form on $\mathbb{F}_q^n$, $\binom{n+2}{3}$ field elements are required. To store an alternating trilinear form on $\mathbb{F}_q^n$, $\binom{n}{3}$ field elements are needed. The difference between $\binom{n+2}{3}$ and $\binom{n}{3}$ could be significant for practical purposes. For example, when $n = 9$, $\binom{n+2}{3} = \binom{11}{3} = 165$, while $\binom{n}{3} = \binom{9}{3} = 84$. This means that in the authentication protocol of Patarin [17], using alternating trilinear forms instead of cubic forms for $n = 9$,[2] one saves almost one half in the public key size, which is an important saving in practice.

The second motivation originates from comparing symmetric and alternating bilinear forms. It is well-known that, *in the bilinear case*, the structure of alternating forms is simpler than that for symmetric ones [14]. Indeed, up to equivalence, an alternating bilinear form is completely determined by its rank over any field, while the classification of symmetric bilinear forms depends crucially on the underlying field. For example, recall that over $\mathbb{R}$, a symmetric form is determined by its "signature", so just the rank is not enough.

A third motivation is implied by the representation theory of the general linear groups; namely that alternating trilinear forms are the "last" natural case for $d = 3$. If we consider the action of $\mathrm{GL}(n, \mathbb{C})$ acting on d-tensors in $\mathbb{C}^n \otimes \mathbb{C}^n \otimes \cdots \otimes \mathbb{C}^n$ diagonally (that is, the same matrix acts on each tensor factor), it is a classical result [19] that the invariant subspaces of $(\mathbb{C}^n)^{\otimes d}$ under this action are completely determined by the irreducible representations of $\mathrm{GL}(n, \mathbb{C})$. When $d = 3$, there are only three such representations, which correspond precisely to: symmetric trilinear forms, Lie algebras, and alternating trilinear forms. From the complexity point of view, it was previously shown that isomorphism of symmetric trilinear forms [1, 2] and Lie algebras [11] are equivalent to algebra isomorphism. Here we show that the last case, isomorphism of alternating trilinear forms, is also equivalent to the others.

The complexity of alternating trilinear form equivalence. Given the above discussion on the comparison between symmetric and alternating bilinear forms, one may wonder whether alternating trilinear form equivalence was easier than symmetric trilinear form equivalence. Interestingly, we show that this is not the case; rather, they are polynomial-time equivalent.

▶ **Theorem 2.** *The alternating trilinear form equivalence problem is polynomial-time equivalent to the symmetric trilinear form equivalence problem.*

[2] The parameters of the cryptosystem are q and n. When $q = 2$, $n = 9$ is not secure as it can be solved in practice [5]. So q needs to be large for $n = 9$ to be secure. Interestingly, according to [4, pp. 227], the parameters $q = 16$ and $n = 8$ seemed difficult for practical attacks via Gröbner basis.

Note here that the reduction from alternating to symmetric trilinear form equivalence requires us to go through the tensor isomorphism problem, which causes polynomial blow-ups in the dimensions of the underlying vector spaces. Therefore, though these two problems are polynomial-time equivalent, these problems may result in cryptosystems with different efficiencies for a given security level.

1.3 Previous works

The relation between $\boldsymbol{PI}$ and $\boldsymbol{AI}$. As mentioned in Section 1.1, the degree-d homogeneous polynomial isomorphism problem (d-HPI) was shown to be almost equivalent to the algebra isomorphism problem (AI) in [1, 2]. (See Section 3.2 for the formal definition of algebra isomorphism problem.) Here, almost refers to that for the reduction from d-HPI to AI in [1, 2], the underlying fields are required to contain a dth root of unity. When $d = 3$, this means that the characteristic of the underlying field p satisfies that $p = 2 \mod 3$ or $p = 0$, which amounts to half of the primes. In [11], another reduction from 3-HPI to AI was presented, which works for fields of characteristics not 2 or 3. The reduction from AI to 3-HPI in [2] works over any field.

The tensor isomorphism complete class. In [8, 11], polynomial-time equivalences are proved between isomorphism testing of many more mathematical structures, including tensors, matrix spaces, polynomial maps, and so on. These problems arise from many areas: besides multivariate cryptography and computational complexity, they appear in quantum information, machine learning, and computational group theory. This motivates the authors of [11] to define the tensor isomorphism complete class TI, which we recall here.

▶ **Definition 3** (The d-Tensor Isomorphism problem, and the complexity class TI)**.** *d-Tensor Isomorphism over a field $\mathbb{F}$ is the problem: given two d-way arrays $\mathtt{A} = (a_{i_1,\dots,i_d})$ and $\mathtt{B} = (b_{i_1,\dots,i_d})$, where $i_k \in [n_k]$ for $k \in [d]$, and $a_{i_1,\dots,i_d}, b_{i_1,\dots,i_d} \in \mathbb{F}$, decide whether there are $P_k \in \mathrm{GL}(n_k, \mathbb{F})$ for $k \in [d]$, such that for all $i_1, \dots, i_d$,*

$$a_{i_1,\dots,i_d} = \sum_{j_1,\dots,j_d} b_{j_1,\dots,j_d} (P_1)_{i_1,j_1} (P_2)_{i_2,j_2} \cdots (P_d)_{i_d,j_d}. \tag{1}$$

For any field $\mathbb{F}$, $\mathrm{TI}_{\mathbb{F}}$ denotes the class of problems that are polynomial-time Turing (Cook) reducible to d-Tensor Isomorphism over $\mathbb{F}$, for some d. A problem is $\mathrm{TI}_{\mathbb{F}}$-complete, if it is in $\mathrm{TI}_{\mathbb{F}}$, and d-Tensor Isomorphism over $\mathbb{F}$ for any d reduces to this problem.

When a problem is naturally defined and is $\mathrm{TI}_{\mathbb{F}}$-complete over any $\mathbb{F}$, then we can simply write that it is TI-complete.

Average-case algorithms for matrix space isometry. In [6, 15], motivated by testing isomorphism of p-groups (widely believed to be the hardest cases of Group Isomorphism, see e.g. [10]), the algorithmic problem alternating matrix space isometry was studied. (In the literature [20], this problem was also known as the alternating bilinear map pseudo-isometry problem.) That problem asks the following: given two linear spaces of alternating matrices $\mathcal{A}, \mathcal{B} \leq \Lambda(n, q)$, decide whether there exists $T \in \mathrm{GL}(n, q)$, such that $\mathcal{A} = T^{\mathrm{t}}\mathcal{B}T = \{T^{\mathrm{t}}BT : B \in \mathcal{B}\}$. (See Section 2 for the definition of alternating matrices.) The main result of [6], improving upon the one in [15], is an average-case algorithm for this problem in time $q^{O(n+m)}$, where $m = \dim(\mathcal{A})$.

1.4 Remarks on the technical side

Techniques for proving Theorem 1. The algorithm for PI in Theorem 1 is based on the algorithmic idea from [6,15]. However, to adapt that idea to the PI setting, there are several interesting conceptual and technical difficulties.

One conceptual difficulty is that for alternating matrix space isometry, there are actually two GL actions, one is by $\mathrm{GL}(n,q)$ as explicitly described above, and the other is by $\mathrm{GL}(m,q)$ performing the basis change of matrix spaces. The algorithm in [6] crucially uses that the $\mathrm{GL}(m,q)$ action is "independent" of the $\mathrm{GL}(n,q)$ action. For PI, there is only one $\mathrm{GL}(n,q)$-action acting on all the variables. Fortunately, as we show in Section 3.1, there is still a natural way of applying the the basic idea from [6,15].

One technical difficulty is that the analysis in [6] relies on properties of random alternating matrices, while for 3-HPI, the analysis relies on properties of random symmetric matrices. To adapt the proof strategy in [6] (based on [15]) to the symmetric setting is not difficult, but suggests some interesting differences between symmetric and alternating matrices (see Appendix A in the full version of this paper).

Techniques for proving Theorem 2. By [8], the trilinear form equivalence problem is in TI, and so are the special cases symmetric and alternating trilinear form equivalence. The proof of Theorem 2 goes by showing that both symmetric and alternating trilinear form equivalence are TI-hard.

Technically, the basic proof strategy is to adapt a gadget construction, which originates from [8] and then is further used in [11]. To use that gadget in the trilinear form setting does require several non-trivial ideas. First, we identify the right TI-complete problem to start with, namely the alternating (resp. symmetric) matrix space isometry problem. Second, we need to arrange a 3-way array $\mathtt{A}$, representing a linear basis of an alternating (resp. symmetric) matrix spaces, into one representing an alternating trilinear form. This requires 3 copies of $\mathtt{A}$, assembled in an appropriate manner. Third, we need to add the gadget in three directions (instead of just two as in previous results). All these features were not present in [8,11]. The correctness proof also requires certain tricky twists compared with those in [8] and [11].

2 Preliminaries

Notations. We collect the notations here, though some of them have appeared in Section 1. Let $\mathbb{F}$ be a field. Vectors in $\mathbb{F}^n$ are column vectors. Let e_i denote the ith standard basis vector of $\mathbb{F}^n$. Let $\mathrm{M}(\ell \times n, \mathbb{F})$ be the linear space of $\ell \times n$ matrices over $\mathbb{F}$, and set $\mathrm{M}(n, \mathbb{F}) := \mathrm{M}(n \times n, \mathbb{F})$. Let I_n denote the identity matrix of size n. For $A \in \mathrm{M}(n, \mathbb{F})$, A is *symmetric* if $A^{\mathrm{t}} = A$, and *alternating* if for every $v \in \mathbb{F}^n$, $v^{\mathrm{t}} A v = 0$. When the characteristic of $\mathbb{F}$ is not 2, A is alternating if and only if A is skew-symmetric. Let $\mathrm{S}(n, \mathbb{F})$ be the linear space of $n \times n$ symmetric matrices over $\mathbb{F}$, and let $\Lambda(n, \mathbb{F})$ be the linear space of alternating matrices over $\mathbb{F}$. When $\mathbb{F} = \mathbb{F}_q$, we may write $\mathrm{M}(n, \mathbb{F}_q)$ as $\mathrm{M}(n, q)$. We use $\langle \cdot \rangle$ to denote the linear span.

3-way arrays. A 3-way array over a field $\mathbb{F}$ is an array with three indices whose elements are from $\mathbb{F}$. We use $\mathrm{M}(n_1 \times n_2 \times n_3, \mathbb{F})$ to denote the linear space of 3-way arrays of side lengths $n_1 \times n_2 \times n_3$ over $\mathbb{F}$.

Let $\mathtt{A} \in \mathrm{M}(\ell \times n \times m, \mathbb{F})$. For $k \in [m]$, the kth *frontal* slice of $\mathtt{A}$ is $(a_{i,j,k})_{i \in [\ell], j \in [n]} \in \mathrm{M}(\ell \times n, \mathbb{F})$. For $j \in [n]$, the jth *vertical* slice of $\mathtt{A}$ is $(a_{i,j,k})_{i \in [\ell], k \in [m]} \in \mathrm{M}(\ell \times m, \mathbb{F})$. For $i \in [\ell]$, the ith *horizontal* slice of $\mathtt{A}$ is $(a_{i,j,k})_{j \in [n], k \in [m]} \in \mathrm{M}(n \times m, \mathbb{F})$. We shall often think of $\mathtt{A}$ as a matrix tuple in $\mathrm{M}(\ell \times n, \mathbb{F})^m$ consisting of its frontal slices.

A natural action of $(P, Q, R) \in \mathrm{GL}(\ell, \mathbb{F}) \times \mathrm{GL}(n, \mathbb{F}) \times \mathrm{GL}(m, \mathbb{F})$ sends a 3-way array $\mathtt{A} \in \mathrm{M}(\ell \times n \times m, \mathbb{F})$ to $P^{\mathrm{t}}\mathtt{A}^{R}Q$, defined as follows. First represent $\mathtt{A}$ as an m-tuple of $\ell \times n$ matrices $\mathbf{A} = (A_1, \dots, A_m) \in \mathrm{M}(\ell \times n, \mathbb{F})^m$. Then P and Q send $\mathbf{A}$ to $P^{\mathrm{t}}\mathbf{A}Q = (P^{\mathrm{t}}A_1Q, \dots, P^{\mathrm{t}}A_mQ)$, and $R = (r_{i,j})$ sends $\mathbf{A}$ to $(A_1', \dots, A_m')$ where $A_i' = \sum_{j \in [m]} r_{i,j}A_j$. Clearly, the actions of P, Q, and R commute. The resulting m-tuple of $\ell \times n$ matrices obtained by applying P, Q, and R to $\mathbf{A}$ is then $P^{\mathrm{t}}\mathbf{A}^{R}Q$. Note that up to possibly relabelling indices, the entries of $P^{\mathrm{t}}\mathbf{A}^{R}Q$ are explicitly defined as in Equation 1.

Useful results. Let $\mathbf{A} = (A_1, \dots, A_m), \mathbf{B} = (B_1, \dots, B_m) \in \mathrm{M}(n, \mathbb{F})^m$. Given $T \in \mathrm{GL}(n, \mathbb{F})$, let $T^{\mathrm{t}}\mathbf{A}T = (T^{\mathrm{t}}A_1T, \dots, T^{\mathrm{t}}A_mT)$. We say that $\mathbf{A}$ and $\mathbf{B}$ are *isometric*, if there exists $T \in \mathrm{GL}(n, \mathbb{F})$ such that $T^{\mathrm{t}}\mathbf{A}T = \mathbf{B}$. Let $\mathrm{Iso}(\mathbf{A}, \mathbf{B}) = \{T \in \mathrm{GL}(n, \mathbb{F}) : \mathbf{A} = T^{\mathrm{t}}\mathbf{B}T\}$, and set $\mathrm{Aut}(\mathbf{A}) := \mathrm{Iso}(\mathbf{A}, \mathbf{A})$. Clearly, $\mathrm{Aut}(\mathbf{A})$ is a subgroup of $\mathrm{GL}(n, q)$, and $\mathrm{Iso}(\mathbf{A}, \mathbf{B})$ is a coset of $\mathrm{Aut}(\mathbf{A})$.

▶ **Theorem 4** ([7, 12])**.** *Let $\mathbf{A}, \mathbf{B} \in \mathrm{S}(n, q)^m$ (resp. $\Lambda(n, q)^m$) for some odd q. There exists a $\mathrm{poly}(n, m, q)$-time deterministic algorithm which takes $\mathbf{A}$ and $\mathbf{B}$ as inputs and outputs $\mathrm{Iso}(\mathbf{A}, \mathbf{B})$, specified by (if nonempty) a generating set of $\mathrm{Aut}(\mathbf{A})$ (by the algorithm in [7]) and a coset representative $T \in \mathrm{Iso}(\mathbf{A}, \mathbf{B})$ (by the algorithm in [12]).*

3 Average-case algorithms for polynomial isomorphism and more

We shall present the algorithm for the cubic form isomorphism problem in detail in Section 3.1. We will state our results for problems like algebra isomorphism in Section 3.2.

3.1 Cubic form isomorphism over fields of odd order

We present the algorithm for cubic form isomorphism over fields of odd characteristic, as this algorithm already captures the essence of the idea, and cubic forms are most interesting from the PI perspective as mentioned in Section 1.2. A full proof of Theorem 1, which is a relatively minor extension of Theorem 5, can be found in the full version of this paper.

▶ **Theorem 5.** *Let $\mathbb{F}_q$ be a finite field of odd order, and $X = \{x_1, \dots, x_n\}$ be a set of commuting variables. Let $f, g \in \mathbb{F}_q[X]$ be two cubic forms. There exists a deterministic algorithm that decides whether f and g are isomorphic in time $q^{O(n)}$, for all but at most $\frac{1}{q^{\Omega(n)}}$ fraction of f.*

Proof. Let r be a constant to be determined later on, and suppose n is sufficiently larger than r. (Indeed, by Lemma 6, the constant r can be taken as 8.) Our goal is to find $T \in \mathrm{GL}(n, q)$, such that $f = g \circ T$.

The algorithm consists of two main steps. Let us first give an overview of the two steps.

In the first step, we show that there exists a set of at most $q^{O(rn)}$-many $T_1 \in \mathrm{GL}(n, q)$, such that every $T \in \mathrm{GL}(n, q)$ can be written as T_1T_2, where T_2 is of the form

$$\begin{bmatrix} I_r & 0 \\ 0 & R \end{bmatrix}. \tag{2}$$

Furthermore, such T_1 can be enumerated in time $q^{O(rn)}$. We then set $g_1 = g \circ T_1$.

In the second step, we focus on searching for T_2 such that $f = g_1 \circ T_2$. The key observation is that those T_2 as in Equation 2 leave x_i, $i \in [r]$, invariant, and send x_j, $j \in [r+1, n]$, to a linear combination of x_k, $k \in [r+1, n]$. It follows that for any fixed $i \in [r]$, T_2 sends $\sum_{r+1 \le j \le k \le n} a_{i,j,k}x_ix_jx_k$ to a linear combination of $x_ix_jx_k$, $r+1 \le j \le k \le n$. We will use this observation to show that for a random f, the number of T_2 satisfying $f = g_1 \circ T_2$ is upper bounded by q^n with high probability. Furthermore, such T_2, if they exist, can be enumerated efficiently. This allows us to go over all possible T_2 and test if $f = g_1 \circ T_2$.

The first step. We show that there exist at most $q^{O(rn)}$-many $T_1 \in \mathrm{GL}(n, q)$, such that any $T \in \mathrm{GL}(n, q)$ can be written as T_1T_2 where T_2 is of the form as in Equation 2.

Recall that e_i is the ith standard basis vector. Let $E_r = \langle e_1, \dots, e_r\rangle$, and let $F_r = \langle e_{r+1}, \dots, e_n\rangle$. Suppose for $i \in [r]$, $T(e_i) = u_i$, and $T(F_r) = V \le \mathbb{F}_q^n$. Let T_1 be any matrix that satisfies $T_1(e_i) = u_i$, and $T_1(F_r) = V$. Let $T_2 = T_1^{-1}T$. Then T_2 satisfies that for $i \in [r]$, $T_2(e_i) = e_i$, and $T_2(F_r) = F_r$. In other words, T_2 is of the form in Equation 2.

We then need to show that these T_1 can be enumerated in time $q^{O(rn)}$.

Recall that T_1 is determined by the images of e_i, $i \in [r]$, and $F_r \le \mathbb{F}_q^n$. So we first enumerate matrices of the form $\begin{bmatrix} u_1 & \dots & u_r & e_{r+1} & \dots & e_n\end{bmatrix}$, where $u_i \in \mathbb{F}_q^n$ are linearly independent. We then need to enumerate the possible images of F_r. Let $U = \langle u_1, \dots, u_r\rangle$. Then the image of F_r is a complement subspace of U. It is well-known that the number of complement subspaces of a dimension-r space is $\sim q^{r(n-r)}$. To enumerate all complement subspaces of U, first compute one complement subspace $V = \langle v_1, \dots, v_{n-r}\rangle$. Then it is easy to verify that, when going over $A = (a_{i,j})_{i\in[r],j\in[n-r]} \in \mathrm{M}(r \times (n-r), q)$, $\langle v_j + \sum_{i\in[r]} a_{i,j}u_i : j \in [n-r]\rangle$ go over all complement subspaces of U. It follows that we can enumerate matrices T_1 of the form $\begin{bmatrix} u_1 & \dots & u_r & v_1 + \sum_{i\in[r]} a_{i,1}u_i & \dots & v_{n-r} + \sum_{i\in[r]} a_{i,n-r}u_i\end{bmatrix}$.

The second step. In Step 1, we computed a set of invertible matrices $\{T_1\} \subseteq \mathrm{GL}(n, q)$ such that every $T \in \mathrm{GL}(n, q)$ can be written as $T = T_1T_2$ where $T_2 = \begin{bmatrix} I_r & 0 \\ 0 & R\end{bmatrix}$. So we set $h := g \circ T_1$ and focus on finding T_2 of the above form such that $f = h \circ T_2$.

Suppose $f = \sum_{1\le i\le j\le k\le n} \alpha_{i,j,k}x_ix_jx_k$, and $h = \sum_{1\le i\le j\le k\le n} \beta_{i,j,k}x_ix_jx_k$. For $i \in [r]$, define $f_i = \sum_{r+1\le j\le k\le n} \alpha_{i,j,k}x_ix_jx_k$. Similarly define h_i.

The key observation is that, due to the form of T_2, we have that $f_i = h_i \circ T_2$. This is because for $i \in [r]$, T_2 sends x_i to x_i, and for $j \in [r+1, n]$, T_2 sends x_j to a linear combination of x_k, $k \in [r+1, n]$.

Let $\ell = n - r$. We then rename the variable x_{r+i}, $i \in [\ell]$ as y_i. Let $Y = \{y_1, \dots, y_\ell\}$. Then from f, we define r quadratic forms in Y,

$$\forall i \in [r], c_i = \sum_{1\le j\le k\le \ell} \alpha'_{i,j,k}y_jy_k, \text{ where } \alpha'_{i,j,k} = \alpha_{i,r+j,r+k}. \tag{3}$$

Correspondingly, we define r quadratic forms $d_i = \sum_{1\le j\le k\le \ell} \beta'_{i,j,k}y_jy_k$, $i \in [r]$, from g_1.

Our task now is to search for the $R \in \mathrm{GL}(\ell, q)$ such that for every $i \in [r]$, $c_i = d_i \circ R$.

To do that, we adopt the classical representation of quadratic forms as symmetric matrices. Here we use the assumption that q is odd. Using the classical correspondence between quadratic forms and symmetric matrices, from c_i we construct

$$C_i = \begin{bmatrix} \alpha'_{i,1,1} & \frac{1}{2}\alpha'_{i,1,2} & \cdots & \frac{1}{2}\alpha'_{i,1,\ell} \\ \frac{1}{2}\alpha'_{i,1,2} & \alpha'_{i,2,2} & \cdots & \frac{1}{2}\alpha'_{i,2,\ell} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{2}\alpha'_{i,1,\ell} & \frac{1}{2}\alpha'_{i,2,\ell} & \cdots & \alpha'_{i,\ell,\ell}\end{bmatrix} \in \mathrm{S}(\ell, q). \tag{4}$$

Similarly define D_i from d_i. It is classical that $c_i = d_i \circ R$ if and only if $C_i = R^{\mathrm{t}}D_iR$.

Let $\mathbf{C} = (C_1, \ldots, C_r) \in \mathrm{S}(\ell, q)^r$, and $\mathbf{D} = (D_1, \ldots, D_r) \in \mathrm{S}(\ell, q)^r$. Recall that $\mathrm{Aut}(\mathbf{C}) = \{R \in \mathrm{GL}(\ell, \mathbb{F}) : R^{\mathrm{t}}\mathbf{C}R = \mathbf{C}\}$, and $\mathrm{Iso}(\mathbf{C}, \mathbf{D}) = \{R \in \mathrm{GL}(\ell, \mathbb{F}) : \mathbf{C} = R^{\mathrm{t}}\mathbf{D}R\}$. Clearly, $\mathrm{Iso}(\mathbf{C}, \mathbf{D})$ is a (possibly empty) coset of $\mathrm{Aut}(\mathbf{C})$. When $\mathrm{Iso}(\mathbf{C}, \mathbf{D})$ is non-empty, $|\mathrm{Iso}(\mathbf{C}, \mathbf{D})| = |\mathrm{Aut}(\mathbf{C})|$. Our main technical lemma is the following, obtained by adapting certain results in [6, 15] to the symmetric matrix setting. Its proof can be found in the full version of this paper.

▶ **Lemma 6.** *Let $\mathbf{C} = (C_1, \ldots, C_8) \in \mathrm{S}(\ell, q)^8$ be a random symmetric matrix tuple. Then we have $|\mathrm{Aut}(\mathbf{C})| \leq q^\ell$ for all but at most $\frac{1}{q^{\Omega(\ell)}}$ fraction of such $\mathbf{C}$.*

Given this lemma, we can use Theorem 4 to decide whether $\mathbf{C}$ and $\mathbf{D}$ are isometric, and if so, compute $\mathrm{Iso}(\mathbf{C}, \mathbf{D})$ represented as a coset in $\mathrm{GL}(\ell, q)$. By Lemma 6, for all but at most $\frac{1}{q^{\Omega(\ell)}}$ fraction of $\mathbf{C}$, $|\mathrm{Iso}(\mathbf{C}, \mathbf{D})| \leq q^\ell \leq q^n$. With $\mathrm{Iso}(\mathbf{C}, \mathbf{D})$ as a coset at hand, we can enumerate all elements in $\mathrm{Aut}(\mathbf{C})$ by the standard recursive closure algorithm [16] and therefore all elements in $\mathrm{Iso}(\mathbf{C}, \mathbf{D})$. We then either conclude that $|\mathrm{Iso}(\mathbf{C}, \mathbf{D})| > q^n$, or have all $\mathrm{Iso}(\mathbf{C}, \mathbf{D})$ at hand. In the former case we conclude that $\mathbf{C}$ does not satisfy the required generic condition. In the latter case, we enumerate $R \in \mathrm{Iso}(\mathbf{C}, \mathbf{D})$, and check whether $T_2 = \begin{bmatrix} I_r & 0 \\ 0 & R \end{bmatrix}$ is an isomorphism from f to g_1.

The algorithm outline. We now summarise the above steps in the following algorithm outline. In the following we assume that $n \gg 8$; otherwise we can use the brute-force algorithm.

Input Cubic forms $f, g \in \mathbb{F}_q[x_1, \ldots, x_n]$.

Output One of the following: (1) "f does not satisfy the generic condition"; (2) "f and g are not isomorphic"; (3) an isomorphism $T \in \mathrm{GL}(n, q)$ sending g to f.

```
Algorithm outline 1. Set $r = 8$, and $\ell = n - r$.
  2. Compute $W = \{T_1\} \subseteq \mathrm{GL}(n, q)$ using the procedure described in Step 1.
     // Every $T \in \mathrm{GL}(n, q)$ can be written as $T_1T_2$ where $T_2$ is of the form in Equation 2.
  3. For every $T_1 \in W$, do the following:
     a. $h :\leftarrow g \circ T_1$.
     b. For $i \in [\ell]$, $y_i \leftarrow x_{r+i}$.
     c. For $i \in [r]$, let $C_i \in \mathrm{S}(\ell, q)$ be defined in Equation 4. Let $D_i \in \mathrm{S}(\ell, q)$ be defined
        from $h$ in the same way. Let $\mathbf{C} = (C_1, \ldots, C_r)$, and $\mathbf{D} = (D_1, \ldots, D_r)$.
     d. Use Theorem 4 to decide whether $\mathbf{C}$ and $\mathbf{D}$ are isometric. If not, break from the
        loop. If so, compute one isometry $R$.
     e. Use Theorem 4 to compute a generating set of $\mathrm{Aut}(\mathbf{C})$. Use the recursive closure
        algorithm to enumerate $\mathrm{Aut}(\mathbf{C})$. During the enumertion, if $|\mathrm{Aut}(\mathbf{C})| > q^\ell$, report
        "$f$ does not satisfy the generic condition." Otherwise, we have the whole $\mathrm{Aut}(\mathbf{C})$ at
        hand, which is of size $\leq q^\ell$.
     f. Given $R$ from Line 3d and $\mathrm{Aut}(\mathbf{C})$ from Line 3e, the whole set $\mathrm{Iso}(\mathbf{C}, \mathbf{D})$ can be
        computed. For every $R \in \mathrm{Iso}(\mathbf{C}, \mathbf{D})$, check whether $T_2 = \begin{bmatrix} I_r & 0 \\ 0 & R \end{bmatrix}$ sends $h$ to $f$. If
        so, return $T = T_1T_2$ as an isomorphism sending $g$ to $f$.
  4. Return that "$f$ and $g$ are not isomorphic".
```

Correctness and timing analyses. The correctness of the algorithm relies on the simple fact that if f satisfies the genericity condition, and f and g are isomorphic via some $T \in \mathrm{GL}(n, q)$, then this T can be decomposed as T_1T_2 for some $T_1 \in W$ from Line 2. Then by the analysis in Step 2, $T_2 = \begin{bmatrix} I_r & 0 \\ 0 & R \end{bmatrix}$ where $R \in \mathrm{Iso}(\mathbf{C}, \mathbf{D})$. When f satisfies the genericity condition, $\mathrm{Iso}(\mathbf{C}, \mathbf{D})$ will be enumerated, so this R will surely be encountered.

To estimate the time complexity of the algorithm, note that $|W| \leq q^{O(rn)}$, and $|\mathrm{Iso}(\mathbf{C}, \mathbf{D})| \leq q^{\ell} = q^{n-r}$. As other steps are performed in time $\mathrm{poly}(n, m, q)$, enumerating over W and $\mathrm{Iso}(\mathbf{C}, \mathbf{D})$ dominates the time complexity. Recall that $r = 8$. So the total time complexity is upper bounded by $q^{O(n)}$. ◀

3.2 Trilinear form equivalence and algebra isomorphism

We describe our results on trilinear form equivalence and algebra isomorphism, and leave the modifications required to achieve these results in the full version of this paper.

Trilinear form equivalence. The trilinear form equivalence problem was stated in Section 1.2. In algorithms, a trilinear form f is naturally represented as a 3-way array $\mathtt{A} = (a_{i,j,k})$ where $a_{i,j,k} = f(e_i, e_j, e_k)$. A random trilinear form over $\mathbb{F}_q$ denotes the setting when $\alpha_{i,j,k}$ are independently sampled from $\mathbb{F}_q$ uniformly at random.

▶ **Theorem 7.** *Let $f : \mathbb{F}_q^n \times \mathbb{F}_q^n \times \mathbb{F}_q^n \to \mathbb{F}_q$ be a random trilinear form, and let $g : \mathbb{F}_q^n \times \mathbb{F}_q^n \times \mathbb{F}_q^n \to \mathbb{F}_q$ be an arbitrary trilinear form. There exists a deterministic algorithm that decides whether f and g are equivalent in time $q^{O(n)}$, for all but at most $\frac{1}{q^{\Omega(n)}}$ fraction of f.*

Algebra isomorphism. Let V be a vector space. An algebra is a bilinear map $* : V \times V \to V$. This bilinear map $*$ is considered as the product. Algebras most studied are those with certain conditions on the product, including unital ($\exists v \in V$ such that $\forall u \in V$, $v * u = u$), associative ($(u * v) * w = u * (v * w)$), and commutative ($u * v = v * u$). The authors of [1, 2] study algebras satisfying these conditions. Here we consider algebras without such restrictions. Two algebras $*, \cdot : V \times V \to V$ are *isomorphic*, if there exists $T \in \mathrm{GL}(V)$, such that $\forall u, v \in V$, $T(u) * T(v) = T(u \cdot v)$. As customary in computational algebra, an algebra is represented by its structure constants, i.e. suppose $V \cong \mathbb{F}^n$, and fix a basis $\{e_1, \ldots, e_n\}$. Then $e_i * e_j = \sum_{k \in [n]} \alpha_{i,j,k} e_k$, and this 3-way array $\mathtt{A} = (\alpha_{i,j,k})$ records the structure constants of the algebra with product $*$. A random algebra over $\mathbb{F}_q$ denotes the setting when $\alpha_{i,j,k}$ are independently sampled from $\mathbb{F}_q$ uniformly at random.

▶ **Theorem 8.** *Let $f : \mathbb{F}_q^n \times \mathbb{F}_q^n \to \mathbb{F}_q^n$ be a random algebra, and let $g : \mathbb{F}_q^n \times \mathbb{F}_q^n \to \mathbb{F}_q^n$ be an arbitrary algebra. There exists a deterministic algorithm that decides whether f and g are isomorphic in time $q^{O(n)}$, for all but at most $\frac{1}{q^{\Omega(n)}}$ fraction of f.*

4 Complexity of symmetric and alternating trilinear form equivalence

As mentioned in Section 1.4, the proof of Theorem 2 follows by showing that symmetric and alternating trilinear form equivalence are TI-hard (recall Definition 3). In the following we focus on the alternating case. The symmetric case can be tackled in a straightforward way, by starting from the TI-complete problem of symmetric matrix tuple pseudo-isometry, from [11, Theorem B], and modifying the alternating gadget to a symmetric one.

▶ **Proposition 9.** *The alternating trilinear form equivalence problem is* TI*-hard.*

Proof. The starting TI-complete problem. We use the following TI-complete problem from [11]. Let $\mathbf{A} = (A_1, \ldots, A_m), \mathbf{B} = (B_1, \ldots, B_m) \in \Lambda(n, \mathbb{F})^m$ be two tuples of alternating matrices. We say that $\mathbf{A}$ and $\mathbf{B}$ are pseudo-isometric, if there exist $C \in \mathrm{GL}(n, \mathbb{F})$ and $D = (d_{i,j}) \in \mathrm{GL}(m, \mathbb{F})$, such that for any $i \in [m]$, $C^{\mathrm{t}}(\sum_{j\in[m]} d_{i,j}A_j)C = B_i$. By [11, Theorem B], the alternating matrix tuple pseudo-isometry problem is TI-complete. Without loss of generality, we assume that $\dim(\langle A_i\rangle) = \dim(\langle B_i\rangle)$, as if not, then they cannot be pseudo-isometric, and this dimension condition is easily checked.

An alternating trilinear form $\phi : \mathbb{F}^n \times \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F}$ naturally corresponds to a 3-way array $\mathtt{A} = (a_{i,j,k}) \in \mathrm{M}(n \times n \times n, \mathbb{F})$, where $a_{i,j,k} = \phi(e_i, e_j, e_k)$. Then $\mathtt{A}$ is also alternating, i.e. $a_{i,j,k} = 0$ if $i = j$ or $i = k$ or $j = k$, and $a_{i,j,k} = \mathrm{sgn}(\sigma)a_{\sigma(i),\sigma(j),\sigma(k)}$ for any $\sigma \in \mathrm{S}_3$. So in the following, we present a construction of an alternating 3-way array from an alternating matrix tuple, in such a way that two alternating matrix tuples are pseudo-isometric if and only if the corresponding alternating trilinear forms are equivalent.

Constructing alternating 3-way arrays from alternating matrix tuples. Given $\mathbf{A} = (A_1, \ldots, A_m) \in \Lambda(n, \mathbb{F})^m$, we first build the $n \times n \times m$ tensor $\mathtt{A}$ which has $A_1, \ldots, A_m$ as its frontal slices. Then we will use essentially the following construction twice in succession. We will give two viewpoints on this construction: one algebraic, in terms of trilinear forms, and another "matricial", in terms of 3-way arrays. Different readers may prefer one viewpoint over the other; our opinion is that the algebraic view makes it easier to verify the alternating property while the matricial view makes it easier to verify the reduction. We thank an anonymous review for the suggestion of the algebraic viewpoint. The construction is, in some sense, the 3-tensor analogue of taking an ordinary matrix A and building the alternating matrix $\begin{bmatrix} 0 & A \\ -A^{\mathrm{t}} & 0 \end{bmatrix}$.

Notation: Just as the transpose acts on matrices by $(A^{\mathrm{t}})_{i,j} = A_{j,i}$, for a 3-tensor $\mathtt{A}$, we have six possible "transposes" corresponding to the six permutations of the three coordinates. Given $\sigma \in S_3$, we write $\mathtt{A}^\sigma$ for the 3-tensor defined by $(\mathtt{A}^\sigma)_{i_1,i_2,i_3} = \mathtt{A}_{i_{\sigma(1)},i_{\sigma(2)},i_{\sigma(3)}}$. Similarly, given a trilinear form $A(x, y, z) = \sum_{i,j,k} a_{i,j,k}x_iy_jz_k$, we have $A^\sigma(x, y, z) = A((x, y, z)^\sigma)$.

Given a 3-way array $\mathtt{A} \in \mathrm{M}(n \times m \times d, \mathbb{F})$, we will make use of $\mathtt{A}^{(23)}$ and $\mathtt{A}^{(13)}$:

- $\mathtt{A}^{(23)}$ is $n \times d \times m$ and has $\mathtt{A}^{(23)}_{i,j,k} = \mathtt{A}_{i,k,j}$. Equivalently, the k-th frontal slice of $\mathtt{A}^{(23)}$ is the k-th vertical slice of $\mathtt{A}$.
- $\mathtt{A}^{(13)}$ is $d \times m \times n$ and has $\mathtt{A}^{(13)}_{i,j,k} = \mathtt{A}_{k,j,i}$. Equivalently, the k-th frontal slice of $\mathtt{A}^{(13)}$ is the transpose of the k-th horizontal slice of $\mathtt{A}$.

▶ **Example 10** (Running example)**.** Let us examine a simple example as follows. Let $\mathbf{A} = (A) \in \Lambda(2, \mathbb{F})^1$, where $A = \begin{bmatrix} 0 & a \\ -a & 0 \end{bmatrix}$. Then $\mathtt{A} = (A)$; $\mathtt{A}^{(23)} = (A_1', A_2') \in \mathrm{M}(2 \times 1 \times 2, \mathbb{F})$, where $A_1' = \begin{bmatrix} 0 \\ -a \end{bmatrix}$, and $A_2' = \begin{bmatrix} a \\ 0 \end{bmatrix}$; $\mathtt{A}^{(13)} = (A_1'', A_2'') \in \mathrm{M}(1 \times 2 \times 2, \mathbb{F})$, where $A_1'' = \begin{bmatrix} 0 & a \end{bmatrix}$, and $A_2'' = \begin{bmatrix} -a & 0 \end{bmatrix}$.

From the above $\mathtt{A}$, $\mathtt{A}^{(23)}$, and $\mathtt{A}^{(13)}$, we construct $\tilde{\mathtt{A}} \in \mathrm{M}((n + m) \times (n + m) \times (n + m), \mathbb{F})$ as follows. We divide $\tilde{\mathtt{A}}$ into the following eight blocks. Set $\tilde{\mathtt{A}} = (\tilde{\mathtt{A}}_1, \tilde{\mathtt{A}}_2)$ (two block frontal slices) where $\tilde{\mathtt{A}}_1 = \begin{bmatrix} 0_{n\times n\times n} & \mathtt{A}^{(23)} \\ \mathtt{A}^{(13)} & 0 \end{bmatrix}$, and $\tilde{\mathtt{A}}_2 = \begin{bmatrix} -\mathtt{A} & 0 \\ 0 & 0_{m\times m\times m} \end{bmatrix}$, where $0_{n\times n\times n}$ indicates the $n \times n \times n$ zero tensor, and analogously for $0_{m\times m\times m}$ (the remaining sizes can be determined from these and the fact that $\mathtt{A}$ is $n \times n \times m$).

The corresponding construction on trilinear forms is as follows. The original trilinear form is $A(x,y,z)=\sum_{i,j\in[n],k\in[m]} a_{i,j,k}x_iy_jz_k$, where $x=(x_1,\dots,x_n)$, $y=(y_1,\dots,y_n)$, and $z=(z_1,\dots,z_m)$, and we have $A(x,y,z)=-A(y,x,z)$. The new trilinear form will be $\tilde{A}(x',y',z')$, where

$$\begin{aligned}
x'=(x^{(1)},x^{(2)}) &= (x_1^{(1)},\dots,x_n^{(1)},x_1^{(2)},\dots,x_m^{(2)})\\
y'=(y^{(1)},y^{(2)}) &= (y_1^{(1)},\dots,y_n^{(1)},y_1^{(2)},\dots,y_m^{(2)})\\
z'=(z^{(1)},z^{(2)}) &= (z_1^{(1)},\dots,z_n^{(1)},z_1^{(2)},\dots,z_m^{(2)}).
\end{aligned}$$

This new form will satisfy $\tilde{A}(x',y',z')=\sum_{i,j,k\in[n+m]}\tilde{a}_{i,j,k}x'_iy'_jz'_k$. Let us unravel what this looks like from the above description of $\tilde{\mathtt{A}}$. We have

$$\begin{aligned}
\tilde{A}(x',y',z') &= \sum_{i\in[n],j\in[m],k\in[n]}(\tilde{\mathtt{A}}_1)_{i,n+j,k}x'_iy'_{n+j}z'_k+\sum_{i\in[m],j,k\in[n]}(\tilde{\mathtt{A}}_1)_{n+i,j,k}x'_{n+i}y'_jz'_k\\
&\quad+\sum_{i,j\in[n],k\in[m]}(\tilde{\mathtt{A}}_2)_{i,j,k}x'_iy'_jz'_{n+k}\\
&= \sum_{i\in[n],j\in[m],k\in[n]}\mathtt{A}^{(23)}_{i,j,k}x'_iy'_{n+j}z'_k+\sum_{i\in[m],j,k\in[n]}\mathtt{A}^{(13)}_{i,j,k}x'_{n+i}y'_jz'_k\\
&\quad-\sum_{i,j\in[n],k\in[m]}\mathtt{A}_{i,j,k}x'_iy'_jz'_{n+k}\\
&= \sum_{i\in[n],j\in[m],k\in[n]}\mathtt{A}_{i,k,j}x'_iy'_{n+j}z'_k+\sum_{i\in[m],j,k\in[n]}\mathtt{A}_{k,j,i}x'_{n+i}y'_jz'_k\\
&\quad-\sum_{i,j\in[n],k\in[m]}\mathtt{A}_{i,j,k}x'_iy'_jz'_{n+k}\\
&= A(x^{(1)},z^{(1)},y^{(2)})+A(z^{(1)},y^{(1)},x^{(2)})-A(x^{(1)},y^{(1)},z^{(2)}).
\end{aligned}$$

From this formula, and the fact that $A(x,y,z)=-A(y,x,z)$, we can now more easily verify that $\tilde{A}$ is alternating in all three arguments. Since the permutations (13) and (23) generate S_3, it suffices to verify it for these two. We have

$$\begin{aligned}
\tilde{A}^{(13)}(x',y',z') &= \tilde{A}(z',y',x')\\
&= A(z^{(1)},x^{(1)},y^{(2)})+A(x^{(1)},y^{(1)},z^{(2)})-A(z^{(1)},y^{(1)},x^{(2)})\\
&= -A(x^{(1)},z^{(1)},y^{(2)})+A(x^{(1)},y^{(1)},z^{(2)})-A(z^{(1)},y^{(1)},x^{(2)})\\
&= -\tilde{A}(x',y',z').
\end{aligned}$$

Similarly, we have:

$$\begin{aligned}
\tilde{A}^{(23)}(x',y',z') &= \tilde{A}(x',z',y')\\
&= A(x^{(1)},y^{(1)},z^{(2)})+A(y^{(1)},z^{(1)},x^{(2)})-A(x^{(1)},z^{(1)},y^{(2)})\\
&= A(x^{(1)},y^{(1)},z^{(2)})-A(z^{(1)},y^{(1)},x^{(2)})-A(x^{(1)},z^{(1)},y^{(2)})\\
&= -\tilde{A}(x',y',z'),
\end{aligned}$$

as claimed.

▶ **Example 11** (Running example, continued from Example 10)**.** We can write out $\tilde{\mathtt{A}}$ in this case explicitly. The first block frontal slice $\tilde{\mathtt{A}}_1$ is $3\times3\times2$, consisting of the two frontal slices

$$\left(\begin{array}{cc:c} 0 & 0 & 0\\ 0 & 0 & -a\\ \hdashline 0 & a & 0 \end{array}\right)\quad\text{and}\quad\left(\begin{array}{cc:c} 0 & 0 & a\\ 0 & 0 & 0\\ \hdashline -a & 0 & 0 \end{array}\right)$$

while the second block frontal slice $\tilde{\mathtt{A}}_2$ is the $3 \times 3 \times 1$ matrix

$$\left(\begin{array}{cc:c} 0 & -a & 0 \\ a & 0 & 0 \\ \hdashline 0 & 0 & 0 \end{array}\right)$$

It can be verified easily that $\tilde{\mathtt{A}} = (a_{i,j,k})$ is alternating: the nonzero entries are $a_{2,3,1} = -a$, $a_{3,2,1} = a$, $a_{1,3,2} = a$, $a_{3,1,2} = -a$, $a_{1,2,3} = -a$, and $a_{2,1,3} = a$, which are consistent with the signs of the permutations.

The gadget construction. We now describe the gadget construction. The gadget can be described as a block 3-way array as follows. Construct a 3-way array $\mathtt{G}$ of size $(n+1)^2 \times (n+1)^2 \times (n+m)$ over $\mathbb{F}$ as follows. For $i \in [n]$, the ith frontal slice of $\mathtt{G}$ is

$$\begin{bmatrix} 0 & 0 & \dots & 0 & I_{n+1} & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ -I_{n+1} & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \end{bmatrix},$$

where 0 here denotes the $(n+1) \times (n+1)$ all-zero matrix, I_{n+1} is at the $(1, i+1)$th block position, and $-I_{n+1}$ is at the $(i+1, 1)$th block position. For $n+1 \le i \le n+m$, the ith frontal slice of $\mathtt{G}$ is the all-zero matrix. We also need the following 3-way arrays derived from $\mathtt{G}$. We will use $\mathtt{G}^{(13)}$ and $\mathtt{G}^{(23)}$. Note that $\mathtt{G}^{(13)}$ is of size $(n+m) \times (n+1)^2 \times (n+1)^2$, and its ith horizontal slice is the ith frontal slice of $\mathtt{G}$. Similarly, $\mathtt{G}^{(23)}$ is of size $(n+1)^2 \times (n+m) \times (n+1)^2$, and its jth vertical slice is the jth frontal slice of $\mathtt{G}$.

Finally, construct a 3-tensor $\hat{\mathtt{A}}$ as follows. It consists of the two block frontal slices

$$\begin{bmatrix} \tilde{\mathtt{A}} & 0 \\ 0 & -\mathtt{G} \end{bmatrix} \text{ and } \begin{bmatrix} 0 & \mathtt{G}^{(13)} \\ \mathtt{G}^{(23)} & 0 \end{bmatrix}.$$

To see how this all fits together, let $\mathtt{G}_1$ be the $(n+1)^2 \times (n+1)^2 \times n$ tensor consisting of the first n frontal slices of $\mathtt{G}$ (these are the only nonzero frontal slices of $\mathtt{G}$). Then we may view $\hat{\mathtt{A}}$ as having three block frontal slices, namely:

$$\begin{bmatrix} 0_{n\times n\times n} & \mathtt{A}^{(23)} & 0 \\ \mathtt{A}^{(13)} & 0_{m\times m\times n} & 0 \\ 0 & 0 & -\mathtt{G}_1 \end{bmatrix}, \begin{bmatrix} -\mathtt{A} & 0 & 0 \\ 0 & 0_{m\times m\times m} & 0 \\ 0 & 0 & 0_{(n+1)^2\times(n+1)^2\times m} \end{bmatrix},$$

and

$$\begin{bmatrix} 0_{n\times n\times(n+1)^2} & 0 & \mathtt{G}_1^{(13)} \\ 0 & 0_{m\times m\times(n+1)^2} & 0 \\ \mathtt{G}_1^{(23)} & 0 & 0 \end{bmatrix}.$$

We claim that $\hat{\mathtt{A}}$ is alternating. To verify this is straightforward but somewhat tedious. So we use the following example from which a complete proof can be extracted easily.

▶ **Example 12** (Running example, continued from Example 11)**.** Let $\mathtt{A}$ be the $2 \times 2 \times 1$ tensor with alternating frontal slice $A = \begin{bmatrix} 0 & a \\ -a & 0 \end{bmatrix}$. In particular, $n = 2, m = 1$, so $\mathtt{G}$ will have size $(n+1)^2 \times (n+1)^2 \times (n+m) = 9 \times 9 \times 3$, and $\mathtt{A}$ will have size $n + m + (n+1)^2 = 12$ in all three directions. We will write out the first $n + m = 3$ frontal slices explicitly, as those are the only ones involving $\mathtt{A}$, and leave the last 9 (involving only transposes of $\mathtt{G}_1$) unwritten.

$$\left(\begin{array}{cc|c|ccc}
0 & 0 & 0 & & & \\
0 & 0 & -a & & & \\ \hline
0 & a & 0 & & & \\ \hline
 & & & 0_3 & I_3 & 0 \\
 & & & -I_3 & 0_3 & 0 \\
 & & & 0 & 0 & 0_3
\end{array}\right), \left(\begin{array}{cc|c|ccc}
0 & 0 & a & & & \\
0 & 0 & 0 & & & \\ \hline
-a & 0 & 0 & & & \\ \hline
 & & & 0_3 & 0 & I_3 \\
 & & & 0 & 0_3 & 0 \\
 & & & -I_3 & 0 & 0_3
\end{array}\right),$$

and $\left(\begin{array}{cc|c|ccc}
0 & a & 0 & & & \\
-a & 0 & 0 & & & \\ \hline
0 & 0 & 0 & & & \\ \hline
 & & & 0_3 & 0 & 0 \\
 & & & 0 & 0_3 & 0 \\
 & & & 0 & 0 & 0_3
\end{array}\right)$

and the remaining 9 frontal slices look like

$$\left(\begin{array}{cc|c|ccc}
0 & 0 & 0 & & & \\
0 & 0 & 0 & & \mathtt{G}_1^{(13)} & \\ \hline
0 & 0 & 0 & & 0_{1\times 9\times 9} & \\ \hline
 & & & 0_{3\times 3\times 9} & 0 & 0 \\
\multicolumn{2}{c|}{\mathtt{G}_1^{(23)}} & 0_{9\times 1\times 9} & 0 & 0_{3\times 3\times 9} & 0 \\
 & & & 0 & 0 & 0_{3\times 3\times 9}
\end{array}\right)$$

Since the a's only appear in positions with the same indices as they did in $\tilde{\mathtt{A}}$ (see Example 11), that portion is still alternating. For the $\mathtt{G}$ parts, note that the identity matrices in the first three frontal slices, when having their indices transposed, end up either in the $\mathtt{G}_1^{(13)}$ portion or the $\mathtt{G}_1^{(23)}$ portion, with appropriate signs.

Proof of correctness. Let $\mathbf{A}, \mathbf{B} \in \Lambda(n, \mathbb{F})^m$. Let $\hat{\mathtt{A}} = \left(\begin{bmatrix} \tilde{\mathtt{A}} & 0 \\ 0 & -\mathtt{G} \end{bmatrix}, \begin{bmatrix} 0 & \mathtt{G}^{(13)} \\ \mathtt{G}^{(23)} & 0 \end{bmatrix} \right)$, $\hat{\mathtt{B}} = \left(\begin{bmatrix} \tilde{\mathtt{B}} & 0 \\ 0 & -\mathtt{G} \end{bmatrix}, \begin{bmatrix} 0 & \mathtt{G}^{(13)} \\ \mathtt{G}^{(23)} & 0 \end{bmatrix} \right) \in \mathrm{M}((n+m+(n+1)^2) \times (n+m+(n+1)^2) \times (n+m+(n+1)^2), \mathbb{F})$ be constructed from $\mathbf{A}$ and $\mathbf{B}$ using the procedure above, respectively.

We claim that $\mathbf{A}$ and $\mathbf{B}$ are pseudo-isometric if and only if $\hat{\mathtt{A}}$ and $\hat{\mathtt{B}}$ are equivalent as trilinear forms.

The only if direction. Suppose $P^{\mathrm{t}} \mathbf{A} P = \mathbf{B}^Q$ for some $P \in \mathrm{GL}(n, \mathbb{F})$ and $Q \in \mathrm{GL}(m, \mathbb{F})$.

We will construct a trilinear form equivalence from $\hat{\mathtt{A}}$ to $\hat{\mathtt{B}}$ of the form $S = \begin{bmatrix} P & 0 & 0 \\ 0 & Q^{-1} & 0 \\ 0 & 0 & R \end{bmatrix} \in \mathrm{GL}(n + m + (n+1)^2, \mathbb{F})$, where $R \in \mathrm{GL}((n+1)^2, \mathbb{F})$ is to be determined later on.

Recall that $\hat{\mathtt{A}} = (\begin{bmatrix} \tilde{\mathtt{A}} & 0 \\ 0 & -\mathtt{G} \end{bmatrix}, \begin{bmatrix} 0 & \mathtt{G}^{(13)} \\ \mathtt{G}^{(23)} & 0 \end{bmatrix})$, $\hat{\mathtt{B}} = (\begin{bmatrix} \tilde{\mathtt{B}} & 0 \\ 0 & -\mathtt{G} \end{bmatrix}, \begin{bmatrix} 0 & \mathtt{G}^{(13)} \\ \mathtt{G}^{(23)} & 0 \end{bmatrix})$. It can be verified that the action of S sends $\tilde{\mathtt{A}}$ to $\tilde{\mathtt{B}}$. It remains to show that, by choosing an appropriate R, the action of S also sends $\mathtt{G}$ to $\mathtt{G}$.

Let $\mathtt{G}_1$ be the first n frontal slices of $\mathtt{G}$, and $\mathtt{G}_2$ the last m frontal slices from $\mathtt{G}$. Then the action of S sends $\mathtt{G}_1$ to $R^{\mathrm{t}}\mathtt{G}_1^P R$, and $\mathtt{G}_2$ to $R^{\mathrm{t}}\mathtt{G}_2^{Q^{-1}} R$. Since $\mathtt{G}_2$ is all-zero, the action of S on $\mathtt{G}_2$ results in an all-zero tensor, so we have $R^{\mathrm{t}}\mathtt{G}_2^{Q^{-1}} R = \mathtt{G}_2$.

We then turn to $\mathtt{G}_1$. For $i \in [n+1]$, consider the ith horizontal slice of $\mathtt{G}_1$, which is of the form $H_i = \begin{bmatrix} 0 & B_{1,i} & B_{2,i} & \dots & B_{n,i} \end{bmatrix}$, where 0 denotes the $n \times (n+1)$ all-zero matrix, and $B_{j,i}$ is the $n \times (n+1)$ elementary matrix with the (j,i)th entry being 1, and other entries being 0. Note that those non-zero entries of H_i are in the $(k(n+1)+i)$th columns, for $k \in [n]$. Let $P^{\mathrm{t}} = \begin{bmatrix} p_1 & \dots & p_n \end{bmatrix}$, where p_i is the ith column of P^{t}. Then P acts on H_i from the left, which yields $P^{\mathrm{t}}H_i = \begin{bmatrix} 0 & P_{1,i} & \dots & P_{n,i} \end{bmatrix}$, where $P_{j,i}$ denotes the $n \times (n+1)$ matrix with the ith column being p_j, and the other columns being 0.

Let us first set $R = \begin{bmatrix} I_{n+1} & 0 \\ 0 & \hat{R} \end{bmatrix}$, where $\hat{R}$ is to be determined later on. Then the left action of R on $\mathtt{G}_1$ preserves H_i through I_{n+1}. The right action of R on $\mathtt{G}_1$ translates to the right action of $\hat{R}$ on H_i. To send $P^{\mathrm{t}}H_i$ back to H_i, $\hat{R}$ needs to act on those $(k(n+1)+i)$th columns of H_i, $i \in [n+1]$, as P^{-1}. Note that for H_i and H_j, $i \neq j$, those columns with non-zero entries are disjoint. This gives $\hat{R}$ the freedom to handle different H_i's separately. In other words, $\hat{R}$ can be set as $P^{-1} \otimes I_{n+1}$. This ensures that for every H_i, $P^{\mathrm{t}}H_i\hat{R} = H_i$. To summarize, we have $R^{\mathrm{t}}\mathtt{G}_1^P R = \mathtt{G}_1$, and this concludes the proof for the only if direction.

The if direction. Suppose $\hat{\mathtt{A}}$ and $\hat{\mathtt{B}}$ are isomorphic as trilinear forms via $P \in \mathrm{GL}(n+m+(n+1)^2, \mathbb{F})$. Set $P = \begin{bmatrix} P_{1,1} & P_{1,2} & P_{1,3} \\ P_{2,1} & P_{2,2} & P_{2,3} \\ P_{3,1} & P_{3,2} & P_{3,3} \end{bmatrix}$, where $P_{1,1}$ is of size $n \times n$, $P_{2,2}$ is of size $m \times m$, and $P_{3,3}$ is of size $(n+1)^2 \times (n+1)^2$. Consider the ranks of the frontal slices of $\hat{\mathtt{A}}$.

- The ranks of the first n frontal slices are in $[2(n+1), 4n]$. This is because a frontal slice in this range consists of two copies of vertical slices of $\mathtt{A}$ (whose ranks are between $[0, n-1]$ due to the alternating condition), and one frontal slice of $\mathtt{G}$ (whose ranks are of $2(n+1)$).
- The ranks of the $n+1$ to $n+m$ frontal slices are in $[0, n]$. This is because a frontal slice in this range consists of only just one frontal slice of $\mathtt{A}$.
- The ranks of the last $n(n+1)$ vertical slices are in $[0, 2n]$. This is because a frontal slice in this range consists of two copies of horizontal slices of $\mathtt{G}$ (whose ranks are either n or 1; see e.g. the form of H_i in the proof of the only if direction).

By the discussions above, we claim that that P must be of the form $\begin{bmatrix} P_{1,1} & 0 & 0 \\ P_{2,1} & P_{2,2} & P_{2,3} \\ P_{3,1} & P_{3,2} & P_{3,3} \end{bmatrix}$. To see this, for the sake of contradiction, suppose there are non-zero entries in $P_{1,2}$ or $P_{1,3}$. Then a non-trivial linear combination of the first n frontal slices is added to one of the last $(m+(n+1)^2)$ frontal slices. This implies that for this slice, the lower-right $(n+1)^2 \times (n+1)^2$ submatrix is of the form $\begin{bmatrix} 0 & a_1 I_{n+1} & a_2 I_{n+1} & \dots & a_n I_{n+1} \\ -a_1 I_{n+1} & 0 & 0 & \dots & 0 \\ -a_2 I_{n+1} & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -a_n I_{n+1} & 0 & 0 & \dots & 0 \end{bmatrix}$, where one of $a_i \in \mathbb{F}$ is non-zero. Then this slice is of rank $\geq 2(n+1)$, which is unchanged by left (resp. right) multiplying P^{t} (resp. P), so it cannot be equal to the corresponding slice of $\hat{\mathtt{B}}$ which is of rank $\leq 2n$. We then arrived at the desired contradiction.

Now consider the action of such P on the $n+1$ to $n+m$ frontal slices. Note that these slices are of the form $\begin{bmatrix} A_i & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$. (Recall that the last m slices of $\mathtt{G}$ are all-zero matrices.) Then we have $\begin{bmatrix} P_{1,1}^{\mathrm{t}} & P_{2,1}^{\mathrm{t}} & P_{3,1}^{\mathrm{t}} \\ 0 & P_{2,2}^{\mathrm{t}} & P_{3,2}^{\mathrm{t}} \\ 0 & P_{2,3}^{\mathrm{t}} & P_{3,3}^{\mathrm{t}} \end{bmatrix} \begin{bmatrix} A_i & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_{1,1} & 0 & 0 \\ P_{2,1} & P_{2,2} & P_{2,3} \\ P_{3,1} & P_{3,2} & P_{3,3} \end{bmatrix} = \begin{bmatrix} P_{1,1}^{\mathrm{t}} A_i P_{1,1} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$. Since $P^{\mathrm{t}} \hat{\mathtt{A}}^P P = \hat{\mathtt{B}}$, we have $P^{\mathrm{t}} \hat{\mathtt{A}} P = \hat{\mathtt{B}}^{P^{-1}}$. Observe that for the upper-left $n \times n$ submatrices of the frontal slices of $\hat{\mathtt{B}}$, P^{-1} simply performs a linear combination of B_i's. It follows that every $P_{1,1}^{\mathrm{t}} A_i P_{1,1}$ is in the linear span of B_i. Since we assumed $\dim(\langle A_i \rangle) = \dim(\langle B_i \rangle)$, we have that $\mathbf{A}$ and $\mathbf{B}$ are pseudo-isometric. This concludes the proof of Proposition 9. ◀

References

1 Manindra Agrawal and Nitin Saxena. Automorphisms of finite rings and applications to complexity of problems. In *STACS 2005, 22nd Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*, pages 1–17, 2005. doi:10.1007/978-3-540-31856-9_1.

2 Manindra Agrawal and Nitin Saxena. Equivalence of $\mathbb{F}$-algebras and cubic forms. In *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*, pages 115–126, 2006. doi:10.1007/11672142_8.

3 Jérémy Berthomieu, Jean-Charles Faugère, and Ludovic Perret. Polynomial-time algorithms for quadratic isomorphism of polynomials: The regular case. *J. Complexity*, 31(4):590–616, 2015. doi:10.1016/j.jco.2015.04.001.

4 Charles Bouillaguet. *Etudes d'hypothèses algorithmiques et attaques de primitives cryptographiques.* PhD thesis, PhD thesis, Université Paris-Diderot–École Normale Supérieure, 2011.

5 Charles Bouillaguet, Jean-Charles Faugère, Pierre-Alain Fouque, and Ludovic Perret. Practical cryptanalysis of the identification scheme based on the isomorphism of polynomial with one secret problem. In *International Workshop on Public Key Cryptography*, pages 473–493. Springer, 2011. doi:10.1007/978-3-642-19379-8_29.

6 Peter A. Brooksbank, Yinan Li, Youming Qiao, and James B. Wilson. Improved algorithms for alternating matrix space isometry: From theory to practice. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 26:1–26:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.26.

7 Peter A. Brooksbank and James B. Wilson. Computing isometry groups of Hermitian maps. *Trans. Amer. Math. Soc.*, 364:1975–1996, 2012. doi:10.1090/S0002-9947-2011-05388-2.

8 Vyacheslav Futorny, Joshua A. Grochow, and Vladimir V. Sergeichuk. Wildness for tensors. *Lin. Alg. Appl.*, 566:212–244, 2019. doi:10.1016/j.laa.2018.12.022.

9 Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity for all languages in NP have zero-knowledge proof systems. *J. ACM*, 38(3):691–729, 1991. doi:10.1145/116825.116852.

10 Joshua A. Grochow and Youming Qiao. Algorithms for group isomorphism via group extensions and cohomology. *SIAM J. Comput.*, 46(4):1153–1216, 2017. Preliminary version in IEEE Conference on Computational Complexity (CCC) 2014 (DOI:10.1109/CCC.2014.19). Also available as arXiv:1309.1776 [cs.DS] and ECCC Technical Report TR13-123. doi:10.1137/15M1009767.

11 Joshua A. Grochow and Youming Qiao. On the complexity of isomorphism problems for tensors, groups, and polynomials I: Tensor Isomorphism-completeness. In *ITCS*, Leibniz International Proceedings in Informatics (LIPIcs), pages 31:1–31:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ITCS.2021.31.

12 Gábor Ivanyos and Youming Qiao. Algorithms based on *-algebras, and their applications to isomorphism of polynomials with one secret, group isomorphism, and polynomial identity testing. *SIAM J. Comput.*, 48(3):926–963, 2019. doi:10.1137/18M1165682.

13 Neeraj Kayal. Efficient algorithms for some special cases of the polynomial equivalence problem. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1409–1421, 2011. doi:10.1137/1.9781611973082.108.

14 Serge Lang. *Algebra*. Number 211 in Graduate Texts in Mathematics. Springer-Verlag, New York, third enlarged edition, 2002.

15 Yinan Li and Youming Qiao. Linear algebraic analogues of the graph isomorphism problem and the Erdős–Rényi model. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 463–474. IEEE Computer Society, 2017. arXiv:1708.04501, version 2. doi:10.1109/FOCS.2017.49.

16 Eugene M. Luks. Permutation groups and polynomial-time computation. In *Groups and computation (New Brunswick, NJ, 1991)*, volume 11 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 139–175. Amer. Math. Soc., Providence, RI, 1993.

17 Jacques Patarin. Hidden fields equations (HFE) and isomorphisms of polynomials (IP): two new families of asymmetric algorithms. In *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, pages 33–48, 1996. doi:10.1007/3-540-68339-9_4.

18 Nitin Saxena. *Morphisms of rings and applications to complexity*. PhD thesis, Indian Institute of Technology, Kanpur, May 2006. URL: https://www.cse.iitk.ac.in/users/nitin/papers/thesis.pdf.

19 H. Weyl. *The classical groups: their invariants and representations*, volume 1. Princeton University Press, 1997.

20 James B. Wilson. Decomposing p-groups via Jordan algebras. *J. Algebra*, 322:2642–2679, 2009. doi:10.1016/j.jalgebra.2009.07.029.

21 R. Wilson. *The Finite Simple Groups*, volume 251 of *Graduate Texts in Mathematics*. Springer London, 2009.

Geometric Cover with Outliers Removal

Zhengyang Guo
School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore

Yi Li
School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore

Abstract

We study the problem of partial geometric cover, which asks to find the minimum number of geometric objects (unit squares and unit disks in this work) that cover at least $(n - t)$ of n given planar points, where $0 \le t \le n/2$. When $t = 0$, the problem is the classical geometric cover problem, for which many existing works adopt a general framework called the shifting strategy. The shifting strategy is a divide and conquer paradigm which partitions the plane into equal-width strips, applies a local algorithm on each strip and then merges the local solutions with only a small loss on the overall approximation ratio. A challenge to extend the shifting strategy to the case of outliers is to determine the number of outliers in each strip. We develop a shifting strategy incorporating the outlier distribution, which runs in $O(tn \log n)$ time. We also develop local algorithms on strips for the outliers case, improving the running time over previous algorithms, and consequently obtain approximation algorithms to the partial geometric cover.

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

Keywords and phrases Geometric Cover, Unit Square Cover, Unit Disk Cover, Shifting Strategy, Outliers Detection, Computational Geometry

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.39

1 Introduction

The geometric cover with outlier is a generalization of the classic geometric cover problems to the case where there are outliers. As the real-world data usually contain outliers, which can dramatically affect the output of a general geometric cover algorithm, we hope to exclude the outliers along with finding an optimal covering. Given n points in the plane and an integer $0 \le t \le n/2$, the geometric cover with outliers asks to find the minimum number of geometric objects of a given type that cover at least $(n - t)$ points. The uncovered points are referred to as outliers. The geometric objects we consider in this work are the most two common ones, namely, unit squares (side length 1) and unit disks (radius 1). Correspondingly, we call the two problems partial square cover and partial disk cover, denoted by PSC and PDC, respectively. The problems are formally defined as follows.

▶ **Problem 1** (Partial Geometric Covers). *Given a planar point set X of size n and an integer $0 \le t \le n/2$, we define three problems as follows.*

- *Partial Unit Square Cover (PSC) find a minimum number of unit squares that cover at least $(n - t)$ points of X.*
- *Partial Unit Disk Cover (PDC): find a minimum number of unit disks that cover at least $(n - t)$ points of X.*
- *Restricted Partial Unit Disk Cover (RPDC): find a minimum subset of X such that the unit disks centered in the subset cover at least $(n - t)$ points of X.*

The special case of PSC and PDC when $t = 0$ are the classical square cover and disk cover problems, which were motivated by the applications in image processing [25] and wireless networking [23]. Both of them are known to be strongly NP-hard and hence no FPTAS

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 39; pp. 39:1–39:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

exists [18]. Even when the square or disk positions are restricted to the given point set, the problem (which becomes RPDC in the disk case) remains NP-hard [7]. Therefore the research has been considering developing polynomial-time algorithms with a small approximation ratio and time complexity.

For PSC, Gonzalez gives a 2-approximation with time complexity $O(n \log n)$ and a $(1+\epsilon)$-approximation algorithm with time complexity $O(\epsilon^{-1}n^{1/\epsilon})$ in [14]. The techniques can be adapted to PDC, producing a $(2+\epsilon)$-approximation solution in $O(\epsilon^{-2}n^7)$ time [14, 3] and also a $(1+\epsilon)$-approximation in $O(\epsilon^{-2}n^{6\lceil\sqrt{2}/\epsilon\rceil-1})$ time [20]. They suffice for a PTAS but for a constant-factor approximation it could incur a high runtime. Furthermore, an $O(n \log n)$-time 4-approximation algorithm was given in [3] and there also exists a 2.8334-approximation with runtime $O(n(\log n \log\log n)^2)$ in [10].

These algorithms are all for the case of no outliers. To the best of out knowledge, we only found three papers studying the outlier case of the covering problems as listed in Problem 1. In [11], Gandhi et al. gave a $(1+\epsilon)^2$-approximation algorithm for PDC which runs in runtime $O(\epsilon^{-1}n^{2\lceil\sqrt{2}/\epsilon\rceil^2+1})$. Later in [12], Ghasemalizadeh and Razzazi gave a $(1+\epsilon)$-approximation to PSC with outliers in runtime $O(\epsilon^{-1}n^{4/\epsilon+2})$ and a $(1+\epsilon)$-approximation to PDC with outliers in time[1] $O(\epsilon^{-1}n^{6\lceil\sqrt{2}/\epsilon\rceil+2})$. Note that their runtimes do not depend on t, the number of outliers, as their algorithms actually compute the solutions for all $t = 0, \ldots, n/2$. Additionally for PDC, Inamdar studies the local search methods in [19] and gives for $0 < \epsilon \leq 1/2$ an $(1+4\epsilon)$-approximation algorithm that runs in time at least $n^{O(1/\epsilon^2)}$ time. The best existing results for both the outlier and the non-outlier cases are summarized in Table 1, where $\ell = 1/\epsilon$ is the approximation parameter.

Although there are only limited works studying the partial geometric over, detecting outliers along with shape fitting tasks is of special significance in computational geometry and has thus become an enduringly popular research topic. Examples include k-means/medians/centers clustering with outliers [15], subsets of size $(n-t)$ with the minimum diameter [6], rectangles of the minimum area that covers at least $(n-t)$ points [24], convex hulls with outliers [2] and projective clustering with outliers [21].

A particularly important variant of the unit disk cover problem is when the given points lie within a vertical strip and we define the strip variants of Problem 1 as follows.

▶ **Problem 2** (Within-Strip Partial Geometric Covers). *Given a planar point set X contained in a vertical strip of width W and an integer $0 \leq t \leq n/2$ where $n = |X|$. We define three problems as follows.*

- *Within-strip Partial Unit Square Cover (StripPSC) find a minimum number of unit squares that cover at least $(n-t)$ points of X.*
- *Within-strip Partial Unit Disk Cover (StripPDC): find a minimum number of unit disks that cover at least $(n-t)$ points of X.*
- *Within-strip Restricted Partial Unit Disk Cover (StripRPDC): find a minimum subset of X such that the unit disks centered in the subset cover at least $(n-t)$ points of X.*

[1] We note an omission in the time complexity for PDC claimed in [12] and corrected it in our claim. In the last paragraph in Section 3 of [12, p553–554], which discusses the adaptation of the strip partial covering algorithm [12, p551] to disks. It divides the plane into strips of width 1 and group ℓ consecutive strips. However, when $\ell = 1$, the claim that "there can be no disk in the set OPT that covers points in two adjacent strips in more than one shift partition" at the bottom of [18, p132] would not hold and the approximation ratio of the original shifting strategy would not continue to hold. To apply the standard shifting strategy [18] when $\ell = 1$, the plane should be divided into strips of width 2 instead of 1. This leads to a runtime of $O(\epsilon^{-1}n^{6\lceil\sqrt{2}/\epsilon\rceil+2})$ instead of the claimed $O(\epsilon^{-1}n^{4\sqrt{2}/\epsilon+2})$.

Table 1 Summary of the best existing results (including the non-outlier case) and our main results. The runtime column suppresses the $O(\cdot)$ notation. The approximation parameter ℓ is always a positive integer. Some of our results are $(\alpha, 1+\delta)$-bicriteria approximations, i.e., they achieve an α-approximation by removing at most $(1+\delta)t$ outliers, where $\delta > 0$ is arbitrary.

	Paper	Problem	Approximation Ratio	Runtime
Unit Square Cover	[14]	PSC, $t = 0$	2	$n \log n$
		PSC, $t = 0$	$1 + \frac{1}{\ell}$	$\ell^2 n^{4\ell-1}$
	[12]	PSC	$1 + \frac{1}{\ell}$	$\ell n^{4\ell+2}$
	This work	PSC	$(2, 1+\delta)$	$\delta^{-1} nt \log n$
Unit Disk Cover	[14]	StripPDC, $t = 0$, $W = 2\ell$	1	$n^{4\lceil\sqrt{2\ell}\rceil+1}$
	[9]	StripPDC, $t = 0$, $W \le 4/5$	1	n^{13}
		StripRPDC, $t = 0$, $W \le 4/5$	1	n^7
	[10]	PDC, $t = 0$	2.8334	$n(\log n \log\log n)^2$
	[3]	PDC, $t = 0$	4	$n \log n$
	[5]	PDC, $t = 0$	7	n
	[11]	PDC	$(1 + \frac{1}{\ell})^2$	$\ell n^{2\lceil\sqrt{2\ell}\rceil^2+1}$
	[12]	PDC	$1 + \frac{1}{\ell}$	$\ell n^{6\lceil\sqrt{2\ell}\rceil+2}$
	This work	PDC	$(\frac{7}{2}, 1+\delta)$	$n^7 t + \delta^{-1} nt \log n$
		StripPDC, $W \le 4/5$	1	$n^7 t$
		RPDC	$(1 + \frac{6}{\sqrt{5}} + \frac{1}{\ell}, 1+\delta)$	$\ell n^4 t + \delta^{-1} \ell nt \log n$
		StripRPDC, $W \le \sqrt{5}/3$	1	$n^4 t$

The strip variants are motivated with the hope to obtain an algorithm of a better approximation ratio when some restriction is imposed on the input set, which is possible since the VC dimension might be smaller [16]. Once a local algorithm for the strip version is obtained, a natural idea for solving the full problem is to first partition the plane into strips and then merge the local within-strip solutions by the shifting strategy introduced in [18].

A challenge is that we need to determine the number of outliers on each strip in order to run the local algorithm. In Section 3, we introduce a new shifting strategy to overcome such challenge. We also prove in Theorem 1 that merging the local solutions will only incur a small multiplicative loss on the approximation ratio.

For notational convenience, we denote a solution to PSC, PDC, RPDC, StripPSC, StripPDC and StripRPDC by $sol_S(X,t)$, $sol_D(X,t)$, $sol_R(X,t)$, $sol_S(X,W,t)$, $sol_D(X,W,t)$ and $sol_R(X,W,t)$, respectively, and the optimal solution by $opt_S(X,t)$, $opt_D(X,t)$, $opt_R(X,t)$, $opt_S(X,W,t)$, $opt_D(X,W,t)$ and $opt_R(X,W,t)$, respectively.

Our Results. We summarize the existing results and our main results in Table 1.

We develop a shifting strategy that approximates the optimal number of outliers on each strip in Section 3. With the new shifting strategy, we give an $O(\delta^{-1} nt \log n)$-time bicriteria approximation algorithm to PSC, which outputs at most $2 \cdot opt(X,t)$ unit squares covering at least $n - (1+\delta)t$ of the given points. This can be viewed as an extension of the 2-approximation $O(n \log n)$-time result in [14] to the outlier case without compromising the time complexity for small t and constant δ.

For the strip variants of the disk cover, we give an $O(n^7 t)$-time exact algorithm for StripPDC when $W \le 4/5$, improving on the best known non-outlier result of $O(n^{13})$ time, and an $O(n^4 t)$-time exact algorithm for StripRPDC when $W \le \sqrt{5}/3$, also improving on the best known runtime of $O(n^7)$. These improvements are close to quadratic for small t.

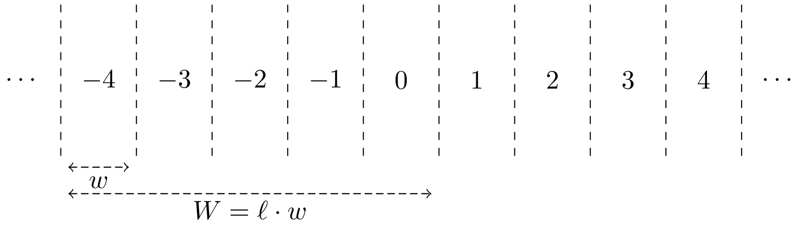


Figure 1 The plane is divided into vertical strips of width w, indexed by integers, and ℓ strips are grouped into a wider one of width W. The figure shows an example of $\ell = 5$.

For the original problem of PDC, based on our new results for the strip variant and the new shifting strategy, we show a 3.5-approximation algorithm with runtime $O(n^7 t)$. This is a new trade-off between the approximation ratio and the running time, and is so far the best running time for an approximation ratio less than 4 for PDC. In the same spirit, we show a $(1 + 6/\sqrt{5} + \epsilon)$-approximation algorithm for RPDC with a runtime of $O(n^4 t/\epsilon)$, where the polynomial dependence on n has a constant exponent, independent of ϵ.

We also extend the previous 4-approximation algorithm [3] for the unit disc cover problem to the outlier case with the same $O(n \log n)$ running time. See Appendix C.

2 Organization of the Paper

The high-level approach to solve PSC, PDC and RPDC follows a divide-and-conquer paradigm known as the shifting strategy [18]. We divide the plane into strips of equal width, run a local algorithm to solve the subproblem on every strip and then merge the local solutions. The main challenge is to determine the number of outliers within each strip. Inspired by [15], we develop a new shifting strategy in the presence of outliers that can in $O(tn \log n)$ time approximate the number of outliers on each strip. We present this new shifting strategy in Section 3 and then derive a 2-approximation to PSC in Section 4. The disk cover problems are discussed in Section 5. We state several new geometric observations in Section 5.1, upon which we design polynomial-time local algorithms that output exact solutions to StripPDC and StripRPDC in Section 5.2. Finally in Section 5.3, with the local algorithms and the new shifting strategy, we obtain one bicriteria algorithm for PDC and one for RPDC.

3 A Shifting Strategy Compatible with Outliers

The shifting strategy introduced by Hochbaum and Maass in [18] has been widely employed in the problem of geometric cover [14, 4, 8, 22, 1, 10, 20, 13]. The strategy requires a partition of the plane and a local algorithm for each single part of the partition. It runs the local algorithm for each part and merges the local solutions with only a small loss on the final approximation ratio. However, in the presence of outliers, we have to determine the number of outliers distributed to each part. Therefore, in this section, we develop a new shifting strategy that can approximate the number of outliers on each strip with provable guarantees.

We now illustrate the shifting strategy for PSC, PDC and RPDC with t outliers. Suppose that the plane is divided into (infinitely many) vertical strips of width w ($w \leq 1$), indexed by integers, say, $\ldots, -2, -1, 0, 1, 2, \ldots$ from left to right. There are in total ℓ ways $G_1, G_2, \ldots, G_\ell$ to group ℓ consecutive strips, where $G_i = \{[k \cdot \ell + i,\ k \cdot \ell + \ell + i - 1] \mid k \in \mathbb{Z}\}$, $i \in \{0, 1, 2, \ldots, \ell - 1\}$, resulting in the plane's being divided into wider strips of width $\ell \cdot w$. See Figure 1 for an illustration. To determine the number of outliers in each wider strip, we combine the shifting strategy [18] and the idea from [15]. We use i for the grouping index and j for the index for non-empty strip from left to right in a grouping G_i.

▶ **Theorem 1.** *With a local algorithm $\mathcal{A}$ to a strip of width $\ell \cdot w \le 1$, we can find a solution*

1. $sol_S(X, \lfloor(1+\delta)t\rfloor) \le \tau \cdot \left(1 + \frac{\lceil 1/w \rceil}{\ell}\right) \cdot opt_S(X,t)$ *for PSC;*
2. $sol_D(X, \lfloor(1+\delta)t\rfloor) \le \tau \cdot \left(1 + \frac{\lceil 2/w \rceil}{\ell}\right) \cdot opt_D(X,t)$ *for PDC;*
3. $sol_R(X, \lfloor(1+\delta)t\rfloor) \le \tau \cdot \left(1 + \frac{\lceil 2/w \rceil}{\ell}\right) \cdot opt_R(X,t)$ *for RPDC;*

where τ denotes the approximation ratio of $\mathcal{A}$ and $\delta > 0$ is arbitrary.

Proof. Let s_i ($s_i \le n$, $i = 1, 2, \dots, \ell$) denote the number of non-empty strips of width $\ell \cdot w$ in G_i. These strips in G_i are denoted as $S_{i,j}$, $j = 1, 2, \dots, s_i$ from left to right. Define $X_{i,j} = X \cap S_{i,j}$. We apply the local algorithm $\mathcal{A}$ to obtain a solution $sol(X_{i,j}, q)$ for each $q \in I$ where $I = \{\lfloor(1+\delta)^r\rfloor | r = 0, 1, 2, \dots, \lfloor\log_{1+\delta} t\rfloor\} \cup \{0, \lfloor(1+\delta)t\rfloor\}$. We also define a function $f_{i,j}$ on $\{0, 1, 2, \dots, \lfloor(1+\delta)t\rfloor\}$, where $f_{i,j}(q)$ ($q \in \{0, 1, 2, \dots, \lfloor(1+\delta)t\rfloor\}$) is defined to be the value of the lower convex hull of $\{(q, sol(X_{i,j}, q)) \,|\, q \in I\}$. The summation $F_i(q_{i,1}, q_{i,2}, \dots, q_{i,s_i}) = \sum_{j=1}^{s_i} f_{i,j}(q_{i,j})$, $\sum_{j=1}^{s_i} q_{i,j} \le \lfloor(1+\delta)t\rfloor$ is a convex function.

The minimum point of F_i can be found by going down along the edges of the convex polygonal surface whose vertices are $(q_{i,1}, q_{i,2}, \dots, q_{i,s_i}, F_i(q_{i,1}, q_{i,2}, \dots, q_{i,s_i}))$, $\sum_{j=1}^{s_i} q_{i,j} \le \lfloor(1+\delta)t\rfloor$. The details are presented in Algorithm 1. We denote the minimum point by $(t_{i,1}, t_{i,2}, \dots, t_{i,s_i})$. We also let $t^*_{i,j}$ denote the number of outliers in $X_{i,j}$ for the optimal solution $opt(X,t)$, and $t'_{i,j}$ be the power of $1+\delta$ between $t^*_{i,j}$ and $\lfloor(1+\delta)t^*_{i,j}\rfloor$. The solutions on each strip are put together to get

$$sol_i(X, \lfloor(1+\delta)t\rfloor) := \sum_{j=1}^{s_i} sol(X_{i,j}, t_{i,j}).$$

As $\sum_{j=1}^{s_i} t'_{i,j} \le \sum_{j=1}^{s_i} \lfloor(1+\delta)t^*_{i,j}\rfloor \le \lfloor(1+\delta)t\rfloor$, we then have

$$\sum_{j=1}^{s_i} sol(X_{i,j}, t_{i,j}) = F_i(t_{i,1}, t_{i,2}, \dots, t_{i,s}) \le F_i(t'_{i,1}, t'_{i,2}, \dots, t'_{i,s_i}) \le \sum_{j=1}^{s_i} sol(X_{i,j}, t'_{i,j}).$$

Moreover $opt(X_{i,j}, t'_{i,j}) \le opt(X_{i,j}, t^*_{i,j})$ as $opt(X_{i,j}, q)$ is a decreasing function on q. We thus have

$$sol_i(X, \lfloor(1+\delta)t\rfloor) \le \sum_{j=1}^{s_i} sol(X_{i,j}, t'_{i,j}) \le \tau \cdot \sum_{j=1}^{s_i} opt(X_{i,j}, t'_{i,j}) \le \tau \cdot \sum_{j=1}^{s_i} opt(X_{i,j}, t^*_{i,j}).$$

We claim that a unit square can cross at most $\lceil 1/w \rceil + \ell$ strips of width $\ell \cdot w$ in $G_1 \cup \dots \cup G_\ell$. The proof is deferred to Lemma 5. This indicates that for unit square

$$\sum_{i=1}^{\ell} sol_i(X, \lfloor(1+\delta)t\rfloor) \le \tau \cdot \sum_{i=1}^{\ell} \sum_{j=1}^{s_i} opt(X_{i,j}, t^*_{i,j}) \le \tau \cdot (\lceil 1/w \rceil + \ell) \cdot opt(X,t).$$

We finally get a solution

$$sol(X, \lfloor(1+\delta)t\rfloor) = \min_{i=1,\dots,\ell} sol_i(X, (1+\delta)t) \le \tau \cdot \left(1 + \frac{\lceil 1/w \rceil}{\ell}\right) \cdot opt(X,t).$$

For unit disks, we shall prove in Lemma 5 that a unit disk can cross at most $\lceil 2/w \rceil + \ell$ strips in $G_1 \cup \dots \cup G_\ell$ and a similar argument as above yields a solution with an approximation factor of $\tau \cdot \left(1 + \frac{\lceil 2/w \rceil}{\ell}\right)$. ◀

Algorithm 1 Local algorithm for every non-empty strip in a grouping.

```
procedure SHIFTING($X$, $t$, $\mathcal{A}$, $G_i$)        ▷ $X$ a set of $n$ points, $0 \le t < n$, $G_i$ a grouping
    $s \leftarrow$ number of nonempty strips in $G_i$                                        ▷ $s \le n$
    $I \leftarrow \{\lfloor (1+\delta)^r \rfloor \mid r = 0, 1, 2, \ldots, \lfloor \log_{1+\delta} t \rfloor\} \cup \{0, \lfloor (1+\delta)t \rfloor\}$
    for $j = 1, 2, \ldots, s$ do
        $X_j \leftarrow$ points of $X$ that are on the $j$-th nonempty strip
        Compute the lower convex hull of $\{(q, \mathcal{A}(X_j, q)) | q \in I\}$ ▷ $\mathcal{A}$ is the local algorithm
        for $t_j = 0, 1, \ldots, \lfloor (1+\delta)t \rfloor$ do
            $f_j(t_j) \leftarrow$ the value of the lower convex hull at $t_j$, as defined in text
    $q_1 \leftarrow 0$, $q_2 \leftarrow 0$, $\ldots$, $q_s \leftarrow 0$
    $T \leftarrow [(f_j(t_j) - f_j(t_j - 1), j, t_j) \,| 1 \le j \le s, 1 \le t_j \le \lfloor (1+\delta)t \rfloor]$
    Sort $T$ according to the partial ordering $\preccurlyeq$
    for $k = 1, 2, \ldots, \lfloor (1+\delta)t \rfloor$ do
        $j \leftarrow$ the index such that $T[k] = (f_j(t_j) - f_j(t_j - 1), j, t_j)$
        $q_j \leftarrow q_j + 1$
    return $(q_1, q_2, \ldots, q_s)$
```

An algorithm to find the minimum point of $F_i(q_{i,1}, q_{i,2}, \ldots, q_{i,s_i})$ subject to $\sum_{j=1}^{s_i} q_{i,j} \le \lfloor (1+\delta)t \rfloor$ is given in [15], which we reproduce in Algorithm 1. In the remaining of this subsection, we focus on explaining the algorithm that outputs the minimum of F_i in one grouping and thus omit the grouping index i in all the subscripts. For example, F_i becomes F and $f_{i,j}$ becomes f_j. Also for convenience, we have the following definition.

▶ **Definition 2.** *Suppose $g_j(q) := f_j(q) - f_j(q-1)$, then we define a partial ordering such that $g_{j_1}(q_1) \preccurlyeq g_{j_2}(q_2)$ if one of the following conditions is satisfied*

1. $f_{j_1}(q_1) - f_{j_1}(q_1 - 1) < f_{j_2}(q_2) - f_{j_2}(q_2 - 1)$
2. $j_1 < j_2$ *and* $f_{j_1}(q_1) - f_{j_1}(q_1 - 1) = f_{j_2}(q_2) - f_{j_2}(q_2 - 1)$
3. $j_1 = j_2$ *and* $q_1 < q_2$ *and* $f_{j_1}(q_1) - f_{j_1}(q_1 - 1) = f_{j_2}(q_2) - f_{j_2}(q_2 - 1)$

With the partial ordering on $\{g_j(q) | 1 \le j \le s, 1 \le q \le (1+\delta)t\}$, we restate the algorithm in [15] below. The algorithm can be regarded as a discrete version of gradient descent, at each step of which we go down along the steepest direction in which the function value decreases the most. The correctness of Algorithm 1 is ensured by the following two lemmata.

▶ **Lemma 3.** *At the beginning of the k-th iteration in Algorithm 1, we have $q_j = t_j - 1$ where $(f_j(t_j) - f_j(t_j - 1), j, t_j) = T[k]$.*

Proof. For any $t'_j < t_j$, we have $f_j(t'_j) - f_j(t'_j - 1) < f_j(t_j) - f_j(t_j - 1)$ by the convexity of f_j. Therefore $f_j(t'_j) - f_j(t'_j - 1)$ must be ahead of $f_j(t_j) - f_j(t_j - 1)$ under the partial ordering $\preccurlyeq$. This indicates that the update $q_j \leftarrow q_j + 1$ has been executed $(t_j - 1)$ times before. As the initial value of q_j is 0, then at the k-th iteration $(f_j(t_j) - f_j(t_j - 1), j, t_j) = T[k]$, it must be true that $q_j = t_j - 1$. ◀

We defer the proof of the following lemma to Appendix A for completeness.

▶ **Lemma 4** (Lemma 3.3 [15])**.** *Algorithm 1 outputs* $\min f(q_1, \ldots, q_s)$ *subject to* $\sum_{j=1}^{s} q_j \le \lfloor (1+\delta)t \rfloor$.

We simply select the minimum among all $sol_i(X, \lfloor (1+\delta)t \rfloor)$, $1 \le i \le \ell$ as our final solution. To prove its approximation ratio, we need the following lemma.

▶ **Lemma 5.** *Suppose that $\ell \cdot w \leq 1$. If the strip boundary lines in the plane partition do not cross any point of X, then a unit square can cover points of X distributed in at most $\lceil 1/w \rceil + \ell$ different strips of width $\ell \cdot w$ in $G_1 \cup \cdots \cup G_\ell$, and a unit disk at most $\lceil 2/w \rceil + \ell$.*

Proof. Let S denote the leftmost and S' the rightmost strip of width $\ell \cdot w$ that intersects a unit square. Also let i_1 and i_2 denote the indices of the left boundary line of S and S' respectively. Then the index of the right boundary line of S is $i_1 + \ell$. Note that the distance between the right boundary line of S and the left boundary line of S' is $(i_2 - i_1 - \ell) \cdot w$ and must be smaller than 1. Therefore, we have $i_2 - i_1 < \ell + 1/w$. Since i_1, i_2 are integers, this implies that $i_2 - i_1 \leq \ell + \lceil 1/w - 1 \rceil$. We then conclude a unit square can intersect at most $\ell + \lceil 1/w - 1 \rceil + 1 = \ell + \lceil 1/w \rceil$ strips of width $\ell \cdot w$ in $G_1 \cup \cdots \cup G_\ell$. A similar argument works for unit disks. ◀

The algorithm to partition the plane into vertical strips of width w is presented in Appendix D. It guarantees that no boundary line crosses a point in X. It remains to develop local algorithms for the StripPSC, StripPDC and StripRPDC on strip of width $W = \ell \cdot w$.

4 Square Cover

In this section, we illustrate the application of Theorem 1 to the partial square cover problem. We first consider the local problem StripPSC with $W = 1$. For convenience, we define the notion of anchored squares as follows. Note that our definition is different from that in [18].

▶ **Definition 6** (Anchored Square). *For a strip of width $W = 1$ and a point set X within the strip, a square is anchored if its left and right sides coincide with the left and right boundary lines of the strip, respectively, and its upper side crosses a point of X.*

As a unit square L can be translated to an anchored one L' so that $L \cap X \subseteq L' \cap X$, we therefore only consider the anchored squares in StripPSC when $W = 1$. There are n of them. Without loss of generality, we can assume that no two points in X have the same y-coordinates, otherwise we simply rotate the plane. We sort X in the increasing order of the y-coordinates, say, $X_1, \ldots, X_n$. Let L_i denote the anchored square with X_i on its upper side and B_{ji} $(j < i)$ denote the number of points above the upper side of L_j and below the lower side of L_i. Let $N[i][k]$ denote the minimum size of a set of squares that covers X_i $(1 \leq i \leq n)$ and at least $(i - k)$ $(0 \leq k \leq t)$ points from X_1 to X_i, then we have the following recursive formula for $N[i][k]$.

$$N[i][k] = \min_{j<i} N[j][k - B_{ji}] + 1. \tag{1}$$

Note that we require X_i to be covered in an optimal cover of $N[i][k]$. Therefore it is only necessary to consider in (1) those j's such that X_j is not covered by L_i and $B_{ji} \leq k$. Let h_i denote the index of the highest point below the lower side of L_i, then only those $j \in [h_i - k, h_i]$ would be considered. As $k \leq t$, there are at most t such candidate j's. Computing all h_i takes $O(n)$ time and we can store these values. We also note that $B_{ji} = h_i - j$ for those $j \in [h_i - k, h_i]$. Therefore the recursive formula (1) can be rewritten as

$$N[i][k] = \min_{h_i - k \leq j \leq h_i} N[j][k - h_i + j] + 1.$$

This is the base of our local algorithm that outputs an exact solution to StripPSC when $W = 1$, which we present in Algorithm 2.

Algorithm 2 The algorithm that outputs an optimal solution to StripPSC.

```
procedure SquareLocal($X$, $t$)                                  ▷ $0 \le t \le n/2$
    $X \leftarrow$ set of planar points
    $n \leftarrow |X|$
    sort $X$ so that the $y$-coordinates of the points are increasing
    $L_i \leftarrow$ the anchored square with $X_i$ on its upper side
    $h_i \leftarrow$ the index of the highest point of $X$ below $L_i$
    Initialize array $N[j][k] \leftarrow \infty$ for $1 \le j \le n$ and $0 \le k \le t$
    for $i = 1, 2, \ldots, n$ do
        for $k = 0, 1, \ldots, t$ do
            $N[i][k] \leftarrow \min_{j \in [h_i - k, h_i]} N[j][k - h_i + j] + 1$
    return $\min_{k \in [0,t]} N[n-k][t-k]$
```

▶ **Lemma 7.** *There exists an exact algorithm to StripPSC in time $O(nt \log n)$ when $W = 1$.*

Proof. It takes $O(n)$ time to compute all h_i and $O(nt)$ time to compute all B_{ij} for $j \in [h_i - t, h_i]$. For each $k = 0, \ldots, t$, we maintain a data structure of the dynamic range minimum query (RMQ) for the one-dimensional array $N[1][k], \ldots, N[n][k]$. The dynamic RMQ structure in [17] supports both update and query in $O(\log n)$ time. For each $i = 1, \ldots, n$ and $k = 0, \ldots, t$, we update the data structure once and query the data structure once. Therefore, filling the $N[i][k]$ array takes time $O(nt \log n)$ in total and the overall runtime is thus $O(nt \log n)$. ◀

▶ **Theorem 8.** *For PSC, there exists an $O(\delta^{-1} nt \log n)$-time algorithm which outputs $2 \cdot opt(X, t)$ unit squares that cover at least $n - (1+\delta)t$ of the given points.*

Proof. Note that in Algorithm 2, we have computed all $N[i][k]$ for $1 \le i \le n$ and $1 \le k \le t$, that is, we know the optimal solution to StripPSC that cover $n - k$ points of X for all $k = 0, \ldots, t$ when $W = 1$. With Algorithm 2 as the local algorithm $\mathcal{A}$ and $\ell = 1$, we can run Algorithm 1 in $O(nt \log_{1+\delta} n) = O(\delta^{-1} nt \log n)$ time and obtain a solution of size $\tau \cdot \left(1 + \frac{\lceil 1/w \rceil}{\ell}\right) \cdot opt(X, t) = 2 \cdot opt(X, t)$ that covers at least $n - (1+\delta)t$ points of X. In total, the time complexity is $O(\delta^{-1} nt \log n)$. ◀

5 Disk Cover

In this section, we study the disk cover problems PDC and RPDC. We first introduce the notions and definitions in Section 5.1. We also prove a few lemmata which are critical for developing the local algorithms to StripPDC and StripRPDC. In Section 5.2 we describe the local algorithm in details. And finally in Section 5.3, we solve PDC and RPDC by the local algorithm in Section 5.2 and the shifting strategy we develop in Section 3 together.

5.1 Geometric Observations

When the disk centers are unrestricted, it is equivalent to considering only the unit disks whose boundary cross two points of X. Such unit disks are defined as the anchored disks in [11, 12]. In this work, we extend this notion to include the unit disks with a point in X as its highest or lowest point and give the formal definition below. We shall only consider the anchored disks for PDC and StripPDC in the rest of the paper.

▶ **Definition 9** (Anchored Unit Disks). *Given a point set X in the plane, a unit disk is anchored if either there are two points of X on its boundary, or the highest, or the lowest point of the unit disk is in X.*

▶ **Lemma 10.** *It is sufficient to consider only the anchored unit disks in* PDC *and* StripPDC*. There are at most $n^2 + n$ anchored disks.*

Proof. For any unit disk D, let D' be the lowest disk such that $D' \cap X \supseteq D \cap X$. We can prove by contradiction that there are two points of X on the boundary of D' or the highest point of D' is in X. If there is no point of X on the boundary of D', then we can translate D' downwards by a small value such that the translated disk still covers $D' \cap X \supseteq D \cap X$. If there is only one point of X on the boundary of D' and the point is not at the highest position of D', we can rotate D' around this point by a small angle so that the y-coordinate of the disk center decreases and the rotated disk still covers $D' \cap X \supseteq D \cap X$. Either of two cases would result in a contradiction. There are at most $\frac{(n-1)n}{2} \cdot 2 + n = n^2$ of such disks. We also include the disks whose lowest point is in X and it total there are $n^2 + n$ of them. ◀

Before presenting the observations related to strip geometric cover, we introduce some basic notions and lemmata which are helpful for the readers to understand the results. Definition 11 is inspired by the mutually spanning set in [9], in which one unit disk is supposed to cover a nonempty subset of X both above and below any other unit disk. However, the definition of mutually spanning set is too strong and unnecessary. We therefore revise it into the top spanning set.

▶ **Definition 11** (Top Spanning Set). *Suppose the points in X lie in a vertical strip of width $w < 1$. A set of unit disks is top spanning if the set is either a singleton, or each unit disk other than the highest one covers a nonempty subset of X above the highest disk.*

▶ **Definition 12** (Inscribed Rectangles [9]). *A unit disk centered in a strip of width $W < 1$ covers a strip segment of height at least $2\sqrt{1 - W^2}$. The strip segment is referred to as the inscribed rectangle of the unit disk.*

An illustration of the inscribed rectangle is shown in Figure 2.

▶ **Definition 13** (Vertical Span). *Given a strip and a set of unit disks $\{D_1, \ldots, D_m\}$, the vertical span is defined to be the height difference between the highest and lowest points of the strip covered by $\bigcup_{i=1}^{m} D_i$.*

The following lemma was proved for the mutually spanning set in [9] and it is still true for the top spanning set. We reproduce a proof in Appendix B.

▶ **Lemma 14** ([9]). *Consider unit disks D_1, D_2 whose centers o_1, o_2 are in a vertical strip of width $W < 1$. If $y(o_1) \geq y(o_2)$ and D_2 covers some point above D_1, then we conclude that $y(o_1) - y(o_2) \leq 1 - \sqrt{1 - W^2}$. Further, the span of a top spanning set is at most $3 - \sqrt{1 - W^2}$.*

We are now ready for proving a few geometric observations. In [9], the authors studied the within strip unit disk cover where the points are within a given strip, and the unit disks can only be selected from a finite set $\mathcal{D}$ where the disk centers are also witihn the strip. Let X' consist of points in X that are covered by the inscribed rectangles of the disks in $\mathcal{D}$. In [9, Lemma 5], it is proved that among all $C \subseteq \mathcal{D}$ that covers X', there exists one covering of the minimum size and the covering does not contain any mutually spanning set of more than 3 disks. In a similar approach, we can prove the following two lemmata.

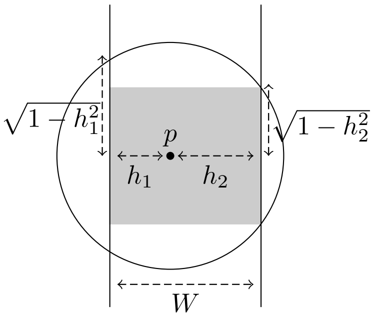


Figure 2 The figure illustrates the inscribed rectangle. The two vertical lines are the left and right boundary lines of a strip of width W. The center point p of the unit disk is inside the strip. The distance from p to the left boundary line is denoted by h_1, and the distance to the right boundary line is denoted by h_2. The shadowed area is the inscribed rectangle whose height is $2\min\{\sqrt{1-h_1^2}, \sqrt{1-h_2^2}\} \geq 2\sqrt{1-W^2}$.

▶ **Lemma 15.** *When $W \leq \frac{4}{5}$, there exists an optimal solution on* StripPDC *that contains no top spanning set of more than* 2 *disks.*

Proof. Assume that *opt* is an optimal solution and contains a top spanning set of 3 disks, say, D_1, D_2 and D_3, from bottom to top. Let p_1 be the lowest point of X covered by $D_1 \cup D_2 \cup D_3$ and p_2 the highest. By Lemma 14, we know the vertical span of $D_1 \cup D_2 \cup D_3$ is at most $3 - \sqrt{1-W^2}$. Let D_1' be the unit disk with p_1 as its lowest point and D_2' be the unit disk with p_2 as its highest point. Then D_1' covers a segment of length at least $2\sqrt{1-W^2}$ above p_1, and D_2' covers a segment of length at least $2\sqrt{1-W^2}$ below p_2. Since $2 \times 2\sqrt{1-W^2} \geq 3 - \sqrt{1-W^2}$ when $W \leq \frac{4}{5}$, we can replace $\{D_1, D_2, D_3\}$ with $\{D_1', D_2'\}$ and obtain a smaller solution, contradicting the optimality of *opt*. Therefore *opt* does not contain any top spanning set of 3 unit disks. ◀

▶ **Lemma 16.** *When $W \leq \frac{\sqrt{5}}{3}$, there is an optimal solution on* StripRPDC *that contains no top spanning set of more than* 2 *disks.*

Proof. Assume *opt* is an optimal cover and contains a top spanning set of 3 unit disks. Let D_1, D_2, D_3, p_1 and p_2 be as in the proof of Lemma 15. Besides, by o_1, o_2 and o_3 we denote the centers of D_1, D_2 and D_3, respectively. From Lemma 14 we know $y(o_3) - y(p_1) \leq y(o_3) - y(o_1) + 1 \leq 2 - \sqrt{1-W^2}$. Let p denote the highest point not covered by D_3 and we have $y(o_3) - y(p) \geq \sqrt{1-W^2}$. The unit disk D centered at p covers a strip segment of length $\sqrt{1-W^2}$ below p, and hence $D \cup D_3$ covers a strip segment of length $2\sqrt{1-W^2}$ below o_3. When $W \leq \frac{\sqrt{5}}{3}$, we have $2\sqrt{1-W^2} \geq 2 - \sqrt{1-W^2}$ and $(D_1 \cup D_2 \cup D_3) \cap X \subseteq (D \cup D_3) \cap X$. Replacing $D_1 \cup D_2 \cup D_3$ with $D \cup D_3$ would give us a smaller cover, which contradicts the optimality of *opt*. ◀

5.2 Exact Algorithms to StripPDC and StripRPDC

In this subsection, we develop an exact algorithm for StripPDC from Lemma 15. The detailed description is in Algorithm 3. In the same way, we can develop an exact algorithm for StripRPDC from Lemma 16. Let $\mathcal{D}$ denote the set of unit disks which are candidates in an optimal covering. For StripPDC they are the anchored disks as defined in Definition 9 and for StripRPDC they are the unit disks centered in the point set X. We first state our main theorem below.

Algorithm 3 The algorithm that outputs an optimal solution to StripPDC.

```
procedure DiskLocal($X$, $t$)
    $U[i][k] \leftarrow \emptyset$ for $i = 0, 1, \ldots, n$ and $k = 0, 1, 2, \ldots, n - t$
    $G[i][k] \leftarrow \emptyset$ for $i = 1, 2, \ldots, n$ and $k = 0, 1, 2, \ldots, n - t$
    for $i = 1, 2, \ldots, n$ do
        for $k = \max(i - t, 0), \max(i - t, 0) + 1, \ldots, \min(i - 1, n - t)$ do
            for $T \in U[i - 1][k]$ do
                if the disks in $T$ cover $X_i$ then
                    $G[i][k + 1] \leftarrow G[i][k + 1] \cup \{T\}$
                else
                    $G[i][k] \leftarrow G[i][k] \cup \{T\}$
                    for $D \in \mathcal{D}$ do
                        if $D$ covers $X_i$ then
                            $G[i][k + 1] \leftarrow G[i][k + 1] \cup \{T \cup \{D\}\}$
        for $k = \max(i - t, 0), \max(i - t, 0) + 1, \ldots, \min(i, n - t)$ do
            for $T \in G[i][k]$ do
                if $|T| \leq 2$ then
                    $U[i][k] \leftarrow U[i][k] \cup \{T\}$
            for $T_1 \in U[i][k]$ do
                for $T_2 \in U[i][k]$ do
                    if $sig(T_1) = sig(T_2)$ and $|T_1| \leq |T_2|$ then
                        $U[i][k] \leftarrow U[i][k] - \{T_2\}$
    return any $T \in U[n][n - t]$
```

▶ **Theorem 17.** *Algorithm 3 computes an exact solution to StripPDC in $O(n^7 t)$ time when $W \leq 4/5$. If the set of anchored disks is replaced by the set of unit disks centered in X, Algorithm 3 would output an exact solution to StripRPDC in $O(n^4 t)$ time.*

We follow the dynamic programming introduced in [12, Section 3] to develop a local algorithm. We assume no two points of X have the same y-coordinates, otherwise we can rotate X. We also sort X in an increasing order of their y-coordinates. Let X_1, X_2, …, X_n denote the sorted points from bottom to top. A subcover is denoted by (k, i, T) where T is the set of the disks that cover at least k points between X_1 and X_i. The signature of (k, i, T), denoted by $sig(k, i, T)$, consists the highest disk D of T and those in T which cover at least one point of X above D.

Now we state the idea of the dynamic programming. Let $U[i][k]$ store all the subcovers that cover at least k points at the i-th step in the algorithm. We iterate over all $T \in U[i][k]$. If T already covers X_{i+1}, we simply add T into $U[i + 1][k + 1]$. Otherwise, we add T to $U[i + 1][k]$ and all possible $T \cup \{D\}$ to $U[i + 1][k + 1]$, where $D \in \mathcal{D}$ is a unit disk that covers X_{i+1}. At the end of the i-th iteration, for two subcovers T_1 and T_2 in $U[i + 1][k]$, if $sig(T_1) = sig(T_2)$ and $|T_1| \leq |T_2|$, we remove T_2 from $U[i + 1][k]$. By Lemma 15, there is an optimal solution to StripPDC that contains no top spanning set of more than 2 disks. It is obvious that the signature is a top spanning set by Definition 11, we therefore remove T if $|sig(T)| > 2$.

Proof of Theorem 17. The correctness of Algorithm 3 is guaranteed by Lemmata 1 and 2 in [12]. For any $T \in U[i][k]$, we have $|sig(T)| \leq 2$ and therefore there are $O(n^4)$ different signatures. There are no two subcovers with the same signature in $U[i][k]$, so $|U[i][k]| = O(n^4)$.

Observe that $U[i][k]$ is nonempty for at most t values of k, it holds that $\sum_k |U[i][k]| = O(n^4 t)$. Furthermore, since the number of disks that cover X_i is $O(n^2)$, we have $|G[i][k]| = O(n^6)$ and further $\sum_k |G[i][k]| = O(n^6 t)$. After we construct $G[i][k]$, we only select some of them into $U[i][k]$. The would cost $O(n \cdot n^6 t)$ time. Besides, we also remove the larger covering of the sane signature in $U[i][k]$. The process can be done in linear time with respect to $|U[i][k]|$, as shown in [12] with the techniques from [14]. The overall time complexity is then $O(n \cdot t \cdot n^4 \cdot n^2 + n \cdot t \cdot n^6) = O(n^7 t)$.

The same algorithm can be applied to StripRPDC and the only difference is that we use the unit disks centered in X instead as the candidates in a covering. There are n such disks and by Lemma 16 there are $O(n^2)$ different signatures of size at most 2. Also we have $U[i][k] = O(n^2)$ and $G[i][k] = O(n^3)$ for StripRPDC. The overall time complexity would be $O(n \cdot t \cdot n^2 \cdot n + n \cdot t \cdot n^3) = O(n^4 t)$. ◀

5.3 Approximation Algorithms to PDC and RPDC

We apply Algorithm 3 as the local algorithm $\mathcal{A}$ for PDC and RPDC. Combining with Theorem 1, we obtain a global algorithm to PDC with approximation ratio $1 \cdot \left(1 + \frac{\lceil 2/w \rceil}{\ell}\right) = 3.5$ for $w = 0.4$ and $\ell = 2$, and a global algorithm to RPDC whose approximation factor is $1 \cdot \left(1 + \frac{\lceil 2/w \rceil}{\ell}\right) \leq 1 + \frac{2}{\ell \cdot w} + \frac{1}{\ell} = 1 + \frac{6}{\sqrt{5}} + \frac{1}{\ell} \approx 3.68 + \frac{1}{\ell}$.

▶ **Theorem 18.** *There exist a $(3.5, 1+\delta)$-bicriteria algorithm for PDC which runs in time $O(n^7 t + \delta^{-1} nt \log n)$, and a $(1 + \frac{6}{\sqrt{5}} + \frac{1}{\ell}, 1 + \delta)$-bicriteria algorithm for RPDC which runs in time $O(\ell n^4 t + \delta^{-1} \ell nt \log n)$.*

Proof. Let $n_j = |X_j|$, the number of points in the j-th nonempty strip of the grouping G_i.

For PDC, we apply Algorithm 3 as the local algorithm $\mathcal{A}$ in Algorithm 1. Note that in Algorithm 3, all $U[n][n-k]$ $(0 \leq k \leq t)$ are computed. Therefore it takes $O(n_j^7 t)$ time to output $sol(X_j, t_j)$ for all $t_j \in \{\lfloor (1+\delta)^r \rfloor \mid r = 0, 1, 2, \ldots, \lfloor \log_{1+\delta} t \rfloor\} \cup \{0, \lfloor (1+\delta)t \rfloor\}$. On all the s_i strips, this would cost $O\left(n_1^7 t + \cdots + n_{s_i}^7 t\right) = O(n^7 t)$ time. Sorting all the values $f_i(t_i) - f_i(t_i - 1)$ in Algorithm 1 takes $O\left(nt \log_{1+\delta} nt\right) = O\left(\delta^{-1} nt \log n\right)$ time. As there are ℓ groupings, the total complexity is $O(\ell n^7 t + \delta^{-1} \ell nt \log n)$. Letting $\ell = 2$ and $w = 2/5$ yields a 3.5-approximation with time complexity $O(n^7 t + \delta^{-1} nt \log n)$.

For RPDC, we prove in the same way that the overall time complexity is $O(\ell n^4 t + \delta^{-1} \ell nt \log n)$. ◀

References

1 Christoph Ambühl, Thomas Erlebach, Matúš Mihalák, and Marc Nunkesser. Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 3–14. Springer, 2006.

2 Rossen Atanassov, Prosenjit Bose, Mathieu Couture, Anil Maheshwari, Pat Morin, Michel Paquette, Michiel Smid, and Stefanie Wuhrer. Algorithms for optimal outlier removal. *Journal of discrete algorithms*, 7(2):239–248, 2009.

3 Ahmad Biniaz, Paul Liu, Anil Maheshwari, and Michiel Smid. Approximation algorithms for the unit disk cover problem in 2D and 3D. *Computational Geometry*, 60:8–18, 2017.

4 Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. *SIAM Journal on Computing*, 33(6):1417–1440, 2004.

5 Adrian Dumitrescu, Anirban Ghosh, and Csaba D Tóth. Online unit covering in euclidean space. *Theoretical Computer Science*, 809:218–230, 2020.

6 David Eppstein and Jeff Erickson. Iterated nearest neighbors and finding minimal polytopes. *Discrete & Computational Geometry*, 11(3):321–350, 1994.

7 Robert J Fowler, Michael S Paterson, and Steven L Tanimoto. Optimal packing and covering in the plane are NP-complete. *Information processing letters*, 12(3):133–137, 1981.

8 Massimo Franceschetti, Matthew Cook, and Jehoshua Bruck. A geometric theorem for approximate disk covering algorithms. Technical Report ETR035, California Institute of Technology, 2001.

9 Robert Fraser and Alejandro López-Ortiz. The within-strip discrete unit disk cover problem. *Theoretical Computer Science*, 674:99–115, 2017.

10 Bin Fu, Zhixiang Chen, and Mahdi Abdelguerfi. An almost linear time 2.8334-approximation algorithm for the disc covering problem. In *International Conference on Algorithmic Applications in Management*, pages 317–326. Springer, 2007.

11 Rajiv Gandhi, Samir Khuller, and Aravind Srinivasan. Approximation algorithms for partial covering problems. *Journal of Algorithms*, 53(1):55–84, 2004.

12 Hossein Ghasemalizadeh and Mohammadreza Razzazi. An improved approximation algorithm for the most points covering problem. *Theory of Computing Systems*, 50(3):545–558, 2012.

13 Anirban Ghosh, Brian Hicks, and Ronald Shevchenko. Unit disk cover for massive point sets. In *International Symposium on Experimental Algorithms*, pages 142–157. Springer, 2019.

14 Teofilo F Gonzalez. Covering a set of points in multidimensional space. *Information processing letters*, 40(4):181–188, 1991.

15 Sudipto Guha, Yi Li, and Qin Zhang. Distributed partial clustering. *ACM Transactions on Parallel Computing (TOPC)*, 6(3):1–20, 2019.

16 Sariel Har-Peled. On complexity, sampling, and ε-nets and ε-samples. *Approximation Algorithm in Geometry*, 2010.

17 Alice Héliou, Martine Léonard, Laurent Mouchard, and Mikael Salson. Efficient dynamic range minimum query. *Theoretical Computer Science*, 656:108–117, 2016.

18 Dorit S Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM (JACM)*, 32(1):130–136, 1985.

19 Tanmay Inamdar. Local search for geometric partial covering problems. In *Proceedings of the 31st Canadian Conference on Computational Geometry*, pages 242–249, 2019.

20 Paul Liu and Daniel Lu. A fast 25/6-approximation for the minimum unit disk cover problem, 2014. arXiv:1406.3838.

21 Nina Mishra, Rajeev Motwani, and Sergei Vassilvitskii. Sublinear projective clustering with outliers. In *15th Annual Fall Workshop on Computational Geometry and Visualization*, page 45. Citeseer, 2005.

22 Sada Narayanappa and Petr Vojtechovskỳ. An improved approximation factor for the unit disk covering problem. In *Proceedings of the 18th Canadian Conference on Computational Geometry*, pages 15–18, 2006.

23 Kaveh Pahlavan and Allen H Levesque. Wireless data communications. *Proceedings of the IEEE*, 82(9):1398–1430, 1994.

24 Michael Segal and Klara Kedem. Enclosing k points in the smallest axis parallel rectangle. *Information Processing Letters*, 65(2):95–99, 1998.

25 Steven L Tanimoto and Robert J Fowler. Covering image subsets with patches. In *Proceedings of the fifty-first International Conference on Pattern Recognition*, pages 835–839, 1980.

A Proof of Lemma 4

Proof. Let $(t_1^*, \dots, t_s^*)$ denote a minimum point of f and it is obvious that $\sum_{j=1}^{s} t_j^* = \lfloor(1+\epsilon)t\rfloor$. We also let $(q_1, \dots, q_s)$ denote the output of Algorithm 1 and $\sum_{j=1}^{s} q_j = \lfloor(1+\epsilon)t\rfloor$. If $(q_1, \dots, q_s) \neq (t_1^*, \dots, t_s^*)$, then there must be some $q_{j_1} < t_{j_1}^*$ and $q_{j_2} > t_{j_2}^*$. As q_{j_1} is the final output value of Algorithm 1, $f_{j_1}(t_{j_1}^*) - f_{j_1}(t_{j_1}^* - 1)$ cannot be in the first $\lfloor(1+\epsilon)t\rfloor$ elements of S. Besides, we have $f_{j_2}(t_{j_2}^* + 1) - f_{j_2}(t_{j_2}^*) \le f_{j_2}(q_{j_2}) - f_{j_2}(q_{j_2} - 1)$ by the convexity of f_{j_2} and

$f_{j_2}(t^*_{j_2}+1)-f_{j_2}(t^*_{j_2})$ must in the first $\lfloor(1+\epsilon)t\rfloor$ elements of S. Therefore $f_{j_2}(t^*_{j_2}+1)-f_{j_2}(t^*_{j_2}) \leq f_{j_1}(t^*_{j_1}) - f_{j_1}(t^*_{j_1}-1)$. This indicates that $(\dots, t^*_{j_1}-1, \dots, t^*_{j_2}+1, \dots)$ is also a minimum point of f and its L_1 distance to $(q_1, \dots, q_s)$ is less than that of $(t^*_1, \dots, t^*_s)$. Repeat this process, we can finally prove that the output of Algorithm 1 is a global minimum. ◀

B Proof of Lemma 14

Proof. Let D_1 and D_2 denote the two disks. Without loss of generality, we assume the center of D_1 is higher than that of D_2. The vertical distance between their centers can not exceed $\left(1-\sqrt{1-W^2}\right)$. Otherwise, D_2 would be disjoint from the upper edge of the inscribed rectangle of D_1, and thus cannot cover any point above D_1. The total span of $D_1 \cup D_2$ is therefore at most $1-\sqrt{1-W^2}+1+1 = 3-\sqrt{1-W^2}$. ◀

C 4-Approximation to PDC

In this section, we present a simple 4-approximation to PDC by generalizing the maximal independent set to the outlier case. The definition of partial maximal independent set is presented below.

▶ **Definition 19** (Partial Maximal Independent Set). *Given a point set X of size n and an integer $0 \leq t < n$, a subset $S \subseteq X$ is called a partial maximal independent set if for any distinct points $p, q \in S$, it holds that $d(p,q) > 2$ and $\left|\bigcup_{p\in S} B(p,2) \cap X\right| \geq n-t$.*

The greedy algorithm with time complexity $O(n \log n)$ in [3] can be slightly modified to a 4-approximation algorithm for PDC, which we present in Algorithm 4.

Algorithm 4 Greedy algorithm which outputs a 4-approximation to PDC.

```
Require: A set $X$ of $n$ planar points and an integer $0 \leq t < n$.
    $Y \leftarrow X$, $S \leftarrow \emptyset$
    Sort $Y$ by $x$-coordinate
    while $|Y| > t$ do
        Find the leftmost uncovered point $p$ and $S \leftarrow S \cup \{p\}$
        Place a right semicircle of radius 2 at $p$
        Remove the points covered by the semicircle from $Y$
    return $S$
```

▶ **Lemma 20.** *Algorithm 4 returns a 4-approximation solution to PDC in time $O(n \log n)$.*

Proof. It is easy to verify that S is a partial maximal independent set. Furthermore, for any $p, q \in S$, $p \neq q$, since $d(p,q) > 2$, there is no unit disk that can cover both p and q. Therefore a distinct unit disk is needed to cover each point in S, which implies that $|S| \leq opt(X,t)$. Note that four unit disks are sufficient to cover a semicircle with radius 2. Together with $|\bigcup_{p\in S} B(p,2) \cap X| \geq n-t$, we can obtain $4|S|$ unit disks that cover at least $(n-t)$ points of X. Note that $4|S| \leq 4 \cdot opt(X,t)$, we see that $4|S|$ unit disks make up to a 4-approximation. ◀

Although the algorithm is simple, there is a fatal drawback when applying the algorithm from left to right, it can only detect outliers X_i where $i \geq n-t$ and cannot detect the others. The bad case is that some outliers are far away from the other points, and at the same time, their x coordinates are around the median of $\{X_1, X_2, \ldots, X_n\}$. A unit disk covering such an outlier usually covers few or no other points, and not removing such isolated outliers could greatly increase the number of disks in the solution.

D Partitioning the Plane

Algorithm 5 Partitioning the plane into strips such that their boundary lines do not intersect X.

```
procedure PARTITION($X$)                    ▷ $X$ is a finite set of planar points
    $S \leftarrow \emptyset$
    $R \leftarrow \emptyset$
    for $p \in X$ do
        $S \leftarrow S \cup \{\lfloor x(p)/w \rfloor\}$
        $R \leftarrow R \cup \{x(p)/w - \lfloor x(p)/w \rfloor\}$
    if $0 \in R$ then
        $S \leftarrow \emptyset$
        $\tau \leftarrow \frac{1}{2}\min\{r \in R : r > 0\}$
        for $p \in X$ do
            $S \leftarrow S \cup \{\lfloor x(p)/w - \tau \rfloor\}$
    return $S$
```

Parameterised Counting in Logspace

Anselm Haak
Institut für Theoretische Informatik, Leibniz Universität Hannover, Germany

Arne Meier
Institut für Theoretische Informatik, Leibniz Universität Hannover, Germany

Om Prakash
Department of Computer Science and Engineering, IIT Madras, Chennai, India

Raghavendra Rao B. V.
Department of Computer Science and Engineering, IIT Madras, Chennai, India

Abstract

Logarithmic space bounded complexity classes such as **L** and **NL** play a central role in space bounded computation. The study of counting versions of these complexity classes have lead to several interesting insights into the structure of computational problems such as computing the determinant and counting paths in directed acyclic graphs. Though parameterised complexity theory was initiated roughly three decades ago by Downey and Fellows, a satisfactory study of parameterised logarithmic space bounded computation was developed only in the last decade by Elberfeld, Stockhusen and Tantau (IPEC 2013, Algorithmica 2015).

In this paper, we introduce a new framework for parameterised counting in logspace, inspired by the parameterised space bounded models developed by Elberfeld, Stockhusen and Tantau (IPEC 2013, Algorithmica 2015). They defined the operators $\mathbf{para_W}$ and $\mathbf{para}_\beta$ for parameterised space complexity classes by allowing bounded nondeterminism with multiple-read and read-once access, respectively. Using these operators, they characterised the parameterised complexity of natural problems on graphs. In the spirit of the operators $\mathbf{para_W}$ and $\mathbf{para}_\beta$ by Stockhusen and Tantau, we introduce variants based on tail-nondeterminism, $\mathbf{para_{W[1]}}$ and $\mathbf{para}_{\beta\mathbf{tail}}$. Then, we consider counting versions of all four operators applied to logspace and obtain several natural complete problems for the resulting classes: counting of paths in digraphs, counting first-order models for formulas, and counting graph homomorphisms. Furthermore, we show that the complexity of a parameterised variant of the determinant function for $(0,1)$-matrices is $\#\mathbf{para}_{\beta\mathbf{tail}}\mathbf{L}$-hard and can be written as the difference of two functions in $\#\mathbf{para}_{\beta\mathbf{tail}}\mathbf{L}$. These problems exhibit the richness of the introduced counting classes. Our results further indicate interesting structural characteristics of these classes. For example, we show that the closure of $\#\mathbf{para}_{\beta\mathbf{tail}}\mathbf{L}$ under parameterised logspace parsimonious reductions coincides with $\#\mathbf{para}_\beta\mathbf{L}$, that is, modulo parameterised reductions, tail-nondeterminism with read-once access is the same as read-once nondeterminism.

Initiating the study of closure properties of these parameterised logspace counting classes, we show that all introduced classes are closed under addition and multiplication, and those without tail-nondeterminism are closed under parameterised logspace parsimonious reductions.

Also, we show that the counting classes defined can naturally be characterised by parameterised variants of classes based on branching programs in analogy to the classical counting classes.

Finally, we underline the significance of this topic by providing a promising outlook showing several open problems and options for further directions of research.

2012 ACM Subject Classification Theory of computation → Complexity classes; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized Complexity, Counting Complexity, Logspace

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.40

Related Version *Technical report at arXiv.*: `https://arxiv.org/abs/1904.12156` [36]

Funding Indo-German co-operation grant: DAAD (57388253), DST (INT/FRG/DAAD/P-19/2018).
Arne Meier: Funded by the German Research Foundation (DFG), project ME4279/1-2.

Acknowledgements The authors thank the anonymous referees for their valuable feedback.

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 40; pp. 40:1–40:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Parameterised complexity theory, introduced by Downey and Fellows [25], takes a two-dimensional view on the computational complexity of problems and has revolutionised the algorithmic world. Two-dimensional here refers to the fact that the complexity of a parameterised problem is analysed with respect to the input size n and a parameter k associated with the given input as two independent quantities. The notion of fixed-parameter tractability is the proposed notion of efficient computation. A problem with parameter k is fixed-parameter tractable (fpt, or in the class **FPT**) if there is a deterministic $f(k) \cdot n^{O(1)}$ time algorithm for deciding it, where f is a computable function. The primary notion of intractability is captured by the **W**-hierarchy in this setting.

Since its inception, the focus of parameterised complexity theory has been to identify parameterisations of **NP**-hard problems that allow for efficient parameterised algorithms, and to address structural aspects of the classes in the **W**-hierarchy and related complexity classes [33]. This led to the development of machine-based and logical characterisations of parameterised complexity classes (see the book by Flum and Grohe [33] for more details). While the structure of classes in hierarchies such as the **W**- and **A**-hierarchy is well understood, a parameterised view of parallel and space-bounded computation lacked attention.

In 2013, Elberfeld et al. [43, 28] focused on parameterised space complexity classes and initiated the study of parameterised circuit complexity classes. In fact, they introduced parameterised analogues of deterministic and nondeterministic logarithmic space-bounded classes. The machine-based characterisation of $\mathbf{W}[\mathbf{P}]$ (the class of problems that are fpt-reducible to a weighted circuit satisfiability question), and the type of access to nondeterministic choices (multi-read or read-once) led to two different variants of parameterised logspace (para-logspace), namely, $\mathbf{para_W L}$ and $\mathbf{para}_\beta\mathbf{L}$. Elberfeld et al. [28] obtained several natural complete problems for these classes, such as parameterised variants of reachability in graphs.

Bannach, Stockhusen and Tantau [6] further studied parameterised parallel algorithms. They used colour coding techniques [4] to obtain efficient parameterised parallel algorithms for several natural problems. A year later, Chen and Flum [15, 16] proved parameterised lower bounds for $\mathbf{AC^0}$ by adapting circuit lower bound techniques.

Apart from decision problems, counting problems have found a prominent place in complexity theory. Valiant [46] introduced the notion of counting complexity classes that capture natural counting problems such as counting the number of perfect matchings in a graph, or counting the number of satisfying assignments of a CNF formula. Informally, $\#\mathbf{P}$ (resp., $\#\mathbf{L}$) consists of all functions $F\colon \{0,1\}^* \to \mathbb{N}$ such that there exists an nondeterministic Turing machine (NTM) running in polynomial time (resp., logarithmic space) in the input length whose number of accepting paths on every input $x \in \{0,1\}^*$ is equal to $F(x)$. Valiant's theory of $\#\mathbf{P}$-completeness led to several structural insights into complexity classes around **NP** and interactive proof systems, as well as to the seminal result of Toda [45].

While counting problems in $\#\mathbf{P}$ stayed in the focus of research for long, the study of the determinant by Damm [23], Vinay [47], and Toda [44] established that the complexity of computing the determinant of an integer matrix characterises the class $\#\mathbf{L}$ up to a closure under subtraction. Allender and Ogihara [3] analysed the structure of complexity classes based on $\#\mathbf{L}$. The importance of counting classes based on logspace-bounded Turing machines (TMs) was further established by Allender, Beals and Ogihara [2]. They characterised the complexity of testing feasibility of linear equations by a class which is based on $\#\mathbf{L}$. Beigel and Fu [7] then showed that small depth circuits built with oracle access to $\#\mathbf{L}$ functions lead to a hierarchy of languages which can be seen as the logspace version of the counting hierarchy. In a remarkable result, Ogihara [40] showed that this hierarchy collapses to the first level. Further down the complexity hierarchy, Caussinus et al. [12] introduced counting versions of $\mathbf{NC^1}$ based on various characterisations of $\mathbf{NC^1}$. The counting and probabilistic analogues of $\mathbf{NC^1}$ exhibit properties similar to their logspace counterparts [24]. Moreover, counting and gap variants of the class $\mathbf{AC^0}$ were defined by Agrawal et al. [1].

The theory of parameterised counting classes was pioneered by Flum and Grohe [32] as well as McCartin [39]. The class $\#\mathbf{W}[1]$ consists of all parameterised counting problems that reduce to the problem of counting k-cliques in a graph. Flum and Grohe [32] proved that counting cycles of length k is complete for $\#\mathbf{W}[1]$. Curticapean [18] further showed that counting matchings with k edges in a graph is also complete for $\#\mathbf{W}[1]$. These results led to several remarkable completeness results and new techniques (see, e.g., the works of Curticapean [19, 20], Curticapean, Dell and Marx [21], Jerrum and Meeks [37], Brand and Roth [10]).

Motivation. Given the rich structure of logspace-bounded counting complexity classes, the study of parameterised variants of these classes is vital to obtain a finer classification of counting problems.

A theory on para-logspace counting did not exist before. We wanted to overcome this defect to further understand the landscape of counting problems with decision versions in para-logspace-based classes. Our new framework allows us to classify many of these problems more precisely. In this article, we define counting variants inspired by the parameterised space complexity classes introduced by Elberfeld et al. [43, 28].

In the realm of space-bounded computation, different manners in which nondeterministic bits are accessed lead to different complexity classes. For example, the standard definition of **NL** implicitly gives the corresponding NTMs only read-once access to their nondeterministic bits [5]: nondeterminism is given only in the form of choices between different transitions. This means that nondeterministic bits are not re-accessible by the machine later in the computation. When instead using an auxiliary read-only tape for these bits and allowing for multiple passes on it, one obtains the class **NP**. This is due to the fact that **SAT** is **NP**-complete with respect to logspace many-one reductions [5], and that one can evaluate a CNF formula in deterministic logspace even when the assignment is given on a read-only tape. However, polynomial time bounded NTMs still characterise **NP** even when the machine is allowed to do only one pass on the nondeterministic bits as they can simply store all nondeterministic bits on the work-tape. So, it is very natural to investigate whether the differentiation from above leads to new insights in our setting.

With parameterisation as a means for a finer classification, Stockhusen and Tantau [43] defined nondeterministic logarithmic space-bounded computation based on *how* (unrestricted or read-once) the nondeterministic bits are accessed. Based on this distinction, they defined two operators: $\mathbf{para}_{\mathbf{W}}$ (unrestricted) and $\mathbf{para}_{\beta}$ (read-once). Their study led to many compelling natural problems that are complete for logspace-bounded nondeterministic computations with suitable parameters. Thereby, a rich structure of computational power based on the restrictions on the number of reads of the nondeterministic bits was exhibited. In this article, we additionally differentiate based on *when* (unrestricted or tail access) the nondeterministic bits are accessed. The classes $\mathbf{W}[1]$ and $\mathbf{W}[\mathbf{P}]$ are the two most prominent nondeterministic classes in the parameterised world which is why we wanted to see the effect of such a restriction on the rather small classes in our setting. This leads to the new operators $\mathbf{para}_{\mathbf{W}[1]}$ and $\mathbf{para}_{\beta\mathbf{tail}}$. The concept of tail-nondeterminism allowed to capture the parameterised complexity class $\mathbf{W}[1]$ – via tail-nondeterministic, k-bounded machines – and thereby relates to many interesting problems such as searching for cliques, independent sets, or homomorphism, and evaluating conjunctive queries [33]. Intuitively, tail-nondeterminism means that all nondeterministic bits are read at the end of the computation, and k-boundedness limits the number of these nondeterministic bits to $f(k) \cdot \log |x|$ for all inputs (x, k).

Studying counting complexity often improves the understanding of related classical problems and classes (e.g., Toda's theorem [45]). With regard to space-bounded complexity, there are several characterisations of logspace-bounded counting classes in terms of natural problems. For example, counting paths in directed graphs is complete for $\#\mathbf{L}$, and checking if an integer matrix is singular or not is complete for the class $\mathbf{C_{=}L}$ (see Allender et al. [2]). Furthermore, testing if a system of linear equations is feasible or not can be done in $\mathbf{L}$ with queries to any complete language for $\mathbf{C_{=}L}$. Moreover, two hierarchies built over counting classes for logarithmic space collapse either to the first level [40] or to the second level [2]. Apart from this, the separation of various counting classes over logarithmic space remains widely open. For example, it is not known if the class $\mathbf{C_{=}L}$ is closed under complementation.

We consider different parameterised variants of the logspace-bounded counting class $\#\mathbf{L}$ to give a new perspective on its fine structure.

Results. We introduce the counting variants of parameterised space-bounded computation and show that each of the parameterised logspace complexity classes, defined by Stockhusen and Tantau [43], has a natural counting counterpart. Moreover, by considering also tail-nondeterminism with respect to their classes, we obtain four different variants of parameterised logspace counting classes, namely, $\#\mathbf{para_{W}L}$, $\#\mathbf{para}_{\beta}\mathbf{L}$, $\#\mathbf{para_{W[1]}L}$, and $\#\mathbf{para}_{\beta\mathbf{tail}}\mathbf{L}$. We show that $\#\mathbf{para_{W}L}$ and $\#\mathbf{para}_{\beta}\mathbf{L}$ are closed under para-logspace parsimonious reductions and that all of our new classes are closed under addition and multiplication.

Furthermore, we develop a complexity theory by obtaining natural complete problems for these new classes. We introduce variants of the problem of counting walks of parameter-bounded length that are complete for the classes $\#\mathbf{para}_{\beta}\mathbf{L}$ (Theorems 14, 15 and 18), $\#\mathbf{para}_{\beta\mathbf{tail}}\mathbf{L}$ (Theorem 16) and $\#\mathbf{para_{W}L}$ (Theorem 19). Since the same problem is shown to be complete for both, $\#\mathbf{para}_{\beta}\mathbf{L}$ and $\#\mathbf{para}_{\beta\mathbf{tail}}\mathbf{L}$, we get the somewhat surprising result that the closure of $\#\mathbf{para}_{\beta\mathbf{tail}}\mathbf{L}$ under para-logspace parsimonious reductions coincides with $\#\mathbf{para}_{\beta}\mathbf{L}$ (Corollary 17). Also, we show that a parameterised version of the problem of counting homomorphisms from coloured path structures to arbitrary structures is complete for $\#\mathbf{para}_{\beta}\mathbf{L}$ with respect to para-logspace parsimonious reductions (Theorem 28).

Afterwards, we study variants of the problem of counting assignments to free first-order variables in a quantifier-free FO formula. We identify complete problems for the classes $\#\mathbf{para}_{\beta}\mathbf{L}$ and $\#\mathbf{para_{W[1]}L}$ in this context. More specifically, counting assignments to free first-order variables in a quantifier-free formula with relation symbols of bounded arity and the syntactical locality of the variables in the formula being restricted (p-$\#\mathrm{MC}(\Sigma_0^{r\text{-local}})_a$) is shown to be complete for the classes $\#\mathbf{para}_{\beta\mathbf{tail}}\mathbf{L}$ and $\#\mathbf{para}_{\beta}\mathbf{L}$ with respect to para-logspace parsimonious reductions (Theorem 22). When there is no restriction on the arity of relational symbols or on the locality of the variables, counting the number of satisfying assignments to free first-order variables in a quantifier-free formula in a given structure (p-$\#\mathrm{MC}(\Sigma_0)$) is complete for $\#\mathbf{para_{W[1]}L}$ with respect to para-logspace parsimonious reductions (Theorem 23).

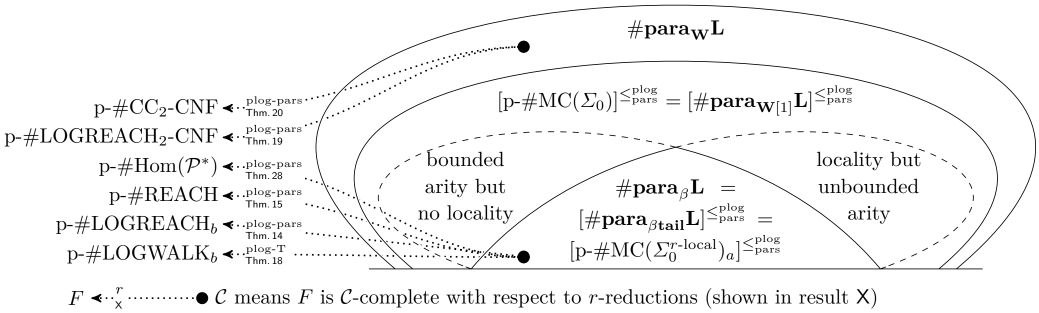

$\#\mathbf{para_W L}$
$[\text{p-}\#\mathrm{MC}(\Sigma_0)]^{\leq^{\text{plog}}_{\text{pars}}} = [\#\mathbf{para_{W[1]}L}]^{\leq^{\text{plog}}_{\text{pars}}}$
bounded arity but no locality
$\#\mathbf{para}_\beta\mathbf{L} = [\#\mathbf{para}_{\beta\mathbf{tail}}\mathbf{L}]^{\leq^{\text{plog}}_{\text{pars}}} = [\text{p-}\#\mathrm{MC}(\Sigma_0^{r\text{-local}})_a]^{\leq^{\text{plog}}_{\text{pars}}}$
locality but unbounded arity
p-#CC$_2$-CNF plog-pars Thm. 20
p-#LOGREACH$_2$-CNF plog-pars Thm. 19
p-#Hom($\mathcal{P}^*$) plog-pars Thm. 28
p-#REACH plog-pars Thm. 15
p-#LOGREACH$_b$ plog-pars Thm. 14
p-#LOGWALK$_b$ plog-T Thm. 18
F r X ● $\mathcal{C}$ means F is $\mathcal{C}$-complete with respect to r-reductions (shown in result X)

Figure 1 Diagram assuming pair-wise difference of studied classes with list of complete problems.

Finally, we consider a parameterised variant of the determinant function (p-det) introduced by Chauhan and Rao [13]. By adapting the arguments of Mahajan and Vinay [38], we show that p-det on $(0, 1)$-matrices can be expressed as the difference of two functions in $\#\mathbf{para}_\beta\mathbf{L}$, and is $\#\mathbf{para}_{\beta\mathbf{tail}}\mathbf{L}$-hard with respect to para-logspace many-one reductions (Theorem 33).

Figure 1 shows a class diagram with complete problems.

Main Techniques. Our primary contribution is laying foundations for the study of parameterised logspace-bounded counting complexity classes. The completeness results in Theorems 15 and 23 required a quantised normal form for k-bounded nondeterministic Turing Machines (NTMs) (Lemma 8). This normal form quantises the nondeterministic steps of a k-bounded NTM into chunks of $\log n$-many steps such that the total number of accepting paths remains the same. We believe that the normal form given in Lemma 8 will be useful in the structural study of parameterised counting classes. The study of p-det involved definitions of so-called parameterised clow sequences generalising the classical notion [38]. Besides, a careful assignment of signs to clow sequences was necessary for our complexity analysis of p-det.

Related Results. Chen and Müller [14] studied the parameterised complexity of counting homomorphisms and divided the problems into four equivalence classes. However, their equivalence is only based on reductions among variants of counting homomorphisms but not in terms of concrete complexity classes. In this context, Dalmau and Johnson [22] investigated the complexity of counting homomorphisms as well, and provided generalisations of results by Grohe [34] to the counting setting. A similar classification regarding our classes can give new insights into the complexity of the homomorphism problem (Open Problem 29). The behaviour of our classes with respect to reductions is similar to the one observed for $\mathbf{W}[1]$ by Bottesch [8, 9].

Outline. In Section 2, we introduce the considered machine model, as well as needed foundations of parameterised complexity theory, and logic. Section 3 presents structural results regarding our introduced notions in the parameterised counting context. Afterwards, in Section 4, our main results on counting walks, FO-assignments, homomorphisms as well as regarding the determinant are shown. Finally, we conclude in Section 5.

Due to space limitations, all proof details can be found in the technical report [36].

2 Preliminaries

In this section, we describe the computational models and complexity classes that are relevant for parameterised complexity theory. We use standard notions and notations from parameterised complexity theory [25, 33]. Without loss of generality, we restrict the input alphabet to be $\{0,1\}$.

Turing Machines (TMs) with Random Access to the Input. We consider an intermediate model between TMs and Random Access Machines (RAMs) on words. Particularly, we make use of TMs that have random access to the input tape and can query relations in input structures in constant time. This can be achieved with two additional tapes of logarithmic size (in the input length), called the *random access tape* and the *relation query tape*. On the former, the machine can write the index of an input position to get the value of the respective bit of the input. On the relation query tape, the machine can write a tuple t of the input structure together with a relation identifier R to get the bit stating whether t is in the relation specified by R. Note that our model achieves linear speed-up for accessing the input compared to the standard TM model. (This is further justified by Remark 6.) For convenience, in the following, whenever we speak about TMs we mean the TM model with random access to the input. Denote by $\mathbf{SPACETIME}(s,t)$ ($\mathbf{NSPACETIME}(s,t)$) with $s,t\colon \mathbb{N}\to\mathbb{N}$ the class of languages that are accepted by (nondeterministic) TMs with space-bound $O(s(n))$ and time-bound $O(t(n))$. A $\mathcal{C}$-machine for $\mathcal{C}=\mathbf{SPACETIME}(s,t)$ ($\mathcal{C}=\mathbf{NSPACETIME}(s,t)$) is a (nondeterministic) TM that is $O(s(n))$ space-bounded and $O(t(n))$ time-bounded.

NTMs are a generalisation of TMs where multiple transitions from a given configuration are allowed. This can be formalised by allowing the transition to be a relation rather than a function. Sometimes, it is helpful to view NTMs as deterministic TMs with an additional tape, called the (nondeterministic) choice tape which is read-only. Let M be a deterministic TM with a choice tape. A nondeterministic step in the computation of M is a step where M moves the head on the choice tape to a cell that was not visited before. The *language accepted by* M, $L(M)$ is defined as

$$\{\, x\in\{0,1\}^* \mid \exists y\in\{0,1\}^* \text{ s.t. } M \text{ accepts } x \text{ when the choice tape is initialised with } y \,\}.$$

Notice that in this framework the machine M has two-way access to the choice tape. Furthermore, resource bounds are with respect to the input only (the content of the choice tape is not part of the input) and the choice tape is not counted for space bounds. In this paper, we regard nondeterministic TMs as deterministic ones with a choice tape.

Before we proceed to the definition of parameterised complexity classes, a clarification on the choice of the model is due. Note that RAMs and NRAMs are often appropriate in the parameterised setting as exhibited by several authors (see, e.g., the textbook of Flum and Grohe [33]). They allow to define bounded nondeterminism quite naturally. On the other hand, in the classical setting, branching programs (BPs) are one of the fundamental models that represent space bounded computation, in particular logarithmic space. Since BPs inherently use bit access, this relationship suggests the use of a bit access model. Consequently, we consider a hybrid computational model: Turing machines with random access to the input. While the computational power of this model is the same as that of Turing machines and RAMs, it seems to be a natural choice to guarantee a certain robustness, allowing for desirable characterisations of our classes.

Parameterised Complexity Classes. Let $\mathbf{FPT}$ denote the set of parameterised problems that can be decided by a deterministic TM running in time $f(k) \cdot p(|x|)$ for any input (x, k) where f is a computable function and p is a polynomial. Two central classes in parameterised complexity theory are $\mathbf{W}[1]$ and $\mathbf{W}[\mathbf{P}]$ which were originally defined via special types of circuit satisfiability [33]. Flum, Chen and Grohe [17] obtained a characterisation of these two classes using the following notion of k-bounded NTMs.

▶ **Definition 1** (k-bounded TMs). *An NTM M, working on inputs of the form (x, k) with $x \in \{0,1\}^*, k \in \mathbb{N}$, is said to be* k-bounded *if for all inputs (x, k) it reads at most $f(k) \cdot \log|x|$ bits from the choice tape on input (x, k), where f is a computable function.*

Here, we will work with the following characterisation of $\mathbf{W}[\mathbf{P}]$. The characterisation for $\mathbf{W}[1]$ needs another concept that will be defined on the next page.

▶ **Proposition 2** ([17, 33]). $\mathbf{W}[\mathbf{P}]$ *is the set of all parameterised problems that can be accepted by k-bounded* $\mathbf{FPT}$*-machines with a choice tape.*

Now, we recall three complexity theoretic operators that define parameterised complexity classes from an arbitrary classical complexity class, namely $\mathbf{para}, \mathbf{para}_{\mathbf{W}}$ and $\mathbf{para}_{\beta}$, following the notation of Stockhusen [42].

▶ **Definition 3** ([31]). *Let $\mathcal{C}$ be any complexity class. Then* $\mathbf{para}\mathcal{C}$ *is the class of all parameterised problems $P \subseteq \{0,1\}^* \times \mathbb{N}$ for which there is a computable function $\pi\colon \mathbb{N} \to \{0,1\}^*$ and a language $L \in \mathcal{C}$ with $L \subseteq \{0,1\}^* \times \{0,1\}^*$ such that for all $x \in \{0,1\}^*$, $k \in \mathbb{N}$: $(x, k) \in P \Leftrightarrow (x, \pi(k)) \in L$.*

Notice that $\mathbf{paraP} = \mathbf{FPT}$ is the standard precomputation characterisation of $\mathbf{FPT}$ [31]. A $\mathbf{para}\mathcal{C}$-machine for $\mathcal{C} = \mathbf{SPACETIME}(s, t)$ ($\mathcal{C} = \mathbf{NSPACETIME}(s, t)$) is a (nondeterministic) TM, working on inputs of the form (x, k), that is $O(s(|x| + f(k)))$ space-bounded and $O(t(|x| + f(k)))$ time-bounded where f is a computable function.

The class $\mathbf{XP}$ (problems accepted in time $|x|^{f(k)}$ for a computable function f) and the $\mathbf{W}$-hierarchy [33] capture intractability of parameterised problems. Though the $\mathbf{W}$-hierarchy was defined using the weighted satisfiability of formulas with bounded weft, which is the number of alternations between gates of high fan-in, Flum and Grohe [31] characterised central classes in this context using bounded nondeterminism. Stockhusen and Tantau [43, 42] considered space-bounded and circuit-based parallel complexity classes with bounded nondeterminism.

The following definition is a more formal version of the one given by Stockhusen and Tantau [43, Def. 2.1]. They use $\mathrm{para}\exists^{\leftrightarrow}_{f \log}\mathcal{C}$ instead of $\mathbf{para}_{\mathbf{W}}\mathcal{C}$ for a complexity class $\mathcal{C}$.

▶ **Definition 4.** *Let $\mathcal{C} = \mathbf{SPACETIME}(s, t)$ for some $s, t\colon \mathbb{N} \to \mathbb{N}$. Then,* $\mathbf{para}_{\mathbf{W}}\mathcal{C}$ *is the class of all parameterised problems Q that can be accepted by a k-bounded* $\mathbf{para}\mathcal{C}$*-machine with a choice tape.*

For example, $\mathbf{para}_{\mathbf{W}}\mathbf{L}$ denotes the parameterised version of $\mathbf{NL}$ with k-bounded nondeterminism. One can also restrict this model by only giving one-way access to the nondeterministic tape. The following definition is a more formal version of the one of Stockhusen and Tantau [43, Def. 2.1] who use the symbol $\mathrm{para}\exists^{\rightarrow}_{f \log}$ instead.

▶ **Definition 5.** *Let $\mathcal{C} = \mathbf{SPACETIME}(s, t)$ for some $s, t\colon \mathbb{N} \to \mathbb{N}$. Then* $\mathbf{para}_{\beta}\mathcal{C}$ *denotes the class of all parameterised problems Q that can be accepted by a k-bounded* $\mathbf{para}\mathcal{C}$*-machine with a choice tape with one-way read access to the choice tape.*

As there is only read-once access to the nondeterministic bits, $\mathbf{para}_\beta\mathcal{C}$ can be equivalently defined via nondeterministic transitions and without using a choice tape.

Another notion studied in parameterised complexity is tail-nondeterminism. A k-bounded machine M is *tail-nondeterministic* if there exists a computable function g such that on all inputs (x, k), M makes at most $g(k) \cdot \log n$ further steps in the computation, after its first nondeterministic step. The value of this concept is evidenced by the machine characterisation of $\mathbf{W}[1]$ (Chen et al. [17]). We hope to get new insights by transferring this concept to space-bounded computation. In consequence, we introduce the tail-nondeterministic versions of $\mathbf{para}_{\mathbf{W}}\mathcal{C}$ and $\mathbf{para}_\beta\mathcal{C}$ which are denoted by $\mathbf{para}_{\mathbf{W}[1]}\mathcal{C}$ and $\mathbf{para}_{\beta\mathbf{tail}}\mathcal{C}$.

▶ Remark 6. Note that it is important to have random access to the input tape in the case of tail-nondeterminism. Without random access to input bits and input relations, a TM cannot even make reasonable queries to the input in time $g(k) \cdot \log(n)$.

Logic. We assume basic familiarity with first-order logic (FO). A *vocabulary* is a finite ordered set of relation symbols and constants. Each relation symbol R has an associated *arity* $\mathrm{arity}(R) \in \mathbb{N}$. Let τ be a vocabulary. A *τ-structure* $\mathbf{A}$ consists of a nonempty finite set $\mathrm{dom}(\mathbf{A})$ (its *universe*), and an *interpretation* $R^{\mathbf{A}} \subseteq \mathrm{dom}(\mathbf{A})^{\mathrm{arity}(R)}$ for every relation symbol $R \in \tau$. Syntax and semantics are defined as usual (see, e.g., the textbook of Ebbinghaus et al. [27]). Let $\mathbf{A}$ be a structure with universe A. We denote by $|\mathbf{A}|$ the *size of a binary encoding of* $\mathbf{A}$, i.e., the number of bits required to represent the universe and relations as lists of tuples. For example, if R is a relation of arity 3, then $R^{\mathbf{A}}$ is represented as a subset of A^3, i.e., a set of triples over A. This requires $O(|R^{\mathbf{A}}| \cdot \mathrm{arity}(R)) \cdot \log |A|)$ bits to represent the relation $R^{\mathbf{A}}$, assuming $\log |A|$ bits to represent an element in A. As analysed by Flum et al. [30, Sect. 2.3], this means that $|\mathbf{A}| \in \Theta((|A| + |\tau| + \sum_{R \in \tau} |R^{\mathbf{A}}| \cdot \mathrm{arity}(R)) \cdot \log |A|)$. Also recall that the fragment Σ_i (for $i \in \mathbb{N}$) refers to the class of FO-formulas with i quantifier blocks alternating between existential and universal quantifiers and the outermost quantifier being existential.

3 Parameterised Counting in Logarithmic Space

Now, we define the counting counterparts based on the parameterised classes defined using bounded nondeterminism. The definitions of the decision classes based on tail-nondeterminism can be found in the technical report [36]. A *parameterised function* is a function $F \colon \{0,1\}^* \times \mathbb{N} \to \mathbb{N}$. For an input (x, k) of F with $x \in \{0,1\}^*$, $k \in \mathbb{N}$, we call k the *parameter* of that input. If $\mathcal{C}$ is a complexity class and a parameterised function F belongs to $\mathcal{C}$, we say that F is $\mathcal{C}$-computable. Let M be a TM. We denote by $\mathrm{acc}_M(x)$ the number of accepting paths of M on input x, and similarly, $\mathrm{acc}_M(x, k)$, for parameterised inputs of the form (x, k).

▶ **Definition 7.** *Let $\mathcal{C} = \mathbf{SPACETIME}(s, t)$ for some $s, t \colon \mathbb{N} \to \mathbb{N}$. Then a parameterised function F is in $\#\mathbf{para}_{\mathbf{W}}\mathcal{C}$ if there is a k-bounded nondeterministic $\mathbf{para}\mathcal{C}$-machine M such that for all inputs (x, k), we have that $\mathrm{acc}_M(x, k) = F(x, k)$. Furthermore, F is in*

- *$\#\mathbf{para}_\beta\mathcal{C}$ if there is such an M with read-once access to its nondeterministic bits,*
- *$\#\mathbf{para}_{\mathbf{W}[1]}\mathcal{C}$ if there is such an M that is tail-nondeterministic, and*
- *$\#\mathbf{para}_{\beta\mathbf{tail}}\mathcal{C}$ if there is such an M with read-once access to its nondeterministic bits that is tail-nondeterministic.*

By definition, we get $\#\mathbf{para}_{\beta\mathbf{tail}}\mathbf{L} \subseteq \mathcal{C} \subseteq \#\mathbf{para}_{\mathbf{W}}\mathbf{L}$ for $\mathcal{C} \in \{\#\mathbf{para}_\beta\mathbf{L}, \#\mathbf{para}_{\mathbf{W}[1]}\mathbf{L}\}$. Note that the restriction of the above classes to k-boundedness is crucial. If we drop this restriction, the machines are able to access $2^{f(k)+log|x|}$, so fpt-many, nondeterministic bits.

Regarding multiple-read access, this allows for solving SAT (with constant parameterisation). So this class then would contain a **paraNP**-complete problem. For read-once access, we expect a similar result for **paraNL**. When adding tail-nondeterminism, we implicitly get k-boundedness again, so this does not lead to new classes.

The following lemma shows that **paraL**-machines can be normalised in a specific way. This normalisation will play a major role in Section 4.

▶ **Lemma 8.** *For any k-bounded nondeterministic* **paraL***-machine M there exists a k-bounded nondeterministic* **paraL***-machine M' with $\#\mathrm{acc}_M(x,k) = \#\mathrm{acc}_{M'}(x,k)$ for all inputs (x,k) such that M' has the following properties:*

(1) *M' has a unique accepting configuration,*

(2) *on any input (x,k), every computation path of M' accesses exactly $g(k)\cdot\log|x|$ nondeterministic bits (for some computable function g), and M' counts on an extra tape (tape S) the number of nondeterministic steps, and*

(3) *M' has an extra tape (tape C) on which it remembers previous nondeterministic bits, resetting the tape after every $\log|x|$-many nondeterministic steps.*

Additionally, if M has read-once access to its nondeterministic bits, or is tail-nondeterministic, or both, then M' also has these properties.

The following result follows from a simple simulation of nondeterministic machines by deterministic ones. Let **FFPT** be the class of functions computable by **FPT**-machines with output.

▶ **Theorem 9.** $\#\mathbf{para}_\beta\mathbf{L} \subseteq \mathbf{FFPT}$*.*

Using the notion of oracle machines (see, e.g., [41]), we define Turing, metric, and parsimonious reductions computable in **paraL**. For our purposes, the oracle tape is always exempted from space restrictions which is often the case in the context of logspace Turing reductions [11]. A study on the effect of changing this assumption might be interesting.

▶ **Definition 10** (Reducibilities)**.** *Let $F, F'\colon \{0,1\}^* \times \mathbb{N} \to \mathbb{N}$ be two functions. Then, F is* para-logspace Turing reducible *to F', $F \le_T^{\mathrm{plog}} F'$, if there is a* **paraL** *oracle TM M that computes F with oracle F' and the parameter of any oracle query of M is bounded by a computable function in the parameter. If there is such an M that uses only one oracle query, then F is* para-logspace metrically reducible *to F', $F \le_{\mathrm{met}}^{\mathrm{plog}} F'$. If there is such an M that returns the answer of the first oracle query, then F is* para-logspace parsimoniously reducible *to F', $F \le_{\mathrm{pars}}^{\mathrm{plog}} F'$.*

Note that the definition of parsimonious reductions ensures that the size of the witness set is preserved by the fact that M immediately returns the answer of its only oracle query (without further computations). For any reducibility relation $\preccurlyeq$ and any complexity class $\mathcal{C}$, $[\mathcal{C}]^{\preccurlyeq} := \{\, A \mid \exists C \in \mathcal{C} \text{ such that } A \preccurlyeq C \,\}$ is the *$\preccurlyeq$-closure of $\mathcal{C}$*.

Next, we show that both new classes without tail-nondeterminism are closed under $\le_{\mathrm{pars}}^{\mathrm{plog}}$.

▶ **Lemma 11.** *The classes $\#\mathbf{para}_{\mathbf{W}}\mathbf{L}$ and $\#\mathbf{para}_\beta\mathbf{L}$ are closed under $\le_{\mathrm{pars}}^{\mathrm{plog}}$.*

For the tail-classes, such a closure property is not obvious. Corollary 16 will show that closing the class with read-once access and tail-nondeterminism under these reductions gives the full power of the class without tail-nondeterminism. Open Problem 24 on page 12 asks what class is obtained when closing the class without read-once access and with tail-nondeterminism.

Another important question is whether classes are closed under certain arithmetic operations. We show that all newly introduced classes are closed under addition and multiplication.

▶ **Theorem 12.** *For any $o \in \{\mathbf{W}, \mathbf{W}[1], \beta, \beta\textbf{-tail}\}$, the class $\#\mathbf{para}_o\text{-}\mathbf{L}$ is closed under addition and multiplication.*

▶ **Open Problem 13.** *Which of the classes are closed under* monus*, that is,* $\max\{F - G, 0\}$*?*

4 Complete Problems

This section studies complete problems for the previously defined classes: counting problems in the context of walks in directed graphs, model-checking problems for FO formulas, and homomorphisms between FO-structures as well as a parameterised version of the determinant.

4.1 Counting Walks

We start with parameterised variants of counting walks in directed graphs which will be shown to be complete for the introduced classes.

Problem:	p-#LOGREACH$_b$
Input:	directed graph $\mathfrak{G} = (V, E)$ with out-degree b, $s, t \in V$ and $a, k \in \mathbb{N}$.
Parameter:	k.
Output:	number of s-t-walks of length a if $a \leq k \cdot \log |V|$, 0 otherwise.

▶ **Theorem 14.** *For every $b \geq 2$,* p-#LOGREACH$_b$ *is $\#\mathbf{para}_\beta\mathbf{L}$-complete with respect to $\leq^{\text{plog}}_{\text{pars}}$-reductions.*

Proof Idea. For the upper bound, guess a path of length exactly a. The number of nondeterministic bits is bounded by $O(k \cdot \log |V|)$ since successors can be referenced by a number in $\{0, \ldots, b-1\}$.

For the lower bound, using Lemma 8, construct the configuration graph $\mathfrak{G}$ restricted to nondeterministic configurations and the unique accepting configuration C_{acc}, where the edge relation expresses whether a configuration is reachable with exactly one nondeterministic, but an arbitrary number of deterministic steps. Accepting computations of the machine correspond to paths from the first nondeterministic configuration to C_{acc} of length $f(k) \cdot \log |\mathfrak{G}|$ in $\mathfrak{G}$. ◀

Now consider the problem p-#REACH, defined as follows.

Problem:	p-#REACH
Input:	directed graph $\mathfrak{G} = (V, E)$, $s, t \in V$, $k \in \mathbb{N}$.
Parameter:	k.
Output:	number of s-t-walks of length exactly k.

The difference to the previous problem is the unbounded out-degree of nodes and the length of counted walks being k instead of $a \leq k \cdot \log |x|$. Note that the analogue problem for counting paths is $\#\mathbf{W}[1]$-complete [32]. However, we will see now that the problem for walks is $\#\mathbf{para}_\beta\mathbf{L}$-complete.

▶ **Theorem 15.** p-#REACH *is $\#\mathbf{para}_\beta\mathbf{L}$-complete with respect to $\leq^{\text{plog}}_{\text{pars}}$.*

As the length of paths that are counted in p-#REACH is k, the runtime of the whole algorithm used to prove membership in the previous theorem is actually bounded by $k \cdot \log |x|$ on input (x, k). This means that the computation is tail-nondeterministic.

▶ **Theorem 16.** p-#REACH *is* $\#\mathbf{para}_{\beta\mathbf{tail}}\mathbf{L}$*-complete with respect to* $\leq^{\text{plog}}_{\text{pars}}$.

The previous results together with the fact that $\#\mathbf{para}_{\beta}\mathbf{L}$ is closed under $\leq^{\text{plog}}_{\text{pars}}$ yield the following surprising collapse (a similar behaviour was observed by Bottesch [8, 9]).

▶ **Corollary 17.** $[\#\mathbf{para}_{\beta\mathbf{tail}}\mathbf{L}]^{\leq^{\text{plog}}_{\text{pars}}} = \#\mathbf{para}_{\beta}\mathbf{L}$*.*

We continue with another variant of p-#LOGREACH$_b$, namely p-#LOGWALK$_b$. Here, all walks of length a are counted, if $a \leq k \cdot \log |x|$ (and s, t are not part of the input).

▶ **Theorem 18.** p-#LOGWALK$_b$ *is* $\#\mathbf{para}_{\beta}\mathbf{L}$*-complete with respect to* $\leq^{\text{plog}}_{T}$.

Now, consider a problem that combines a reachability problem with model-checking for propositional logic, that is, it only counts walks that are models of a propositional formula (see Haak et al. [35]). Let $\mathfrak{G} = (V, E)$ be a DAG, $(e_1, \ldots, e_n)$ be a walk in $\mathfrak{G}$ with $e_i \in E$ for $1 \leq i \leq n$, and $P = \{e_1, \ldots, e_n\}$. Define the function $c_P \colon E \to \{0, 1\}$ to be the characteristic function of P with respect to E: $c_P(e) = 1$ iff $e \in P$.

Problem:	p-#LOGREACH$_2$-CNF
Input:	directed graph $\mathfrak{G} = (V, E)$ of out-degree 2, $s, t \in V$, CNF formula φ with $\text{Vars}(\varphi) \subseteq E$, $a, k \in \mathbb{N}$.
Parameter:	k.
Output:	Number of s-t-walks $(s = e_1, \ldots, e_a = t)$ such that $c_P \models \varphi$, where $P = \{e_1, \ldots, e_a\}$, if $a \leq k \cdot \log(\lvert V\rvert + \lvert\varphi\rvert)$, 0 otherwise.

▶ **Theorem 19.** p-#LOGREACH$_2$-CNF *is* $\#\mathbf{para}_{W}\mathbf{L}$*-complete with respect to* $\leq^{\text{plog}}_{\text{pars}}$.

Proof Idea. Regarding membership, we can first use the algorithm outlined in the proof idea of Theorem 14 to nondeterministically guess a path, and then use the standard logspace model-checking algorithm for propositional formulas. Since edges in the graph are associated with variables of the formula, whenever we need the value of an edge variable e, we run the original algorithm re-using nondeterministic bits to determine it.

For the lower bound, we use the same graph as in Theorem 14, and the formula is used to express consistency of the re-used nondeterministic bits in the configuration graph. ◀

Similarly, define the problem p-#CC$_2$-CNF: Given a graph $\mathfrak{G} = (V, E)$ of bounded out-degree 2, a CNF-formula φ with $\text{Vars}(\varphi) \subseteq E$ and $a, k \in \mathbb{N}$, with k as the parameter and $a \leq \log(\lvert G\rvert + \lvert\varphi\rvert)$, output the number of cycle covers $D \subseteq E$ in which the number of non-selfloop-cycles is $\leq k$, exactly $k \cdot a$ vertices are covered non-trivially and $\text{Vars}(D) \models \varphi$.

▶ **Theorem 20.** p-#CC$_2$-CNF *is* $\#\mathbf{para}_{W}\mathbf{L}$*-complete with respect to* $\leq^{\text{plog}}_{\text{pars}}$.

4.2 Counting FO-Assignments

Let $\mathcal{F}$ be a class of well-formed formulas. The problem of counting satisfying assignments to free FO-variables in $\mathcal{F}$-formulas, *p*-#MC($\mathcal{F}$), is defined as follows.

Problem:	*p*-#MC($\mathcal{F}$)
Input:	formula $\varphi \in \mathcal{F}$, structure $\mathbf{A}$, $k \in \mathbb{N}$.
Parameter:	$\lvert\varphi\rvert$.
Output:	$\lvert\varphi(\mathbf{A})\rvert$ if $k = \lvert\varphi\rvert$, 0 otherwise.

Here, $\varphi(\mathbf{A})$ is the set of satisfying assignments of φ in $\mathbf{A}$:

$$\varphi(\mathbf{A}) = \{\,(a_1,\dots,a_n) \mid (a_1,\dots,a_n) \in \mathrm{dom}(\mathbf{A})^n, \mathbf{A} \models \varphi(a_1,\dots,a_n)\,\}.$$

Denote by p-#MC$(\mathcal{F})_a$ the variant where for all relations the arity is at most $a \in \mathbb{N}$. We investigate parameterisations that yield complete problems for some of the new classes in this setting.

In particular, we consider a fragment of FO obtained by restricting the occurrence of variables in the syntactic tree of a formula in a purely syntactic manner. Formally, the *syntax tree* of a quantifier-free FO-formula φ is a tree with edge-ordering whose leaves are labelled by atoms of φ and whose inner vertices are labelled by Boolean connectives.

▶ **Definition 21.** *Let $r \in \mathbb{N}$ and φ be a quantifier-free* FO*-formula. Let $\theta_1,\dots,\theta_m$ be the atoms of φ in the order of their occurrence in the order-respecting depth-first run through the syntax tree of φ. We say that φ is* r-local *if for any θ_i, θ_j that involve the same variable, we have $|i-j| \le r$. We define $\Sigma_0^{r\text{-}local} := \{\,\varphi \in \Sigma_0 \mid \varphi \text{ is } r\text{-local}\}$.*

Using this syntactic notion, we obtain a complete problem for our classes with read-once access to nondeterministic bits in the setting of first-order model-checking.

▶ **Theorem 22.** *For $a \ge 2, r \ge 1$,* p-#MC$(\Sigma_0^{r\text{-}local})_a$ *is* $\#\mathbf{para}_\beta\mathbf{L}$*-complete and* $\#\mathbf{para}_{\beta\mathbf{tail}}\mathbf{L}$*-complete with respect to* $\le^{\mathrm{plog}}_{\mathrm{pars}}$.

Proof Idea. Regarding membership, we evaluate the given φ in $\mathbf{A}$ top to bottom using the locality of φ by storing assignments to variables until we encountered r more atoms. As a result, at most $a \cdot r$ assignments to variables are simultaneously stored and each one needs $\log|\mathbf{A}|$ space. Moreover, the runtime of the whole procedure is bounded by $f(|\varphi|) \cdot \log|\mathbf{A}|$ for some computable function f and thereby the procedure is tail-nondeterministic.

Regarding the lower bound, we reduce from p-#REACH and use the formula

$$\varphi_k(x_1,\dots,x_k) := (x_1 = s) \wedge E(x_1,x_2) \wedge E(x_2,x_3) \wedge \ldots \wedge E(x_{k-1},x_k) \wedge x_k = t$$

expressing that a tuple of vertices $(v_1,\dots,v_k)$ is an $s^{\mathbf{A}}$-$t^{\mathbf{A}}$-walk in an (E,s,t)-structure $\mathbf{A}$. ◀

Note that the decision version of p-#MC(Σ_0) is equivalent to parameterised model-checking for Σ_1-sentences, as we count assignments to free variables. This problem characterises tail-nondeterministic para-logspace with read-once access to nondeterministic bits.

▶ **Theorem 23.** *p-#MC(Σ_0) is* $\#\mathbf{para}_{\mathbf{W}[1]}\mathbf{L}$*-complete with respect to* $\le^{\mathrm{plog}}_{\mathrm{pars}}$.

The complexity status of counting assignments to free first-order variables in a Σ_0 formula with unbounded arity or without the local restrictions is not known. In particular, it is not clear if the restriction on the arity or the locality property of the formula can be removed while preserving completeness. Finally, we close this section with three open questions.

▶ **Open Problem 24.** *What is the complexity of* $[\text{p-\#MC}(\Sigma_0)_a]^{\le^{\mathrm{plog}}_{\mathrm{pars}}}$ *for fixed $a \in \mathbb{N}$? What is the complexity of* $[\text{p-\#MC}(\Sigma_0^{r\text{-}local})]^{\le^{\mathrm{plog}}_{\mathrm{pars}}}$ *for fixed $r \in \mathbb{N}$?*

▶ **Open Problem 25.** *Is the class* $[\#\mathbf{para}_{\mathbf{W}[1]}\mathbf{L}]^{\le^{\mathrm{plog}}_{\mathrm{pars}}}$ *equivalent to some known class?*

4.3 Counting Homomorphisms

This subsection is devoted to the study of the problem of counting homomorphisms between two structures in the parameterised setting. Typically, the size of the universe of the first structure is considered as the parameter. The complexity of counting homomorphisms has been intensively investigated for almost two decades [26, 34, 22, 14].

▶ **Definition 26** (Homomorphism)**.** *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be structures over some vocabulary* τ *with universes* A *and* B*, respectively. A function* $h\colon A \to B$ *is a* homomorphism from $\mathbf{A}$ to $\mathbf{B}$ *if for all* $R \in \tau$ *and for all tuples* $(a_1, \ldots, a_{\mathrm{arity}(R)}) \in R^{\mathbf{A}}$*, we have* $(h(a_1), \ldots, h(a_{\mathrm{arity}(R)})) \in R^{\mathbf{B}}$*.*

A bijective homomorphism h between two structures $\mathbf{A}, \mathbf{B}$ such that the inverse of h is also a homomorphism is called an *isomorphism.* If there is an isomorphism between $\mathbf{A}$ and $\mathbf{B}$, then $\mathbf{A}$ is said to be *isomorphic* to $\mathbf{B}$.

▶ **Definition 27.** *Let* $\mathbf{A}$ *be a structure with universe* A*. Then denote by* $\mathbf{A}^*$ *the extension of* $\mathbf{A}$ *by a fresh unary relation symbol* C_a *interpreted as* $C_a^{\mathbf{A}} = \{a\}$ *for each* $a \in \mathrm{dom}(\mathbf{A})$*. Analogously, denote by* $\mathcal{A}^*$ *for a class of structures* $\mathcal{A}$ *the class* $\{\, \mathbf{A}^* \mid \mathbf{A} \in \mathcal{A} \,\}$*.*

Define p-#Hom($\mathcal{A}$) as the following problem. Given a pair of structures $(\mathbf{A}, \mathbf{B})$ where $\mathbf{A} \in \mathcal{A}$, and parameter k, output the number of homomorphisms from $\mathbf{A}$ to $\mathbf{B}$, if $|\mathrm{dom}(\mathbf{A})| \leq k$, and 0 otherwise.

Problem:	p-#Hom($\mathcal{A}$)
Input:	A pair of structures $(\mathbf{A}, \mathbf{B})$ where $\mathbf{A} \in \mathcal{A}$.
Parameter:	$\lvert\mathbf{A}\rvert$, $k \in \mathbb{N}$.
Output:	the number of homomorphisms from $\mathbf{A}$ to $\mathbf{B}$ if $\lvert\mathrm{dom}(\mathbf{A})\rvert \leq k$, 0 otherwise.

Notice that $\mathbf{B}$ can be any structure. For $n \geq 2$, let P_n be the canonical undirected path of length n, that is, the (E)-structure with universe $\{1, \ldots, n\}$ and $E^{P_n} = \{\, (i, i+1), (i+1, i) \mid 1 \leq i < n \,\}$. Let $\mathcal{P}$ be the class of structures isomorphic to some P_n. For the next theorem, reduce to p-#REACH for membership, and from a normalised, coloured variant of p-#REACH for hardness.

▶ **Theorem 28.** p-#Hom($\mathcal{P}^*$) *is* $\#\mathbf{para}_\beta\mathbf{L}$*-complete with respect to* $\leq^{\mathrm{plog}}_{\mathrm{pars}}$*.*

▶ **Open Problem 29.** *Is there a natural class of structures* $\mathcal{A}$ *such that* p-#Hom($\mathcal{A}$) *is* $\#\mathbf{para}_{\mathbf{W}[1]}\mathbf{L}$*-complete with respect to* $\leq^{\mathrm{plog}}_{\mathrm{pars}}$*?*

4.4 The Parameterised Complexity of the Determinant

In this section, we consider a parameterised variant of the determinant function introduced by Chauhan and Rao [13]. For $n > 0$ let $\mathcal{S}_n$ denote the set of all permutations of $\{1, \ldots, n\}$. For $k \leq n$, let $\mathcal{S}_{n,k}$ denote the following subset of $\mathcal{S}_n$: $\mathcal{S}_{n,k} = \{\, \pi \mid \pi \in S_n, |\{\, i : \pi(i) \neq i \,\}| = k \,\}$.

We define the parameterised determinant function of an $n \times n$ square matrix $A = (a_{i,j})_{1 \leq i,j \leq n}$ as $p\text{-}\mathrm{det}(A, k) = \sum_{\pi \in S_{n,k}} \mathrm{sign}(\pi) \prod_{i:\pi(i) \neq i} a_{i,\pi(i)}$.

Using an interpolation argument, it can be shown that p-det is in **FP** when k is part of the input and thereby in **FFPT** [13], the functional counterpart of **FPT**. In fact, the same interpolation argument can be used to show that p-det is in **GapL** (the class of functions $f(x)$ such that for some **NL**-machine, $f(x)$ is the number of accepting minus the number of rejecting paths). However, this does not give a space efficient algorithm for p-det in the sense of parameterised classes. The **GapL** algorithm may require a large number of nondeterministic steps and accordingly is not k-bounded. We show that the space efficient algorithm for the determinant given by Mahajan and Vinay [38] can be adapted to the parameterised setting, proving that p-det can be written as a difference of two $\#\mathbf{para}_{\beta}\mathbf{L}$ functions. Recall the notion of a *clow sequence* introduced by Mahajan and Vinay [38].

▶ **Definition 30** (Clow). *Let $\mathfrak{G} = (V, E)$ be a directed graph with $V = \{1, \ldots, n\}$ for some $n \in \mathbb{N}$. A* clow *in $\mathfrak{G}$ is a walk $C = (w_1, \ldots, w_{r-1}, w_r = w_1)$ where w_1 is the minimal vertex among $w_1, \ldots, w_{r-1}$ with respect to the natural ordering of V and $w_1 \neq w_j$ for all $1 < j < r$. Node w_1 is called the* head *of C, denoted by* $\mathrm{head}(C)$.

▶ **Definition 31** (Clow sequence). *A* clow sequence *of a graph $\mathfrak{G} = (\{1, \ldots, n\}, E)$ is a sequence $W = (C_1, \ldots, C_k)$ such that C_i is a clow of $\mathfrak{G}$ for $1 \leq i \leq k$ and*

- *the heads of the sequence are in ascending order* $\mathrm{head}(C_1) < \cdots < \mathrm{head}(C_k)$*, and*
- *the total number of edges that appear in some C_i (including multiplicities) is exactly n.*

For a clow sequence W of some graph $\mathfrak{G} = (\{1, \ldots, n\}, E)$ with r clows the *sign of* W, $\mathrm{sign}(W)$, is defined as $(-1)^{n+r}$. Note that, if the clow sequence is a cycle cover σ, then $(-1)^{n+r}$ is equal to the sign of the permutation represented by σ (that is, $(-1)^{\#\text{ inversions in }\sigma}$). Mahajan and Vinay came up with this sign-function to derive their formula for the determinant.

For an $(n \times n)$-matrix A, $\mathfrak{G}_A$ is the weighted directed graph with vertex set $\{1, \ldots, n\}$ and weighted adjacency matrix A. For a clow (sequence) W, $\mathrm{weight}(W)$ is the product of weights of the edges (clows) in w. For any $\mathfrak{G}$ as above, $\mathcal{W}_{\mathfrak{G}}$ is the set of all clow sequences of $\mathfrak{G}$. Mahajan and Vinay [38] proved that $\det(A) = \sum_{W \in \mathcal{W}_{\mathfrak{G}_A}} \mathrm{sign}(W) \cdot \mathrm{weight}(W)$.

We adapt these notions to the parameterised setting. First observe that for a permutation $\sigma \in S_{n,k}$, we have that $\mathrm{sign}(\sigma) = (-1)^{n+r}$, where r is the number of cycles in the permutation. However, the number of cycles in σ is $n - k + r'$, where r' is the number of cycles of length at least two in σ. Accordingly, we have $\mathrm{sign}(\sigma) = (-1)^{2n-k+r'}$. Adapting the definition of a clow sequence, for $k \geq 0$, define a *k-clow sequence* to be a clow sequence where the total number of edges (including multiplicity) in the sequence is exactly k, every clow has at least two edges, and no self loop edge of the form (i, i) occurs in any of the clows. For any graph $\mathfrak{G}$ with vertex set $\{1, \ldots, n\}$ for $n \in \mathbb{N}$, $\mathcal{W}_{\mathfrak{G},k}$ is the set of all k-clow sequences of $\mathfrak{G}$. For a k-clow sequence $W \in \mathcal{W}_{\mathfrak{G},k}$, $\mathrm{sign}(W)$ is $(-1)^{2n-k+r'}$, where r' is the number of clows in W. Mahajan and Vinay [38, Theorem 1] showed that the signed sum of the weights of all clow sequences is equal to the determinant. At the outset, this is a bit surprising, since the determinant is equal to the signed sum of weights of cycle covers, whereas there are clow sequences that are not cycle covers. Mahajan and Vinay [38] observed that every clow sequence that is not a cycle cover can be associated with a unique clow sequence of opposite sign, and thereby all clow sequences cancel out. We observe a parameterised version of the above result [38, Theorem 1].

▶ **Lemma 32.** $p\text{-}\det(A, k) = \sum_{W \in \mathcal{W}_{\mathfrak{G}_A,k}} \mathrm{sign}(W) \cdot \mathrm{weight}(W)$*, for $\{0, 1\}$-matrix A, $k \in \mathbb{N}$.*

Using this characterisation, the upper bound in the following theorem can be obtained. For hardness a reduction from p-#REACH suffices.

▶ **Theorem 33.** *The problem p-det for $(0, 1)$-matrices can be written as a difference of two functions in $\#\mathbf{para}_{\beta\mathbf{tail}}\mathbf{L}$, and is $\#\mathbf{para}_{\beta\mathbf{tail}}\mathbf{L}$-hard with respect to $\leq^{\mathrm{plog}}_{\mathrm{met}}$.*

5 Conclusions and Outlook

We developed foundations for the study of parameterised space complexity of counting problems. Our results show interesting characterisations for classes defined in terms of k-bounded para-logspace NTMs. We believe that our results will lead to further research of parameterised logspace counting complexity. Notice, that the studied walk problems in Section 4.1 can be considered restricted to DAGs yielding the same completeness results.

Branching programs are immanent for the study of space-bounded and parallel complexity classes. Languages accepted by polynomial-size logspace-uniform branching programs characterise **NL**. In fact, this result carries forward to the counting versions. Motivated by this, one can consider parameterised counting classes based on deterministic branching programs (DBPs) and nondeterministic branching programs (BPs). It can be shown that for any $o \in \{\mathbf{W}, \mathbf{W}[1], \beta, \beta\text{-}\mathbf{tail}\}$, $\#\mathbf{para}_o\text{-}\mathbf{L}$ and $\#\mathbf{para}_o\text{-}\mathbf{NL}$, can be characterised in terms of an adequate parameterised counting version of DBPs and BPs, respectively (see the technical report [36]).

Comparing our newly introduced classes with the **W**-hierarchy (which is defined in terms of weighted satisfiability problems for circuits of a so-called bounded weft), one might ponder whether there is an alternative definition of our classes with such circuit problems. Though in this article we did not explore the weighted satisfiability, the closely related problem $p\text{-MC}(\Sigma_0)$ sheds some light on this. Theorem 23 shows that $p\text{-MC}(\Sigma_0)$ is complete for $\mathbf{para}_{\mathbf{W}[1]}\mathbf{L}$ (in fact, we show this for their counting versions) under $\leq_m^{\text{plog}}$-reductions. However, if we take **FPT**-reductions, $p\text{-MC}(\Sigma_0)$ is complete for $\mathbf{W}[1]$. Though we could not prove it so far, we believe this is a general phenomenon: Any $\mathbf{W}[1]$-complete problem is complete for $\mathbf{para}_{\mathbf{W}[1]}\mathbf{L}$ under $\leq_m^{\text{plog}}$-reductions. More generally, there is a possibility that the FPT-closure of $\mathbf{para}_W\mathbf{L}$-classes is equal to the corresponding class in the **W**-hierarchy.

One might also ask the question if $\mathbf{para}_W\mathbf{L}$ is contained in **FFPT**. This is unlikely based on the view expressed above. For example, $p\text{-MC}(\Sigma_0)$ is complete for both $\mathbf{para}_{\mathbf{W}[1]}\mathbf{L}$ and $\mathbf{W}[1]$ but under two different reductions. As a result, $\mathbf{para}_W\mathbf{L} \subseteq \mathbf{FFPT}$ would imply that $p\text{-MC}(\Sigma_0) \in \mathbf{FPT}$ and, accordingly, $\mathbf{FPT} = \mathbf{W}[1]$ as **FPT** is closed under **FPT**-reductions. We close with interesting tasks for further research:

- Study further closure properties of the new classes (e.g., Open Problem 13).
- Improve the understanding of the influence of syntactic locality, resp., bounded arity in the setting of p-#MC(Σ_0) (Open Problem 24).
- Find a characterisation of the $\leq_{\text{pars}}^{\text{plog}}$-closure of $\#\mathbf{para}_{\mathbf{W}[1]}\mathbf{L}$ (Open Problem 25).
- Identify a natural class of structures for which the homomorphism problem is $\#\mathbf{para}_{\mathbf{W}}\mathbf{L}$-complete (Open Problem 29).
- Establish a broader spectrum of complete problems for the classes $\mathbf{para}_\beta\mathbf{L}$ and $\mathbf{para}_W\mathbf{L}$, e.g., in the realm of satisfiability questions.
- Identify further characterisations of the introduced classes, e.g., in the vein of descriptive complexity, which could highlight their robustness.
- Study gap classes [29] based on our classes. This might help improve Theorem 33.

References

1 Manindra Agrawal, Eric Allender, and Samir Datta. On TC^0, AC^0, and arithmetic circuits. *J. Comput. Syst. Sci.*, 60(2):395–421, 2000.

2 Eric Allender, Robert Beals, and Mitsunori Ogihara. The complexity of matrix rank and feasible systems of linear equations. *Computational Complexity*, 8(2):99–126, 1999.

3 Eric Allender and Mitsunori Ogihara. Relationships among PL, #L, and the determinant. *ITA*, 30(1):1–21, 1996.

4 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

5 Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

6 Max Bannach, Christoph Stockhusen, and Till Tantau. Fast parallel fixed-parameter algorithms via color coding. In *IPEC*, volume 43 of *LIPIcs*, pages 224–235. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015.

7 Richard Beigel and Bin Fu. Circuits over PP and PL. *J. Comput. Syst. Sci.*, 60(2):422–441, 2000.

8 Ralph Bottesch. Relativization and interactive proof systems in parameterized complexity theory. In *IPEC*, volume 89 of *LIPIcs*, pages 9:1–9:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

9 Ralph Christian Bottesch. On W[1]-hardness as evidence for intractability. In *MFCS*, volume 117 of *LIPIcs*, pages 73:1–73:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

10 Cornelius Brand and Marc Roth. Parameterized counting of trees, forests and matroid bases. In *CSR*, volume 10304 of *Lecture Notes in Computer Science*, pages 85–98. Springer, 2017.

11 Jonathan F. Buss. Relativized alternation and space-bounded computation. *J. Comput. Syst. Sci.*, 36(3):351–378, 1988.

12 Hervé Caussinus, Pierre McKenzie, Denis Thérien, and Heribert Vollmer. Nondeterministic NC^1 computation. *J. Comput. Syst. Sci.*, 57(2):200–212, 1998.

13 Ankit Chauhan and B. V. Raghavendra Rao. Parameterized analogues of probabilistic computation. In *CALDAM*, volume 8959 of *Lecture Notes in Computer Science*, pages 181–192. Springer, 2015.

14 Hubie Chen and Moritz Müller. The fine classification of conjunctive queries and parameterized logarithmic space. *TOCT*, 7(2):7:1–7:27, 2015.

15 Yijia Chen and Jörg Flum. Some lower bounds in parameterized AC^0. In *MFCS*, volume 58 of *LIPIcs*, pages 27:1–27:14. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016.

16 Yijia Chen and Jörg Flum. Some lower bounds in parameterized AC^0. *CoRR*, abs/1606.08014, 2016. `arXiv:1606.08014`.

17 Yijia Chen, Jörg Flum, and Martin Grohe. Bounded nondeterminism and alternation in parameterized complexity theory. In *Computational Complexity Conference*, pages 13–29. IEEE Computer Society, 2003.

18 Radu Curticapean. Counting matchings of size k is W[1]-hard. In *ICALP (1)*, volume 7965 of *Lecture Notes in Computer Science*, pages 352–363. Springer, 2013.

19 Radu Curticapean. *The simple, little and slow things count: on parameterized counting complexity*. PhD thesis, Saarland University, 2015.

20 Radu Curticapean. Block interpolation: A framework for tight exponential-time counting complexity. *Inf. Comput.*, 261(Part):265–280, 2018.

21 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *STOC*, pages 210–223. ACM, 2017.

22 Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theor. Comput. Sci.*, 329(1-3):315–323, 2004.

23 Carsten Damm. DET = $L^{\#L}$? Technical report, Informatik-Preprint 8, Fachbereich Informatik der Humboldt-Universitlt zu Berlin, 1991.

24 Samir Datta, Meena Mahajan, B. V. Raghavendra Rao, Michael Thomas, and Heribert Vollmer. Counting classes and the fine structure between NC^1 and L. *Theor. Comput. Sci.*, 417:36–49, 2012.

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

26 Martin E. Dyer and Catherine S. Greenhill. The complexity of counting graph homomorphisms. *Random Struct. Algorithms*, 17(3-4):260–289, 2000.

27 Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. *Mathematical logic (2. ed.)*. Undergraduate texts in mathematics. Springer, 1994.

28 Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the space and circuit complexity of parameterized problems: Classes and completeness. *Algorithmica*, 71(3):661–701, 2015.

29 Stephen A. Fenner, Lance Fortnow, and Stuart A. Kurtz. Gap-definable counting classes. *J. Comput. Syst. Sci.*, 48(1):116–148, 1994.

30 Jörg Flum, Markus Frick, and Martin Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6):716–752, 2002.

31 Jörg Flum and Martin Grohe. Describing parameterized complexity classes. *Inf. Comput.*, 187(2):291–319, 2003.

32 Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM J. Comput.*, 33(4):892–922, 2004.

33 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

34 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1):1:1–1:24, 2007.

35 Anselm Haak, Juha Kontinen, Fabian Müller, Heribert Vollmer, and Fan Yang. Counting of teams in first-order team logics. In *MFCS*, volume 138 of *LIPIcs*, pages 19:1–19:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

36 Anselm Haak, Arne Meier, Om Prakash, and B. V. Raghavendra Rao. Parameterised counting in logspace. *CoRR*, abs/1904.12156, 2019. `arXiv:1904.12156`.

37 Mark Jerrum and Kitty Meeks. The parameterised complexity of counting even and odd induced subgraphs. *Combinatorica*, 37(5):965–990, 2017.

38 Meena Mahajan and V. Vinay. Determinant: Combinatorics, algorithms, and complexity. *Chicago J. Theor. Comput. Sci.*, 1997, 1997.

39 Catherine McCartin. Parameterized counting problems. In *MFCS*, volume 2420 of *Lecture Notes in Computer Science*, pages 556–567. Springer, 2002.

40 Mitsunori Ogihara. The PL hierarchy collapses. *SIAM J. Comput.*, 27(5):1430–1437, 1998.

41 Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

42 Christoph Stockhusen. *On the space and circuit complexity of parameterized problems*. PhD thesis, University of Lübeck, Germany, 2017. URL: `http://www.zhb.uni-luebeck.de/epubs/ediss1780.pdf`.

43 Christoph Stockhusen and Till Tantau. Completeness results for parameterized space classes. In *IPEC*, volume 8246 of *Lecture Notes in Computer Science*, pages 335–347. Springer, 2013.

44 Seinosuke Toda. Counting problems computationally equivalent to the determinant. Technical Report CSIM 91-07, Dept. Comp. Sci. and Inf. Math., Univ. of Electro-Communications, Tokyo, 1991. URL: `http://perso.ens-lyon.fr/natacha.portier/papers/toda91.pdf`.

45 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

46 Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

47 V. Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In *Proceedings of 6th Structure in Complexity Theory Conference*, pages 270–284, 1991.

Digraph Coloring and Distance to Acyclicity

Ararat Harutyunyan

Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243, LAMSADE, Paris, France

Michael Lampis

Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243, LAMSADE, Paris, France

Nikolaos Melissinos

Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243, LAMSADE, Paris, France

Abstract

In k-Digraph Coloring we are given a digraph and are asked to partition its vertices into at most k sets, so that each set induces a DAG. This well-known problem is NP-hard, as it generalizes (undirected) k-Coloring, but becomes trivial if the input digraph is acyclic. This poses the natural parameterized complexity question of what happens when the input is "almost" acyclic. In this paper we study this question using parameters that measure the input's distance to acyclicity in either the directed or the undirected sense.

In the directed sense perhaps the most natural notion of distance to acyclicity is directed feedback vertex set (DFVS). It is already known that, for all $k \geq 2$, k-Digraph Coloring is NP-hard on digraphs of DFVS at most $k+4$. We strengthen this result to show that, for all $k \geq 2$, k-Digraph Coloring is already NP-hard for DFVS exactly k. This immediately provides a dichotomy, as k-Digraph Coloring is trivial if DFVS is at most $k-1$. Refining our reduction we obtain two further consequences: (i) for all $k \geq 2$, k-Digraph Coloring is NP-hard for graphs of feedback *arc* set (FAS) at most k^2; interestingly, this leads to a second dichotomy, as we show that the problem is FPT by k if FAS is at most k^2-1; (ii) k-Digraph Coloring is NP-hard for graphs of DFVS k, even if the maximum degree Δ is at most $4k-1$; we show that this is also *almost* tight, as the problem becomes FPT for DFVS k and $\Delta \leq 4k-3$.

Since these results imply that the problem is also NP-hard on graphs of bounded directed treewidth, we then consider parameters that measure the distance from acyclicity of the underlying graph. On the positive side, we show that k-Digraph Coloring admits an FPT algorithm parameterized by treewidth, whose parameter dependence is $(\mathrm{tw}!)k^{\mathrm{tw}}$. Since this is considerably worse than the k^{tw} dependence of (undirected) k-Coloring, we pose the question of whether the tw! factor can be eliminated. Our main contribution in this part is to settle this question in the negative and show that our algorithm is essentially optimal, even for the much more restricted parameter treedepth and for $k=2$. Specifically, we show that an FPT algorithm solving 2-Digraph Coloring with dependence $\mathrm{td}^{o(\mathrm{td})}$ would contradict the ETH.

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

Keywords and phrases Digraph Coloring, Dichromatic number, NP-completeness, Parameterized complexity, Feedback vertex and arc sets

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.41

Related Version *Full Version*: `https://arxiv.org/abs/2010.06317`

Funding Michael Lampis was partially supported by a grant from the French National Research Agency under the JCJC program (ASSK: ANR-18-CE40-0025-01).

1 Introduction

In Digraph Coloring, we are given a digraph D and are asked to calculate the smallest k such that the vertices of D can be partitioned into k *acyclic* sets. In other words, the objective of this problem is to color the vertices with the minimum number of colors so that no directed

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 41; pp. 41:1–41:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

cycle is monochromatic. The notion of dichromatic number was introduced by V. Neumann-Lara [37]. More recently, digraph coloring has received much attention, in part because it turns out that many results about the chromatic number of undirected graphs quite naturally carry over to the dichromatic number of digraphs [1, 2, 4, 7, 11, 20, 21, 22, 23, 24, 32, 34, 35, 38]. We note that Digraph Coloring generalizes Coloring (if we simply replace all edges of a graph by pairs of anti-parallel arcs) and is therefore NP-complete.

In this paper we are interested in the computational complexity of Digraph Coloring from the point of view of structural parameterized complexity[1]. Our main motivation for studying this is that (undirected) Coloring is a problem of central importance in this area whose complexity is well-understood, and it is natural to hope that some of the known tractability results may carry over to digraphs – especially because, as we mentioned, Digraph Coloring seems to behave as a very close counterpart to Coloring in many respects. In particular, for undirected graphs, the complexity of Coloring for "almost-acyclic" graphs is very precisely known: for all $k \geq 3$ there is a $O^*(k^{\mathrm{tw}})$ algorithm, where tw is the input graph's treewidth, and this is optimal (under the SETH) even if we replace treewidth by much more restrictive parameters [27, 33]. Can we achieve the same amount of precision for Digraph Coloring?

Our results. The main question motivating this paper is therefore the following: Does Digraph Coloring also become tractable for "almost-acyclic" inputs? We attack this question from two directions.

First, we consider the notion of acyclicity in the digraph sense and study cases where the input digraph is close to being a DAG. Possibly the most natural such measure is directed feedback vertex set (DFVS), which is the minimum number of vertices whose removal destroys all directed cycles. The problem is paraNP-hard for this parameter, as for all fixed $k \geq 2$, k-Digraph Coloring is already known to be NP-hard, for inputs of DFVS at most $k+4$ [34]. Our first contribution is to tighten this result by showing that actually k-Digraph Coloring is already NP-hard for DFVS of size *exactly* k. This closes the gap left by the reduction of [34] and provides a complete dichotomy, as the problem is trivially FPT by k when the DFVS has size strictly smaller than k (the only non-trivial part of the problem in this case is to find the DFVS [10]).

This negative result motivates us to either consider a more restricted notion of near-acyclicity, or to impose further restrictions, such as bounding the maximum degree of the graph. Unfortunately, we show that neither of these suffices to make the problem tractable. In particular, by refining our reduction we obtain the following: First, we show that for all $k \geq 2$, k-Digraph Coloring is NP-hard for digraphs of feedback *arc* set (FAS) k^2, that is, digraphs where there exists a set of k^2 arcs whose removal destroys all cycles (feedback arc set is of course a more restrictive parameter than feedback vertex set). Interestingly, this also leads us to a complete dichotomy, this time for the parameter FAS: we show that k-coloring becomes FPT (by k) on graphs of FAS at most k^2-1, by an argument that reduces this problem to coloring a subdigraph with at most $O(k^2)$ vertices, and hence the correct complexity threshold for this parameter is k^2. Second, we show that k-coloring a digraph with DFVS k remains NP-hard even if the maximum degree is at most $4k-1$. This further strengthens the reduction of [34], which showed that the problem is NP-hard for bounded *degeneracy* (rather than degree). Almost completing the picture, we show that k-coloring a digraph with DFVS k and maximum degree at most $4k-3$ is FPT by k, leaving open only the case where the DFVS is exactly k and the maximum degree exactly $4k-2$.

[1] In the remainder, we assume the reader is familiar with the basics of parameterized complexity theory, such as the class FPT, as given in standard textbooks [12].

The results we obtain for DFVS and FAS are mostly negative, but one could argue that this is because directed acyclicity allows a much richer class of inputs than undirected acyclicity and hence it is unreasonable to expect Digraph Coloring with these parameters to be as tractable as Coloring for treewidth. Therefore, in the second part of the paper we make a more "fair" comparison and parameterize the problem by the treewidth of the underlying graph. It turns out that, finally, this suffices to lead to an FPT algorithm, obtained with standard DP techniques. However, our algorithm has a somewhat disappointing running time of $(\mathrm{tw}!)k^{\mathrm{tw}}n^{O(1)}$, which is significantly worse than the $k^{\mathrm{tw}}n^{O(1)}$ complexity which is known to be optimal for undirected Coloring, especially for small values of k. This raises the question of whether the extra (tw!) factor can be removed. Our main contribution in this part is to show that this is likely impossible, even for a more restricted case. Specifically, we show that if the ETH is true, no algorithm can solve 2-Digraph Coloring in time $td^{o(\mathrm{td})}n^{O(1)}$, where td is the input graph's treedepth, a parameter more restrictive than treewidth (and pathwidth). As a result, this paper makes a counterpoint to the line of research that seeks to find ways in which dichromatic number replicates the behavior of chromatic number in the realm of digraphs by pinpointing one important aspect where the two notions are quite different, namely their complexity with respect to treewidth.

Other related work. Structural parameterizations of Digraph Coloring have been studied in [38], who showed that the problem is FPT by modular width generalizing the algorithms of [18, 29]; and [20] who showed that the problem is in XP by clique-width (note that hardness results for Coloring rule out an FPT algorithm in this case [16, 17, 30]). Our results on the hardness of the problem for bounded DFVS and FAS build upon the work of [34]. The fact that the problem is hard for bounded DFVS implies that it is also hard for most versions of directed treewidth, including DAG-width, Kelly-width, and directed pathwidth [6, 19, 25, 28, 31]. Indeed, hardness for FAS implies also hardness for bounded elimination width, a more recently introduced restriction of directed treewidth [15]. For undirected treewidth, a problem with similar behavior is DFVS: (undirected) FVS is solvable in $O^*(3^{\mathrm{tw}})$ [13] but DFVS cannot be solved in time $\mathrm{tw}^{o(\mathrm{tw})}n^{O(1)}$, and this is tight under the ETH [8]. For other natural problems whose complexity by treewidth is $tw^{\Theta(\mathrm{tw})}$ see [3, 5, 9]

With respect to maximum degree, it is not hard to see that k-Digraph Coloring is NP-hard for graphs of maximum degree $2k+2$, because k-Coloring is NP-hard for graphs of maximum degree $k+1$, for all $k \geq 3$ [2]. On the converse side, using a generalization of Brooks' theorem due to Mohar [36] one can see that k-Digraph Coloring digraphs of maximum degree $2k$ is in P. This leaves as the only open case digraphs of degree $2k+1$, which in a sense mirrors our results for digraphs of DFVS k and degree $4k-2$. We note that the NP-hardness of 2-Digraph Coloring for bounded degree graphs is known even for graphs of large girth, but the degree bound follows the imposed bound on the girth [14].

2 Definitions and Notation

We use standard graph-theoretic notation. All digraphs are loopless and have no parallel arcs; two oppositely oriented arcs between the same pair of vertices, however, are allowed and are called a *digon*. The in-degree (respectively, out-degree) of a vertex is the number of

[2] Note that this argument does not prove that 2-Digraph Coloring is NP-hard for maximum degree 6, but this is not too hard to show. We give a proof in the full version of the paper for the sake of completeness.

arcs coming into (respectively going out of) a vertex. The degree of a vertex is the sum of its in-degree and out-degree. For a set of arcs F, $V(F)$ denotes the set of their endpoints. For a set of vertices S of a digraph D, $D[S]$ denotes the digraph induced by S and $N[S]$ denotes the closed neighborhood of S, that is, S and all vertices that have an arc to or from S.

The *chromatic number* of a graph G is the minimum number of colors k needed to color the vertices of G such that each color class is an independent set. We say that a digraph $D = (V, E)$ is *k-colorable* if we can color the vertices of D with k colors such that each color class induces an acyclic subdigraph (such a coloring is called a *proper k-coloring*). The *dichromatic number*, denoted by $\vec{\chi}(D)$, is the minimum number k for which D is k-colorable.

A subset of vertices $S \subset V$ of D is called a *feedback vertex set* if $D - S$ is acyclic. A subset of arcs $A \subset E$ of D is called a *feedback arc set* if $D - A$ is acyclic. For the definition of treewidth and nice tree decompositions we refer the reader to [12]. A graph G has treedepth at most k if one of the following holds: (i) G has at most k vertices (ii) G is disconnected and all its components have treedepth at most k (iii) there exists $u \in V(G)$ such that $G - u$ has treedepth at most $k - 1$. We use $\mathrm{tw}(G), \mathrm{td}(G)$ to denote the treewidth and treedepth of a graph. It is known that $\mathrm{tw}(G) \leq \mathrm{td}(G)$ for all graphs G.

The Exponential Time Hypothesis (ETH) [26] states that there is a constant $c > 1$ such that no algorithm which decides if 3-SAT formulas with n variables and m clauses are satisfiable can run in time c^{n+m}. In this paper we will use the simpler (and slightly weaker) version of the ETH which simply states that 3-SAT cannot be solved in time $2^{o(n+m)}$.

Throughout the paper, when n is a positive integer we use $[n]$ to denote the set $\{1, \ldots, n\}$. For a set V an ordering of V is an injective function $\sigma : V \to [|V|]$. It is a well-known fact that a digraph D is acyclic if and only if there exists an ordering σ of $V(D)$ such that for all arcs uv we have $\sigma(u) < \sigma(v)$. This is called a topological ordering of D.

3 Bounded Feedback Vertex Set

In this section we study the complexity of the problem parameterized by the size of the feedback vertex set of a digraph. Throughout we will assume that a feedback vertex set is given to us; if not we can use known FPT algorithms to find the smallest such set [10].

Our main result in this section is that k-Digraph Coloring is NP-hard for graphs of DFVS k. We begin with an easy observation showing that this result will be best possible.

▶ Remark 1. Every digraph $D = (V, E)$ with feedback vertex set of size at most $k - 1$ is k-colorable.

The remark holds because we can use distinct colors for the vertices of the feedback vertex set and the remaining color for the rest of the graph. Building on this we can make a further easy remark.

▶ Remark 2. Let $D = (V, E)$ be a digraph with feedback vertex set F of size $|F| = k$. If F does not induce a bi-directed clique, then D is k-colorable.

Indeed, if $u, v \in F$ are not connected by a digon we can use one color for $\{u, v\}$, $k - 2$ distinct colors for the rest of F, and the remaining color for the rest of the graph. Remark 2 will also be useful later in designing an algorithm, but at this point it is interesting because it tells us that, since the graphs we construct in our reduction have DFVS k and must in some cases have $\vec{\chi}(D) > k$, our reduction needs to construct a bi-directed clique of size k.

Before we go on to our reduction let us also mention that we will reduce from a restricted version of 3-SAT with the following properties: (i) all clauses must have either only positive literals or only negative literals (ii) all variables appear at most 2 times positive and 1 time negative. We call this Restricted-3-SAT.

▶ **Lemma 3.** *Restricted-3-SAT is NP-hard and cannot be solved in $2^{o(n+m)}$ time unless the ETH is false.*

The proof for the Lemma 3 is deferred to the full version of the paper.

▶ **Theorem 4.** *For all $k \geq 2$, it is NP-hard to decide if a digraph $D = (V, E)$ is k-colorable even when the size of its feedback vertex set is k. Furthermore, this problem cannot be solved in time $2^{o(n)}$ unless the ETH is false.*

Proof. We will prove the theorem for $k = 2$. To obtain the proof for larger values one can add to the construction $k - 2$ vertices which are connected to everything with digons: this increases both the dichromatic number and the feedback vertex set by $k - 2$. Note that this does indeed construct a "palette" clique of size k, as indicated by Remark 2.

We make a reduction from Restricted-3-SAT, which is NP-hard by Lemma 3. Our reduction will produce an instance of size linear in the input formula, which leads to the ETH-based lower bound. Let ϕ be the given formula with variables $x_1, \ldots, x_n$, and suppose that clauses $c_1, \ldots, c_\ell$ contain only positive literals, while clauses $c_{\ell+1}, \ldots, c_m$ contain only negative literals. We will assume without loss of generality that all variables appear in ϕ both positive and negative (otherwise ϕ can be simplified).

We begin by constructing two "palette" vertices v_1, v_2 which are connected by a digon. Then, for each clause $c_i, i \in [m]$ we do the following: if the clause has size three we construct a directed path with vertices $l_{i,1}, w_{i,1}, l_{i,2}, w_{i,2}, l_{i,3}$, where the vertices $l_{i,1}, l_{i,2}, l_{i,3}$ represent the literals of the clause; if the clause has size two we similarly construct a directed path with vertices $l_{i,1}, w_{i,1}, l_{i,2}$, where again $l_{i,1}, l_{i,2}$ represent the literals of the clause.

For each variable $x_j, j \in [n]$ we do the following: for each clause c_{i_1} where x_j appears positive and clause c_{i_2} where x_j appears negative we construct a vertex w'_{j,i_1,i_2} and add an incoming arc from the vertex that represents the literal x_j in the directed path of c_{i_1} to w'_{j,i_1,i_2}; and an outgoing arc from w'_{j,i_1,i_2} to the vertex that represents the literal $\neg x_j$ in the directed path of c_{i_2}.

Finally, to complete the construction we connect the palette vertices to the rest of the graph as follows: v_1 is connected with a digon to all existing vertices $w_{i,j}$, $i \in [m], j \in [2]$; v_2 is connected with a digon to all existing vertices w'_{j,i_1,i_2}; v_2 has an outgoing arc to the first vertex of each directed path representing a clause and an incoming arc from the last vertex of each such path; v_1 has an outgoing arc to all vertices that represent positive literals and an incoming arc from all vertices representing negative literals. (See Figure 1)

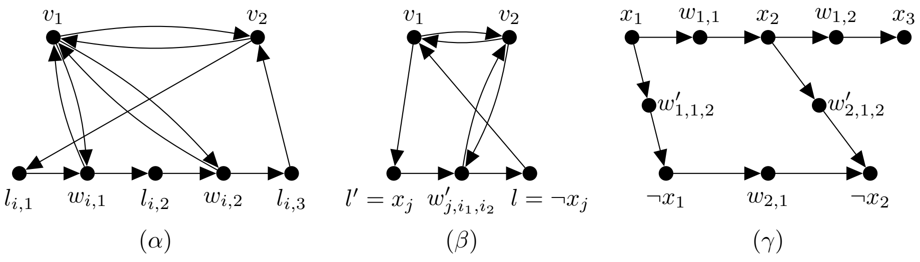


Figure 1 (α): The cycles created by $\{v_1, v_2\}$ and clauses with three literals. (β): The cycles created by $\{v_1, v_2\}$ and each pair $\{x, \neg x\}$. (γ): An example digraph for the formula $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2)$, without showing v_1, v_2.

Let us now prove that this reduction implies the theorem. First, we claim that in the digraph we constructed $\{v_1, v_2\}$ is a feedback vertex set. Indeed, suppose we remove these two vertices. Now every arc in the remaining graph either connects vertices that represent the same clause, or is incident on a vertex w'_{j,i_1,i_2}. Observe that these vertices have only one incoming and one outgoing arc and because of the ordering of the clauses $i_1 < i_2$ (since clauses that contain negative literals come later in the numbering). We conclude that every directed path must either stay inside the path representing the same clause or lead to a path the represents a later clause. Hence, the digraph is acyclic.

Let us now argue that if ϕ is satisfiable then the digraph is 2-colorable. We give color 1 to v_1 and 2 to v_2. We give color 2 to each $w_{i,j}$ and color 1 to each w'_{j,i_1,i_2}. Fix a satisfying assignment for ϕ. We give color 1 to all vertices $l_{i,j}$ that represent literals set to True by the assignment and color 2 to all remaining vertices. Let us see why this coloring is acyclic. First, consider a vertex w'_{j,i_1,i_2}. This vertex has color 1 and one incoming and one outgoing arc corresponding to opposite literals. Because the literals are opposite, one of them has color 2, hence w'_{j,i_1,i_2} cannot be in any monochromatic cycle and can be removed. Now, suppose there is a monochromatic cycle of color 1. As $\{v_1, v_2\}$ is a feedback vertex set, this cycle must include v_1. Since v_2 and all $w_{i,j}$ have color 2 the vertex after v_1 in the cycle must be some $l_{i,j}$ representing a positive literal which was set to True by our assignment. The only outgoing arc leaving from $l_{i,j}$ and going to a vertex of color 1 must lead it to a vertex $w'_{j',i,i'}$, which as we said cannot be part of any cycle. Hence, no monochromatic cycle of color 1 exists. Consider then a monochromatic cycle of color 2, which must begin from v_2. The next vertex on this cycle must be a $l_{i,1}$ and since we have eliminated vertices w'_{j,i_1,i_2} the cycle must continue in the directed path of clause i. But, since we started with a satisfying assignment, at least one of the literal vertices of this path has color 1, meaning the cycle cannot be monochromatic.

Finally, let us argue that if the digraph is 2-colorable, then ϕ is satisfiable. Consider a 2-coloring which, without loss of generality, assigns 1 to v_1 and 2 to v_2. The coloring must give color 2 to all $w_{i,j}$ and color 1 to all w_{j,i_1,i_2}, because of the digons connecting these vertices to the palette. Now, we obtain an assignment for ϕ as follows: for each x_j, we find the vertex in our graph that represents the literal $\neg x_j$ (this is unique since each variable appears exactly once negatively): we assign x_j to True if and only if this vertex has color 2. Let us argue that this assignment satisfies all clauses. First, consider a clause with all negative literals. If this clause is not satisfied, then all the vertices representing its literals have color 2. Because vertices $w_{i,j}$ also all have color 2, this creates a monochromatic cycle with v_2, contradiction. Hence, all such clauses are satisfied. Second, consider a clause c_i with all positive literals. In the directed path representing c_i at least one literal vertex must have color 1, otherwise we would get a monochromatic cycle with v_2. Suppose this vertex represents the literal x_j and has an out-neighbor w'_{j,i,i_2}, which is colored 1. If the out-neighbor of w'_{j,i_1,i_2} is also colored 1, we get a monochromatic cycle with v_1. Therefore, that vertex, which represents the literal $\neg x_j$ has color 2. But then, according to our assignment x_j is True and c_i is satisfied. ◀

4 Bounded Feedback Arc Set and Bounded Degree

In this section we first present two algorithmic results: we show that k-Digraph Coloring becomes FPT (by k) if either the input graph has feedback vertex set k and maximum degree at most $4k-3$; or if it has feedback arc set at most k^2-1 (and unbounded degree). Interestingly, the latter of these results is exactly tight and the former is almost tight: in the second part we refine the reduction of the previous section to show that k-Digraph Coloring is NP-hard for digraphs which have simlutaneously a FAS of size k^2, a feedback vertex set of size k and maximum degree $\Delta = 4k-1$.

4.1 Algorithmic Results

Our first result shows that for k-Digraph Coloring, if we are promised a feedback vertex set of size k (which is the smallest value for which the problem is non-trivial), then the problem remains tractable for degree up to $4k-3$. Observe that in the case of general digraphs (where we do not bound the feedback vertex set) the problem is already hard for maximum degree $2k+2$ (see Other Related Work section), so this seems encouraging. However, we show in Theorem 8 that this tractability cannot be extended much further.

▶ **Theorem 5.** *Let $D=(V,E)$ be a digraph with feedback vertex set F of size $|F|=k$ and maximum degree $\Delta \leq 4k-3$. Then, D is k-colorable if and only if $D[N[F]]$ is k-colorable. Furthermore, a k-coloring of $D[N[F]]$ can be extended to a k-coloring of D in polynomial time.*

Proof. Let $D=(V,E)$ be such a digraph. If $D[N[F]]$ is not k-colorable, then D is not k-colorable, so we need to prove that if $D[N[F]]$ is k-colorable then D is k-colorable and we can extend this coloring to D. Assume that $D[N[F]]$ is k-colorable. By Remark 2 we can assume that $D[F]$ is a bi-directed clique. Let $c: N[F] \to [k]$ be the assumed k-coloring and without loss of generality say that $F=\{v_1,\dots,v_k\}$ and $c(v_i)=i$ for all $i \in [k]$.

Before we continue let us define the following sets of vertices: we will call $V_{i,in}$ the set of vertices $v \in N[F]\setminus F$ such that $c(v)=i$ and there exists an arc $vv_i \in E$. Similarly we will call $V_{i,out}$ the set of vertices $v \in N[F]\setminus F$ where $c(v)=i$ and there exists an arc $v_iv \in E$. The sets $V_{i,in}$ and $V_{i,out}$ are disjoint in any proper coloring (otherwise we would have a monochromatic digon). Furthermore, $V_{i,in} \cup V_{i,out}$ is disjoint from $V_{j,in} \cup V_{j,out}$ for $j \neq i$ (because their vertices have different colors), so all these $2k$ sets are pair-wise disjoint. We first show that if one of these $2k$ sets is empty, then we can color D.

▷ Claim 6. If for some $i \in [k]$ one of the sets $V_{i,in}$, $V_{i,out}$ is empty then we can extend c to a k-coloring of D in polynomial time.

Proof. We keep c unchanged and color all of $V(D)\setminus N[F]$ with color i. This is a proper k-coloring. Indeed, this cannot create a monochromatic cycle with color $j \neq i$. Furthermore, if a monochromatic cycle of color i exists, since this cycle must intersect F, we conclude that it must contain v_i. However, in the current k-coloring v_i either has no incoming neighbor or no outgoing neighbor colored i, so no monochromatic cycle can go through it. ◁

In the remainder we assume that all sets $V_{i,in}, V_{i,out}$ are non-empty. Our strategy will be to edit the k-coloring of $D[N[F]]$ so that we retain a proper k-coloring, but one of these $2k$ sets becomes empty. We will then invoke Claim 6 to complete the proof.

We now define, for each pair $i,j \in [k]$ with $i<j$ the set $E_{i,j}$ which contains all arcs with one endpoint in $\{v_i,v_j\}$ and the other in $V_{i,in}\cup V_{i,out}\cup V_{j,in}\cup V_{j,out}$ and whose endpoints have *distinct* colors. We call $E_{i,j}$ the set of *cross* arcs for the pair (i,j). We will now argue that for some pair (i,j) we must have $|E_{i,j}| \leq 3$. For the sake of contradiction, assume that $|E_{i,j}| \geq 4$ for all pairs. Then, by summing up the degrees of vertices of F we have:

$$\sum_{i\in[k]} d(v_i) \geq 2k + k(2k-2) + \sum_{i,j\in[k],i<j} |E_{i,j}| \geq 2k^2 + 4\binom{k}{2} = 4k^2-2k$$

In the first inequality we used the fact that each $v_i \in F$ has at least two arcs connecting it to $V_{i,in}\cup V_{i,out}$ (since these sets are non-empty); $2k-2$ arcs connecting it to other vertices of F (since F is a clique); and each cross arc of a set $E_{i,j}$ contributes one to the degree of one vertex of F. From this calculation we infer that the average degree of F is at least $4k-2$, which is a contradiction, since we assumed that the digraph has maximum degre $4k-3$.

Fix now i, j such that $|E_{i,j}| \leq 3$. We will recolor $V_{i,in} \cup V_{i,out} \cup V_{j,in} \cup V_{j,out}$ in a way that allows us to invoke Claim 6. Since we do not change any other color, we will only need to prove that our recoloring does not create monochromatic cycles of colors i or j in $D[N[F]]$. We can assume that $|E_{i,j}| = 3$, since if $|E_{i,j}| < 3$ we can add an arbitrary missing cross arc and this can only make the recoloring process harder. Furthermore, without loss of generality, we assume that v_i has strictly more cross arcs of $E_{i,j}$ incident to it than v_j.

We now have to make a case analysis. First, suppose all three arcs of $E_{i,j}$ are incident on v_i. Then, there exists a set among $V_{j,in}, V_{j,out}$ that has at most one arc connecting it to v_i. We color this set i, and leave the other set colored j. We also color $V_{i,in} \cup V_{i,out}$ with j. This creates no monochromatic cycle because: (i) v_i now has at most one neighbor colored i in $V_{i,in} \cup V_{i,out} \cup V_{j,in} \cup V_{j,out}$, so no monochromatic cycle goes through v_i; (ii) v_j has either no out-neighbors or no in-neighbors colored j in $V_{i,in} \cup V_{i,out} \cup V_{j,in} \cup V_{j,out}$. With the new coloring we can invoke Claim 6. In the remainder we therefore assume that two arcs of $E_{i,j}$ are incident on v_i and one is incident on v_j.

Second, suppose that one of $V_{j,in}, V_{j,out}$ has no arcs connecting it to v_i. We color this set i and leave the other set colored j. Observe that one of $V_{i,in}, V_{i,out}$ has no arc connecting it to v_j. We color that set j and leave the other set colored i. In the new coloring both v_i, v_j either have no out-neighbor or no in-neighbor with the same color in $V_{i,in} \cup V_{i,out} \cup V_{j,in} \cup V_{j,out}$, so the coloring is proper and we can invoke Claim 6. In the remainder we assume that v_i has one arc connecting it to each of $V_{j,in}, V_{j,out}$.

Third, suppose that both arcs of $E_{i,j}$ incident on v_i have the same direction (into or out of v_i). We then color $V_{i,in} \cup V_{i,out}$ with j and $V_{j,in} \cup V_{j,out}$ with i. In the new coloring v_j has at most one neighbor with the same color and v_i has either only in-neighbors or only out-neighbors with color i, so the coloring is acyclic and we again invoke Claim 6.

Finally, we have the case where two arcs of $E_{i,j}$ are incident on v_i, they have different directions, one has its other endpoint in $V_{j,in}$ and the other in $V_{j,out}$. Observe that one of $V_{i,in}, V_{i,out}$ has no arc connecting it to v_j and suppose without loss of generality that it is $V_{i,in}$ (the other case is symmetric). We color $V_{i,in}$ with j and leave $V_{i,out}$ with color i. One of $V_{j,in}, V_{j,out}$ has an incoming arc from v_i; we color this set i and leave the other colored j. Now, v_i only has out-neighbors with color i, while v_j has either only in-neighbors or only out-neighbors colored j, so we are done in this final case. ◀

Our second result concerns a parameter more restricted than feedback vertex set, namely feedback arc set. Note that, in a sense, the class of graphs of bounded feedback arc set contains the class of graphs that have bounded feedback vertex set and bounded degree (selecting all incoming or outgoing arcs of each vertex of a feedback vertex set produces a feedback arc set), so the following theorem may seem more general. However, a closer look reveals that the following result is incomparable to Theorem 5, because graphs of feedback vertex set k and maximum degree $4k-3$ could have feedback arc set of size up to almost $2k^2$ (consider for example a bi-direction of the complete graph $K_{k,2k-2}$), while the following theorem is able to handle graphs of unbounded degree but feedback arc set up to (only) k^2-1. As we show in Theorem 8, this is tight.

▶ **Theorem 7.** *Let D be a digraph with a feedback arc set F of size at most k^2-1. Then D is k-colorable if and only if $D[V(F)]$ is k-colorable, and such a coloring can be extended to D in polynomial time.*

Proof. It is obvious that if $D[V(F)]$ is not k-colorable then D is not k-colorable. We will prove the converse by induction. For $k = 1$ it is trivial to see that if $|F| = 0$ then D is acyclic so is 1-colorable. Assume then that any digraph D with a feedback arc set F of size at most $(k-1)^2-1$ is $(k-1)$-colorable, if and only if $D[V(F)]$ is $(k-1)$-colorable.

Suppose now that we have D with a feedback arc set F with $|F| \leq k^2 - 1$ and we find that $D[V(F)]$ is k-colorable (this can be tested in $2^{O(k^2)}$ time). Let $c : V(F) \to [k]$ be a coloring of $V(F)$. We distinguish two cases:

Case 1. There exists a color class (say V_k) such that at least $2k-1$ arcs of F are incident on V_k. Then $D - V_k$ has a feedback arc set of size at most $|F| - (2k-1) \leq k^2 - 1 - (2k-1) \leq (k-1)^2 - 1$ and $V_1, \dots, V_{k-1}$ remains a valid coloring of the remainder of $V(F)$. So by inductive hypothesis $D - V_k$ has a $(k-1)$-coloring. By using the color k on V_k we have a k-coloring for D.

Case 2. Each color class is incident on at most $2k-2$ arcs of F. We then claim that there is a way to color $V(F)$ so that all arcs of F have distinct colors on their endpoints. If we achieve this, we can trivially extend the coloring to the rest of the graph, as arcs of F become irrelevant. Now, let us call $v \in V(F)$ as *type one* if v is incident on at least k arcs of F. We will show that there is at most one *type one* vertex in each color class. Indeed, if $u, v \in V_i$ are both *type one*, then they are incident on at least $2k-1$ arcs of F (there is no digon between u and v because they share a color), which we assumed is not the case, as V_i is incident on at most $2k-2$ arcs of F. Therefore, we can use k distinct colors to color all the *type one* vertices of $V(F)$. Each remaining vertex of $V(F)$ is incident on at most $k-1$ arcs of F. We consider these vertices in some arbitrary order, and give each a color that does not already appear on the other endpoints of its incident arcs from F. Such a color always exists, and proceeding this way we color all arcs of F with distinct colors. This completes the proof. ◀

4.2 Hardness

In this section we improve upon our previous reduction by producing a graph which has bounded degree and bounded feedback arc set. Our goal is to do this efficiently enough to (almost) match the algorithmic bounds given in the previous section.

▶ **Theorem 8.** *For all $k \geq 2$, it is NP-hard to decide if a digraph $D = (V, E)$ is k-colorable, even if D has a feedback vertex set of size k, a feedback arc set of size k^2, and maximum degree $\Delta = 4k - 1$.*

Proof. Recall that in the proof of Theorem 4 for $k \geq 2$ we construct a graph that is made up of two parts: the palette part, which is a bi-directed clique that contains v_1, v_2 and the $k-2$ vertices we have possibly added to increase the chromatic number (call them $v_3, \dots, v_k$); and the part that represents the formula. We perform the same reduction except that we will now edit the graph to reduce its degree and its feedback arc set. In particular, we delete the palette vertices and replace them with a gadget that we describe below.

We construct a new palette that will be a bi-directed clique of size k, whose vertices are now labeled $v^i, i \in [k]$. Let M be the number of vertices of the graph we constructed for Theorem 4. We construct M "rows" of $2k$ vertices each. More precisely, for each $\ell \in [M], i \in [k]$ we construct the two vertices $v^i_{\ell,in}, v^i_{\ell,out}$. In the remainder, when we refer to row ℓ, we mean the set $\{v^i_{\ell,in}, v^i_{\ell,out} \mid i \in [k]\}$. For all $i, j \in [k], i < j$ we connect the vertices of row 1 to the vertices of the clique as shown in Figure 2. For all $i, j \in [k], i < j$ and $\ell \in [M-1]$ we connect the vertices of rows $\ell, \ell+1$ as shown in Figure 3.

In more detail we have:

1. For each $i \in [k]$ the vertex v^i has an arc to all $v^j_{1,out}$ for $j \geq i$, an arc to $v^j_{1,in}$ for all $j \neq i$, and an arc from $v^j_{1,in}$ for all $j \leq i$.

2. For each $\ell \in [M]$, for all $i < j$ we have the following four arcs: $v^j_{\ell,out}v^i_{\ell,out}$, $v^i_{\ell,out}v^j_{\ell,in}$, $v^j_{\ell,in}v^i_{\ell,in}$, and $v^j_{\ell,out}v^i_{\ell,in}$. As a result, inside a row arcs are oriented from *out* to *in* vertices; and between vertices of the same type from larger to smaller indices i.
3. For each $\ell \in [M-1]$, for all $i \in [k]$ we have the arcs $v^i_{\ell,out}v^i_{\ell+1,out}$ and $v^i_{\ell+1,in}v^i_{\ell,in}$. As a result, the $v^i_{\ell,out}$ vertices form a directed path going out of v^i and the $v^i_{\ell,in}$ vertices form a directed path going into v^i.
4. For each $\ell \in [M-1]$, for all $i, j \in [k]$ with $i < j$ we have the arcs $v^i_{\ell,out}v^j_{\ell+1,in}$, $v^i_{\ell,out}v^j_{\ell+1,out}$, $v^i_{\ell+1,in}v^j_{\ell,in}$, $v^j_{\ell,out}v^i_{\ell+1,in}$. Again, arcs incident on an *out* and an *in* vertex are oriented towards the *in* vertex.

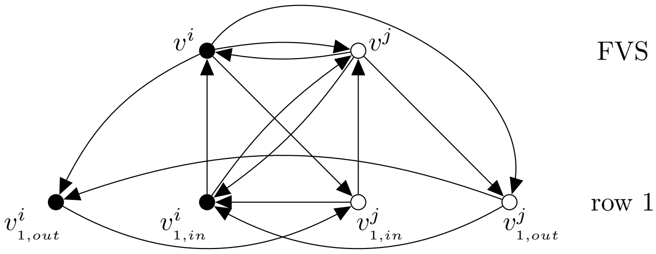


Figure 2 Graph showing the connections between two vertices of the clique palette (v^i, v^j, where $i < j$) and the corresponding vertices of row 1.

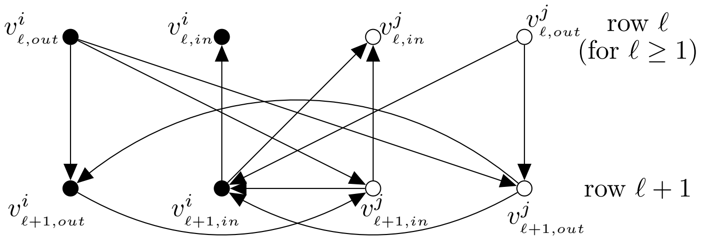


Figure 3 Here we present the way we are connecting the vertices of the rows i and $i + 1$.

Let P be the gadget we have constructed so far, consisting of the clique of size k and the M rows of $2k$ vertices each. We will establish the following properties.

1. Deleting all vertices $v^i, i \in [k]$ makes P acyclic and eliminates all directed paths from any vertex $v^i_{\ell,in}$ to any vertex $v^j_{\ell',out}$, for all $i, j \in [k], \ell, \ell' \in [M]$.
2. The maximum degree of any vertex of P is $4k - 2$.
3. There is a k-coloring of P that gives all vertices of $\{v^i_{\ell,in}, v^i_{\ell,out} \mid \ell \in [M]\}$ color i, for all $i \in [k]$.
4. In any k-coloring of P, for all i, all vertices of $\{v^i_{\ell,in}, v^i_{\ell,out} \mid \ell \in [M]\}$ receive the same color as v^i.

Before we go on to prove these four properties of P, let us explain why they imply the theorem. To complete the construction, we insert P in our graph in the place of the palette clique we were previously using. To each vertex of the original graph, we associate a distinct row of P (there are sufficiently many rows to do this). Now, if vertex u of the original graph, which is associated to row ℓ, had an arc from (respectively to) the vertex v_i in the palette, we add an arc from $v^i_{\ell,out}$ (respectively to $v^i_{\ell,in}$).

Let us first establish that the new graph has the properties promised in the theorem. The maximum degree of any vertex in the main (non-palette) part remains unchanged and is $2k+2 \leq 4k-1$ while the maximum degree of any vertex of P is now at most $4k-1$, as we added at most one arc to each vertex. Deleting $\{v^i \mid i \in [k]\}$ eliminates all cycles in P, but also all cycles going through P, because such a cycle would need to use a path from a vertex $v^i_{\ell,in}$ (since these are the only vertices with incoming arcs from outside P) to a vertex $v^j_{\ell',out}$. Deleting all of P leaves the graph acyclic (recall that the palette clique was a feedback vertex set in our previous construction), so there is a feedback vertex set of size k.

For the feedback arc set we remove the arcs $\{v^j v^i \mid j > i,\ i,j \in [k]\} \cup \{v^i_{1,in} v^j \mid j > i,\ i,j \in [k]\} \cup \{v^i_{1,in} v^i \mid i \in [k]\}$. Note that these are indeed k^2 arcs. To see that this is a feedback arc set, suppose that the graph contains a directed cycle after its removal. This cycle must contain some vertex v^i, because we argued that $\{v^i \mid i \in [k]\}$ is a feedback vertex set. Among these vertices, select the v^i with minimum i. We now examine the arc of the cycle going into v^i. Its tail cannot be v^j for $j > i$, as we have removed such arcs, nor v_j for $j < i$, as this contradicts the minimality of i. It cannot be $v^i_{1,in}$ as we have also removed these arcs. And it cannot be $v^j_{1,in}$ for $j < i$, as these arcs are also removed. But no other in-neighbor of v^i remains, contradiction.

Let us also argue that using the gadget P instead of the palette clique does not affect the k-colorability of the graph. This is not hard to see because, following Properties 3 and 4 we can assume that any k-coloring of P will give color i to all vertices of $\{v^i\} \cup \{v^i_{\ell,in}, v^i_{\ell,out} \mid \ell \in [M]\}$. The important observation is now that, for all $\ell \in [M]$ there will always exist a monochromatic path from v^i to $v^i_{\ell,out}$ and from $v^i_{\ell,in}$ to v^i. We now note that, if we fix a coloring of the non-palette part of the graph, this coloring contains a monochromatic cycle involving vertex v_i of the original palette if and only if the same coloring gives a monochromatic cycle in the new graph going through v^i.

Finally, we need to prove the four properties. Their proofs are given in the appendix. ◀

5 Treewidth

In this section we consider the complexity of Digraph Coloring with respect to parameters measuring the acyclicity of the underlying graph, namely, treewidth and treedepth. Before we proceed let us recall that in all graphs G we have $\chi(G) \leq \text{tw}(G)+1 \leq \text{td}(G)+1$. This means that if our goal is simply to obtain an FPT algorithm then parameterizing by treewidth implies that the graph's chromatic number (and therefore also the digraph's dichromatic number) is bounded. We first present an algorithm with complexity $k^{\text{tw}}(\text{tw}!)$ which, using the above argument, proves that Digraph Coloring is FPT parameterized by treewidth.

▶ **Theorem 9.** *There is an algorithm which, given a digraph D on n vertices and a tree decomposition of its underlying graph of width* tw *decides if D is k-colorable in time $k^{\text{tw}}(\text{tw}!)n^{O(1)}$.*

The proof for the Theorem 9 is deferred to the full version of the paper.

As we explained, even though Theorem 9 implies that Digraph Coloring is FPT parameterized by treewidth, the complexity it gives is significantly worse than the complexity of Coloring, which is essentially k^{tw}. Our main result in this section is to show that this is likely to be inevitable, even if we focus on the more restricted case of treedepth and 2 colors.

▶ **Theorem 10.** *If there exists an algorithm which decides if a given digraph on n vertices and (undirected) treedepth* td *is 2-colorable in time $\text{td}^{o(\text{td})}n^{O(1)}$, then the ETH is false.*

Proof. Suppose we are given a 3-SAT formula ϕ with n variables and m clauses. We will produce a digraph G such that $|V(G)| = 2^{O(n/\log n)}m$ and $\text{td}(G) = O(n/\log n)$ and G is 2-colorable if and only if ϕ is satisfiable. Before we proceed, observe that if we can construct such a graph the theorem follows, as an algorithm with running time $O^*(\text{td}^{o(td)})$ for 2-coloring G would decide the satisfiability of ϕ in time $2^{o(n)}$.

To simplify presentation we assume without loss of generality that n is a power of 2 (otherwise adding dummy variables to ϕ can achieve this while increasing n be a factor of at most 2). We begin the construction of G by creating $\log n$ independent sets $V_1, \dots, V_{\log n}$, each of size $\lceil \frac{2en}{\log^2 n} \rceil$. We add a vertex u and connect it with arcs in both directions to all vertices of $\cup_{i\in[\log n]} V_i$. We also partition the variables of ϕ into $\log n$ sets $X_1, \dots, X_{\log n}$ of size at most $\lceil \frac{n}{\log n} \rceil$.

The main idea of our construction is that the vertices of V_i will represent an assignment to the variables of X_i. Observe that all vertices of V_i are forced to obtain the same color (as all are forced to have a distinct color from u), therefore the way these vertices represent an assignment is via their topological ordering in the DAG they induce together with other vertices of the graph which obtain the same color.

To continue our construction, for each $i \in [\log n]$ we do the following: we enumerate all the possible truth assignments of the variables of X_i and for each such truth assignment $\sigma : X_i \to \{0,1\}^{|X_i|}$ we define (in an arbitrary way) a distinct ordering $\rho(\sigma)$ of the vertices of V_i. We will say that the ordering $\rho(\sigma)$ is the *translation* of assignment σ. Note that there are $|V_i|! \ge (\frac{2en}{\log^2 n})! \ge (\frac{2n}{\log^2 n})^{\frac{2en}{\log^2 n}} = 2^{\frac{2en}{\log^2 n}(1+\log n - 2\log\log n)} > 2^{\lceil \frac{n}{\log n} \rceil}$ for n sufficiently large, so it is possible to translate truth assignments to X_i to orderings of V_i injectively. Note that enumerating all assignments for each group takes time $2^{O(n/\log n)} = 2^{o(n)}$.

Consider now a clause c_j of ϕ and suppose some variable of the group X_i appears in c_j. For each truth assignment σ to X_i which satisfies c_j we construct an independent set $S_{j,i,\sigma}$ of size $|X_i| - 1$, label its vertices $s^\ell_{j,i,\sigma}$, for $\ell \in [|X_i| - 1]$. For each ℓ we add an arc from $\rho(\sigma)^{-1}(\ell)$ to $s^\ell_{j,i,\sigma}$ and an arc from $s^\ell_{j,i,\sigma}$ to $\rho(\sigma)^{-1}(\ell + 1)$. In other words, the ℓ-th vertex of $S_{j,i,\sigma}$ has an incoming arc from the vertex of V_i which is ℓ-th according to the ordering $\rho(\sigma)$ which is the translation of assignment σ and an outgoing arc to the vertex of V_i which is in position $(\ell + 1)$ in the same ordering. Observe that this implies that if all vertices of V_i and of $S_{j,i,\sigma}$ are given the same color, then the topological ordering of the induced DAG will agree with $\rho(\sigma)$ on the vertices of V_i.

To complete the construction, for each clause c_j we do the following: take all independent sets $S_{j,i,\sigma}$ which we have constructed for c_j and order them in a cycle in some arbitrary way. For two sets $S_{j,i,\sigma}, S_{j,i',\sigma'}$ which are consecutive in this cycle add a new "connector" vertex $p_{j,i,\sigma,i',\sigma'}$, all arcs from $S_{j,i,\sigma}$ to this vertex, and all arcs from this vertex to $S_{j,i',\sigma'}$. Finally, we connect each connector vertex $p_{j,i,\sigma,i',\sigma'}$ we have constructed to an arbitrary vertex of V_1 with a digon. This completes the construction.

Let us argue that if ϕ is satisfiable, then G is 2-colorable. We color u with color 2, all the vertices in V_i for $i \in [\log n]$ with 1 and all connector vertices $p_{i,j,\sigma,i',\sigma'}$ with 2. For each clause c_j there exists a group X_i that contains a variable of c_j such that the supposed satisfying assignment of ϕ, when restricted to X_i gives an assignment $\sigma : X_i \to \{0,1\}^{|X_i|}$ which satisfies c_j. Therefore, there exists a corresponding set $S_{j,i,\sigma}$. Color all vertices of this set with 1. After doing this for all clauses, we color all other vertices with 2. We claim this is a valid 2-coloring. Indeed, the graph induced by color 2 is acyclic, as it contains u (but none of its neighbors) and for each c_j, all but one of the sets $S_{j,i,\sigma}$ and the vertices $p_{j,i,\sigma,i',\sigma'}$. Since these sets have been connected in a directed cycle throught connector vertices, and for each c_j we have colored one of these sets with 1, the remaining sets induce a DAG. For the graph induced by color 1 consider for each V_i the ordering $\rho(\sigma)$, where σ is the satisfying assignment restricted to V_i. Every vertex outside V_i which received color 1 and has arcs to V_i, has exactly one incoming and one outgoing arc to V_i. Furthermore, the directions of these arcs agree with the ordering $\rho(\sigma)$. Hence, since $\cup_{i\in[\log n]}V_i$ touches all arcs with both endpoints having color 1 and all such arcs respect the orderings of V_i, the graph induced by color 1 is acyclic.

For the converse direction, suppose we have a 2-coloring of G. Without loss of generality, u has color 2 and $\cup_{i\in[\log n]}V_i$ has color 1. Furthermore, all connectors $p_{j,i,\sigma,i',\sigma'}$ also have color 2. Consider now a clause c_j. We claim that there must be a group $S_{j,i,\sigma}$ such that $S_{j,i,\sigma}$ does not use color 2. Indeed, if all such groups use color 2, since they are linked in a directed cycle with all possible arcs between consecutive groups and connectors, color 2 would not induce a DAG. So, for each c_j we find a group $S_{j,i,\sigma}$ that is fully colored 1 and infer from this the truth assignment σ for the group X_i. Doing this for all clauses gives us an assignment that satisfies every clause. However, we need to argue that the assignment we extract is consistent, that is, there do not exist $S_{j,i,\sigma}$ and $S_{j',i,\sigma'}$ which are fully colored 1 with $\sigma \neq \sigma'$. For the sake of contradiction, suppose that two such sets exist, and recall that $\rho(\sigma) \neq \rho(\sigma')$. We now observe that if $S_{j,i,\sigma} \cup V_i$ only uses color 1, then any topological ordering of V_i in the graph induced by color 1 must agree with $\rho(\sigma)$, which is a total ordering of V_i. In a similar way, the ordering of V_i must agree with $\rho(\sigma')$, so if $\sigma \neq \sigma'$ we get a contradiction.

Finally, let us argue about the parameters of G. For each clause c_j of ϕ we construct an independent set of size $O(n/\log^2 n)$ for each satisfying assignment of a group X_i containing a variable of c_j. There are at most 3 such groups, and each group has at most $2^{n/\log n}$ satisfying assignments for c_j, so $|V(G)| = 2^{O(n/\log n)}m$.

For the treedepth, recall that deleting a vertex decreases treedepth by at most 1. We delete u and all of $\cup_{i\in[\log n]}V_i$ which are $O(n/\log n)$ vertices in total. It now suffices to prove that in the remainder all components have treedepth $O(n/\log n)$. In the remainder every component is made up of the directed cycle formed by sets $S_{j,i,\sigma}$ and connectors $p_{j,i,\sigma,i',\sigma'}$. We first delete a vertex $p_{j,i,\sigma,i',\sigma'}$ to turn the cycle into a directed "path" of length $L = 2^{O(n/\log n)}$. We now use the standard argument which proves that paths of length L have treedepth $\log L$, namely, we delete the $p_{j,i,\sigma,i',\sigma'}$ vertex that is closest to the middle of the path and then recursively do the same in each component. This shows that the remaining graph has treedepth logarithmic in the length of the path, therefore at most $O(n/\log n)$. ◀

6 Conclusions

In this paper we have strengthened known results about the complexity of Digraph Coloring on digraphs which are close to being DAGs, precisely mapping the threshold of tractability for DFVS and FAS; and we precisely bounded the complexity of the problem parameterized by treewidth, uncovering an important discrepancy with its undirected counterpart. One question for further study is to settle the degree bound for which k-Digraph Coloring is NP-hard for DFVS k, and more generally to map out how the tractability threshold for the degree evolves for larger values of the DFVS from $4k - \Theta(1)$ to $2k + \Theta(1)$, which is the correct threshold when the DFVS is unbounded. With regards to undirected structural parameters, it would be interesting to investigate whether a $\mathrm{vc}^{o(\mathrm{vc})}$ algorithm exists for 2-Digraph Coloring, where vc is the input graph's vertex cover, as it seems challenging to extend our hardness result to this more restricted case.

References

1 Pierre Aboulker, Nathann Cohen, Frédéric Havet, William Lochet, Phablo F. S. Moura, and Stéphan Thomassé. Subdivisions in digraphs of large out-degree or large dichromatic number. *Electron. J. Comb.*, 26(3):P3.19, 2019.

2 Stephan Dominique Andres and Winfried Hochstättler. Perfect digraphs. *Journal of Graph Theory*, 79(1):21–29, 2015.

3 Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. A complexity dichotomy for hitting connected minors on bounded treewidth graphs: the chair and the banner draw the boundary. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 951–970. SIAM, 2020. doi:10.1137/1.9781611975994.57.

4 Julien Bensmail, Ararat Harutyunyan, and Ngoc-Khang Le. List coloring digraphs. *Journal of Graph Theory*, 87(4):492–508, 2018.

5 Benjamin Bergougnoux, Édouard Bonnet, Nick Brettell, and O-joung Kwon. Close relatives of feedback vertex set without single-exponential algorithms parameterized by treewidth. *CoRR*, abs/2007.14179, 2020. arXiv:2007.14179.

6 Dietmar Berwanger, Anuj Dawar, Paul Hunter, Stephan Kreutzer, and Jan Obdrzálek. The dag-width of directed graphs. *J. Comb. Theory, Ser. B*, 102(4):900–923, 2012.

7 Drago Bokal, Gasper Fijavz, Martin Juvan, P. Mark Kayll, and Bojan Mohar. The circular chromatic number of a digraph. *Journal of Graph Theory*, 46(3):227–240, 2004. doi:10.1002/jgt.20003.

8 Marthe Bonamy, Lukasz Kowalik, Jesper Nederlof, Michal Pilipczuk, Arkadiusz Socala, and Marcin Wrochna. On directed feedback vertex set parameterized by treewidth. In Andreas Brandstädt, Ekkehard Köhler, and Klaus Meer, editors, *Graph-Theoretic Concepts in Computer Science - 44th International Workshop, WG 2018, Cottbus, Germany, June 27-29, 2018, Proceedings*, volume 11159 of *Lecture Notes in Computer Science*, pages 65–78. Springer, 2018. doi:10.1007/978-3-030-00256-5_6.

9 Édouard Bonnet, Nick Brettell, O-joung Kwon, and Dániel Marx. Generalized feedback vertex set problems on bounded-treewidth graphs: Chordality is the key to single-exponential parameterized algorithms. *Algorithmica*, 81(10):3890–3935, 2019. doi:10.1007/s00453-019-00579-4.

10 Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):21:1–21:19, 2008.

11 Xujin Chen, Xiaodong Hu, and Wenan Zang. A min-max theorem on tournaments. *SIAM J. Comput.*, 37(3):923–937, 2007. doi:10.1137/060649987.

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

13 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. 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.

14 Tomás Feder, Pavol Hell, and Carlos S. Subi. Complexity of acyclic colorings of graphs and digraphs with degree and girth constraints. *CoRR*, abs/1907.00061, 2019. arXiv:1907.00061.

15 Henning Fernau and Daniel Meister. Digraphs of bounded elimination width. *Discret. Appl. Math.*, 168:78–87, 2014.

16 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM J. Comput.*, 39(5):1941–1956, 2010.

17 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Clique-width III: hamiltonian cycle and the odd case of graph coloring. *ACM Trans. Algorithms*, 15(1):9:1–9:27, 2019.

18 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In *IPEC*, volume 8246 of *Lecture Notes in Computer Science*, pages 163–176. Springer, 2013.

19 Robert Ganian, Petr Hlinený, Joachim Kneis, Daniel Meister, Jan Obdrzálek, Peter Rossmanith, and Somnath Sikdar. Are there any good digraph width measures? *J. Comb. Theory, Ser. B*, 116:250–286, 2016.

20 Frank Gurski, Dominique Komander, and Carolin Rehs. Acyclic coloring of special digraphs. *CoRR*, abs/2006.13911, 2020. `arXiv:2006.13911`.

21 Ararat Harutyunyan. Brooks-type results for coloring of digraphs. *PhD Thesis*, Simon Fraser University, 2011.

22 Ararat Harutyunyan, Mark Kayll, Bojan Mohar, and Liam Rafferty. Uniquely d-colorable digraphs with large girth. *Canad. J. Math.*, 64(6):1310–1328, 2012.

23 Ararat Harutyunyan, Tien-Nam Le, Stéphan Thomassé, and Hehui Wu. Coloring tournaments: From local to global. *J. Comb. Theory, Ser. B*, 138:166–171, 2019.

24 Winfried Hochstättler, Felix Schröder, and Raphael Steiner. On the complexity of digraph colourings and vertex arboricity. *Discret. Math. Theor. Comput. Sci.*, 22(1), 2020.

25 Paul Hunter and Stephan Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. *Theor. Comput. Sci.*, 399(3):206–219, 2008.

26 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

27 Lars Jaffke and Bart M. P. Jansen. Fine-grained parameterized complexity analysis of graph coloring problems. In *CIAC*, volume 10236 of *Lecture Notes in Computer Science*, pages 345–356, 2017.

28 Thor Johnson, Neil Robertson, Paul D. Seymour, and Robin Thomas. Directed tree-width. *J. Comb. Theory, Ser. B*, 82(1):138–154, 2001.

29 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. `doi:10.1007/s00453-011-9554-x`.

30 Michael Lampis. Finer tight bounds for coloring on clique-width. In *ICALP*, volume 107 of *LIPIcs*, pages 86:1–86:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

31 Michael Lampis, Georgia Kaouri, and Valia Mitsou. On the algorithmic effectiveness of digraph decompositions and complexity measures. *Discret. Optim.*, 8(1):129–138, 2011.

32 Zhentao Li and Bojan Mohar. Planar digraphs of digirth four are 2-colorable. *SIAM J. Discret. Math.*, 31(3):2201–2205, 2017.

33 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.

34 Marcelo Garlet Millani, Raphael Steiner, and Sebastian Wiederrecht. Colouring non-even digraphs. *CoRR*, abs/1903.02872, 2019. `arXiv:1903.02872`.

35 Bojan Mohar. Circular colorings of edge-weighted graphs. *Journal of Graph Theory*, 43(2):107–116, 2003.

36 Bojan Mohar. Eigenvalues and colorings of digraphs. *Linear Algebra and its Applications*, 432(9):2273–2277, 2010. Special Issue devoted to Selected Papers presented at the Workshop on Spectral Graph Theory with Applications on Computer Science, Combinatorial Optimization and Chemistry (Rio de Janeiro, 2008). `doi:10.1016/j.laa.2009.05.027`.

37 Victor Neumann-Lara. The dichromatic number of a digraph. *J. Comb. Theory, Ser. B*, 33(3):265–270, 1982.

38 Raphael Steiner and Sebastian Wiederrecht. Parameterized algorithms for directed modular width. In *CALDAM*, volume 12016 of *Lecture Notes in Computer Science*, pages 415–426. Springer, 2020.

Good r-Divisions Imply Optimal Amortized Decremental Biconnectivity

Jacob Holm
University of Copenhagen, Denmark

Eva Rotenberg
Technical University of Denmark, Lyngby, Denmark

Abstract

We present a data structure that, given a graph G of n vertices and m edges, and a suitable pair of nested r-divisions of G, preprocesses G in $O(m + n)$ time and handles any series of edge-deletions in $O(m)$ total time while answering queries to pairwise biconnectivity in worst-case $O(1)$ time. In case the vertices are not biconnected, the data structure can return a cutvertex separating them in worst-case $O(1)$ time.

As an immediate consequence, this gives optimal amortized decremental biconnectivity, 2-edge connectivity, and connectivity for large classes of graphs, including planar graphs and other minor free graphs.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Dynamic graphs, 2-connectivity, graph minors, r-divisions, graph separators

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.42

Related Version *Preliminary Full Version*: https://arxiv.org/abs/1808.02568 [26]

Funding *Jacob Holm*: Partially supported by the VILLUM Foundation grant 16582, "BARC".
Eva Rotenberg: Partially supported by Independent Research Fund Denmark grants 2020-2023 (9131-00044B) "Dynamic Network Analysis" and 2018-2021 (8021-00249B), "AlgoGraph", and the VILLUM Foundation grant 37507 "Efficient Recomputations for Changeful Problems".

Acknowledgements We are thankful to Adam Karczmarz and Jakub Łącki for their encouragement and interest in this work.

1 Introduction

Dynamic graph problems concern maintaining information about a graph, as it undergoes changes. In this paper, the changes we allow are deletions of edges or vertices by an adaptive adversary. The information we maintain is a representation that reflects biconnectivity of vertices, that is, whether they are connected after the removal of any vertex of the graph.

A static (non-changing) graph may in $O(n + m)$ time be pre-processed to answer biconnectivity queries in worst-case $O(1)$ time. This is done by finding the *blocks*, i.e. the biconnected components. We show, for a large class of graphs including minor free graphs, that in the same asymptotic total time, we can handle any sequence of edge- and vertex deletions, while still answering biconnectivity queries, 2-edge connectivity queries, and connectivity queries, in worst-case $O(1)$ time.

If a pair of vertices are not biconnected, then there exists a certificate for this in form of a cutvertex separating them. A natural question, if a pair of vertices are not biconnected, is thus to ask for such a certificate. There may be many cutvertices separating a pair of vertices, so an even more advanced and desired functionality is the ability to point to the cutvertex furthest towards either one of them. Again, for a large class of graphs, our running time for a *decremental* graph matches the state of the art for *non-changing graphs*, by revealing the nearest cutvertex in $O(1)$ worst-case time, while spending only $O(n + m)$ total time for both preprocessing the graph and handling any sequence of deletions.

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 42; pp. 42:1–42:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

When edges and vertices are both deleted and inserted, there are non-trivial lower bounds [36] saying that no data structure for connectivity has both update- and query-time in $o(\log n)$. This is in stark contrast to the incremental situation, where only edge-insertions are allowed, in which the α-time algorithm for union-find is tight [39, 11]. When restricted to deletions, however, even for general graphs, there are no known lower bounds beyond the trivial $O(|G|)$. The research in this paper is inspired by the fundamental open question of whether decremental (deletion-only) connectivity [41], 2-edge connectivity, biconnectivity, or even minimum cut for general graphs can be solved in amortized constant time per edge-deletion, or whether non-trivial lower bounds do exist.

The following table shows how we improve state-of-the-art for planar graphs and minor-free graphs. Here, we present maximum amortized time *per operation*, that is, we do not require $O(1)$ query time. When restricted to constant query time, the best biconnectivity algorithms for non-planar sparse graphs were fully dynamic and had an update time of $\tilde{O}(\sqrt{n})$ [18].

Table 1 Our improvements (**now**) in relation to previous results (*previous*). The table shows amortized time per operation. The table compares with state-of-the-art amortized deterministic algorithms. Allowing randomization, the previous best decremental connectivity algorithm runs in time $\tilde{O}(\log n)$ [42].

	planar	bnd. genus	minor-free graphs	
	previous	*previous*	*previous*	**now**
connectivity	$O(1)$ [34]	$O(\log n)$ [4]	$O(\frac{\log^2 n}{\log\log n})$ [44]	$\mathbf{O(1)}$
2-edge-connectivity	$O(1)$ [25]	$\tilde{O}(\log^2 n)$ [27]		$\mathbf{O(1)}$
biconnectivity	$O(\log n)$ [25]	$\tilde{O}(\log^3 n)$ [27]		$\mathbf{O(1)}$

Dynamic graph connectivity has been studied for decades. Most general is fully dynamic connectivity for general graphs [8, 21, 20, 23, 42, 29, 28, 44, 30, 35], where edges are allowed to be both inserted and deleted. Similarly, fully dynamic two-edge connectivity and biconnectivity have been studied [10, 18, 5, 19, 23, 42, 27] and have algorithms with polylogarithmic update- and query time. For special graph classes, such as planar graphs, graphs of bounded genus, and minor-free graphs, there has been a bulk of work on connectivity and higher connectivity, e.g. [7, 22, 13, 15, 6, 33, 34, 25, 24].

An r-division is, intuitively, a family of $O(n/r)$ subgraphs called the *regions*, with $O(r)$ vertices each, such that the regions partition the edges, and each region shares $O(\sqrt{r})$ *boundary vertices* with the rest of the graph. The concept of r-divisions was introduced in [9] as a tool for finding shortest paths in planar graphs. It naturally generalizes the notion of a separator: a small set of vertices that cause the graph to fall apart into *two* regions, each containing a constant fraction of the original graph [32].

Later, Henzinger et al. [17] generalized this to the concept of a *strict* (r,s)*-division*, which is a family of $O(n/r)$ subgraphs called the regions, each with at most r vertices, that partition the edges, and where each region has at most s boundary vertices. An r-division is thus a strict $(O(r), O(s))$-division. For the rest of our paper, we will use the term r-division to mean any strict $(r, O(r^{1-\epsilon}))$-division for some suitable r, ε. There are algorithms [2, 3, 14, 31] for computing r-divisions of planar graphs in linear time.

Our results. We give a data structure for maintaining biconnectivity for a large class of graphs. In order to state our theorem in its fullest generality, we need to define what it means for a pair of r-divisions to be *a suitable pair*.

Given a graph G with n vertices, we call a pair $(\mathcal{A}, \mathcal{R})$ where $\mathcal{A}$ is a strict (r_1, s_1)-division and $\mathcal{R}$ is a strict (r_2, s_2)-division *a t-suitable pair of r-divisions* if:

- there exists an algorithm for fully dynamic biconnectivity in general graphs with amortized time $t(n)$ per operation[1], such that:
- each boundary vertex of $\mathcal{A}$ is also a boundary vertex of $\mathcal{R}$ ($\partial\mathcal{A} \subseteq \partial\mathcal{R}$); and
- for each region $A \in \mathcal{A}$, $\mathcal{R}$ contains a partition of A into $O(\frac{r_1}{r_2})$ regions of size at most r_2, each having at most s_2 boundary vertices[2]; and
- $r_1, s_1 \in O(\text{poly}(\log n))$ and $\frac{r_1}{s_1} \in \Omega(t(n) \log n)$; and
- $r_2, s_2 \in O(\text{poly}(\log \log n))$ and $\frac{r_2}{s_2} \in \Omega(t(r_1) \log r_1)$.

When t is understood, we will refer to them as simply *suitable*.

Our data structure answers queries to biconnectivity, i.e, a pair of vertices are biconnected if they are connected and not separated by any bridge or cutvertex. If the vertices u and v are connected but not biconnected, we can output a cutvertex separating them, in fact, we can output that of the possibly many cutvertices that is *nearest* to u – we call this the *nearest cutvertex* – or detect the special case where uv is a bridge.

▶ **Theorem 1.** *There exists a data structure that given a graph G with n vertices and m edges, and given a suitable pair of r-divisions, preprocesses G in $O(m + n)$ time and handles any series of edge-deletions in $O(m)$ total time while answering queries to pairwise biconnectivity and queries to nearest cutvertex in $O(1)$ time.*

This can immediately be combined with any algorithm for finding suitable r-divisions in linear time, to obtain optimal decremental biconnectivity data structures for graphs that are planar, bounded genus, or minor free.

The data structure is easily extended to maintain information about connectivity, so as to answer queries to pairwise connectivity in $O(1)$ time, and our techniques can easily be used to obtain a decremental data structure for 2-edge connectivity with the same update- and query times.

For completeness, the full version of this paper presents linear time algorithms for finding r-divisions in minor-free graphs using techniques from [37, 40, 43][3]

▶ **Lemma 2.** *Given a graph G that does not have a K_ℓ-minor, for any $r \in \Omega(\log n)$ we can compute a strict $(r, O(r^{\frac{2}{3}} \log^{\frac{1}{3}} n))$-division in linear time*[4].

Thus, we have the following consequence as a corollary to Theorem 1:

▶ **Corollary 3.** *There exists a data structure that given a minor-free graph G with n vertices, preprocesses G in $O(n)$ time and handles any series of edge- and vertex deletions in $O(n)$ total time while answering queries to pairwise connectivity, 2-edge connectivity, biconnectivity, nearest separating bridge in $O(1)$, and nearest separating cutvertex, in $O(1)$ time.*

[1] e.g. $t(n) = O(\log^5 n)$ using [23], and $t(n) = O(\log^3 n \cdot \log^2 \log n)$ using [27]

[2] This is slightly weaker than requiring $\mathcal{R}$ to contain a strict (r_2, s_2)-division of A.

[3] This result was first claimed by Henzinger et al. [17], but their solution only works for planar graphs, and h-minor-free graphs of bounded degree.

[4] We failed to find a reference in the literature for this fact, but we would not be surprised if it is common knowledge.

Even for graphs where no linear-time algorithms for finding r-divisions are known, our results may still be of interest: as soon as the r-divisions have been computed once, they may be used for several different edge-deletion sequences on the same graph.

Our paper can be seen as a generalization of and improvement upon [34], who showed optimal amortized decremental connectivity for planar graphs, that is, amortized constant update time, and worst-case constant query time.

Note that we generalise [34] in two important ways: We generalise from planar graphs to r-divisible graphs, and we generalise from plain connectivity to handle 2-edge-connectivity and biconnectivity. Our generalisation works by getting rid of some unneccesarily planar-specific techniques and replacing them with a more general framework.

We expect this general framework to be of future interest for deriving optimal decremental algorithms from other dynamic algorithms that have polylog update time: as long as a compact representation of each region can be maintained efficiently, with respect to the the graph property of interest, this can be used in our framework for speeding up decremental algorithms.

1.1 Techniques

Since the property of being an r-division is not violated as edges are deleted, it is natural to use r-divisions to get better decremental data structures for graphs. The idea is to have a top-level graph with size only proportional to the number of boundary vertices, and to handle the regions efficiently simply because they are smaller.

With biconnectivity, the first challenge is to design the top-level graph: a vertex may be not biconnected to any boundary vertex in its region, but yet be biconnected with some other vertex in another region via two separate boundary vertices (see Figure 1). Even vertices from the same region may be biconnected in G although they are not biconnected, or even connected, within the region.

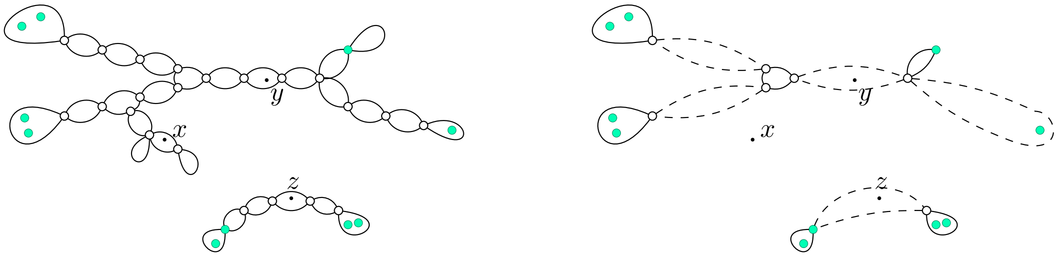


Figure 1 Left: A region R with 10 boundary vertices (green). There is a vertex separating x from the boundary, so x is never biconnected with anything in $G \setminus R$. Vertices y and z however, are not even connected in R, but may be biconnected in G. Right: The structure may be compressed in the sense depicted: x is not represented at all, while y and z are represented in pseudo-blocks (dashed).

We thus need to store an efficient representation of the biconnectivity of the region as seen from the perspective of the boundary vertices. We call this efficient representation the *compressed BC-forest* (see Section 4). It is obtained from the forest of BC-trees (also known as the block-cutpoint trees, see Section 2) by first marking certain blocks and cutvertices as *critical*, and then, basically, contracting the paths that connect them. The critical blocks and cutvertices are spartanly chosen, such that the total size of all the compressed BC-forests is only proportional to the boundary itself. We stitch the compressed BC-forests together by the boundary vertices they share, and obtain the *patchwork graph* (see Figure 2), in which all vertices that are biconnected to anything outside their region are *represented*, and we use the *representatives* of vertices to reveal when they are biconnected by paths that go via boundary vertices. A construction very similar to our compressed BC-forests appears in [12], where it is used in a separator tree for a planar graph, but the rules for what to contract are subtly different.

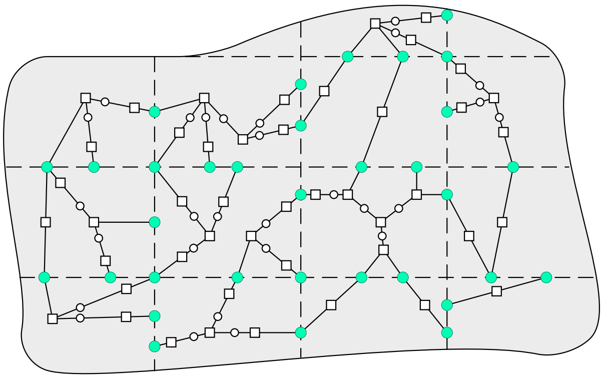


Figure 2 An r-division and its corresponding patchwork graph. The graph is bipartite between, on one hand, round boundary vertices and cutvertices, and, on the other hand, square blocks and contracted (pseudo) blocks.

If decremental changes to a region only gave rise to decremental changes to its forest of BC-trees, we would be close to done. However, and this is the second challenge, the deletion of an edge can cause a block to fall apart into a chain of blocks. Luckily, the damage to the compressed BC-forest is containable: only $O(n/\operatorname{polylog} n)$ vertices can be present in the compressed BC-forest, and the changes can be modeled by only three operations: edge- or path deletions, certain forms of vertex splits, and contractions of paths. These operations, we show, are of a form that can be handled in polylogarithmic time by one of the fully-dynamic biconnectivity data structures (see Section 4).

While using r-divisions once would obtain an improvement from polylog to polyloglog, which might, in practice, be useful already, it is tempting to form r-divisions of the regions themselves and use recursion in order to obtain an even faster speedup (see Figure 3). This would mean that each region should again contain a patchwork made from the compressed BC-forests of its subregions (and, luckily, these patchwork operations compose beautifully). Thus, via recursion, one can obtain a purely combinatorial data structure with $O(\log^* n)$ update- and query time. But in fact, with standard RAM-tricks, if the subregions are of only polyloglog size, one can handle any operation in constant time – simply by using a look-up table. Thus, in the practical RAM-model (i.e. the RAM-model with standard AC^0 operations such as addition, subtraction, bitwise and/or/xor), we can make do with only 3 levels (top, middle and bottom), and obtain $O(1)$ update- and query-time.

Here, as our third challenge, we face that one does not simply recurse into optimality – we need to assure ourselves that when a deletion of an edge causes changes in the compressed BC-trees of the subregion, the changes to the patchwork graph on the level above are manageable. Here, we show that our carefully chosen forms of vertex splits and path contractions do indeed only give rise to the same variant of splits and contractions on the parent level.

Finally, when a pair of vertices u, v are connected but not biconnected, we can in constant time find the nearest cutvertex on any path from u to v – this is called the *nearest cutvertex problem* (see Figure 4). While outputting *some* cutvertex separating u and v is

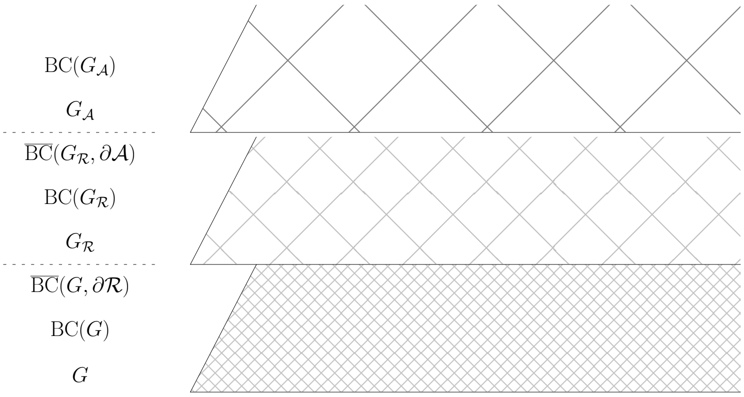

$\mathrm{BC}(G_{\mathcal{A}})$
$G_{\mathcal{A}}$
$\overline{\mathrm{BC}}(G_{\mathcal{R}}, \partial\mathcal{A})$
$\mathrm{BC}(G_{\mathcal{R}})$
$G_{\mathcal{R}}$
$\overline{\mathrm{BC}}(G, \partial\mathcal{R})$
$\mathrm{BC}(G)$
G

Figure 3 We use nested r-divisions and obtain a levelled structure. Each level maintains a graph, its BC-tree, and, for the non-top levels, the compressed BC-tree with relation to the boundary.

easy, augmenting the BC-tree with enough information to facilitate nearest cutvertex queries is technically more demanding. We show that the nearest cutvertex can be determined by at most one nearest cutvertex and one biconnected query in the patchwork graph, and at most one nearest cutvertex and one biconnected query in the region. We also show how to augment an explicit representation of the BC-tree subject to certain splits, contractions, and deletions such that we can still access the nearest cutvertex - a problem that reduces to first-on-path on a dynamic tree subject to certain vertex splits, and certain edge contractions and deletions. Specifically, we exploit an intricate flavour of monotonicity: Although blocks can be split arbitrarily, once an element of the structure has participated in a contraction, it will not be subject to further splitting. We solve this by solving a seemingly harder problem on such trees, namely that of answering an extended form of the nearest common ancestor query, known as the *characteristic ancestor* query. This may be of independent interest.

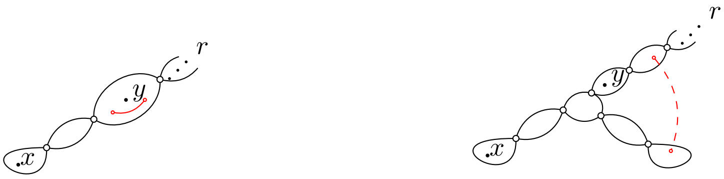


Figure 4 An edge-deletion (red) in the graph can lead to a split of a block which changes the nearest cutvertex from y towards x.

Related techniques. The idea of using recursive separators stems from the sparsification techniques from [5, 6], where it secured $O(\sqrt{n})$ update algorithms for a series of problems, and the idea of using two levels of regions of size $O(\text{polylog}\, n)$ and $O(\text{poly} \log\log n)$, respectively, was introduced in [34] where the idea, together with a union-find structure in the dual graphs, was used to obtain amortized $O(1)$ decremental connectivity for planar graphs.

Paper outline. Section 2 is dedicated to preliminaries and terminology. Then, in Section 3, we introduce the notion of *capacitated biconnectivity*, which is a tool for overcoming the third challenge of making the recursion work. Section 4 is dedicated to an understanding of the patchwork graph in a static setting: how it is defined, how it reflects biconnectivity, and how it behaves when there is not one but two or more nested r-divisions of the same graph. Finally, in Section 5, we show how to maintain the patchwork graph decrementally, thus enabling us solve decremental biconnectivity. The extention to handle nearest cutvertex queries is deferred to Section 6. The extension uses our characteristic ancestors structure, which is deferred to the full version (preliminary version available as [26]). Also deferred to the full version is the reductions that handle 2-edge-connectivity and connectivity and the linear time construction of r-divisions for minor-free graph classes.

2 Preliminaries

Given a graph with vertices u and v, we say they are *connected* if there is a path connecting them. A pair of connected vertices are 2*-edge connected* unless there is an edge whose removal would disconnect them. Such an edge is called a *bridge*. A pair of 2-edge connected vertices u and v are *(locally) biconnected* unless there exists a vertex (other than u and v) whose removal would disconnect them. Such a vertex is called a *cutvertex*. For an ordered pair (u, v) of connected but not biconnected vertices, the *nearest cutvertex* separating them is uniquely defined as the first cutvertex on a path – any path – from u to v. In the special case where u and v are separated by the bridge uv, we say that the nearest cutvertex is **nil**.

The *blocks* of a graph are the maximal biconnected subgraphs. Each block is either a bridge or a maximal set of biconnected vertices. For each connected component of a graph, the *block-cutpoint tree* [16, p. 36], or *BC-tree* for short, reflects the biconnectivity among the vertices. This tree has all the vertices of the graph and, furthermore, a vertex for each block. Its edges are those that connect each vertex to the block or blocks it belongs to. If the graph G is not necessarily connected, its *BC-forest* $\mathrm{BC}(G)$ has a BC-tree for each connected component of the graph. The BC-forest of a graph can be found in linear time [38].

If each BC-tree in the BC-forest is rooted at an arbitrary block, each non-root block has one unique cutvertex separating it from its parent. Then, a pair of vertices are biconnected if and only if they either have the same non-bridge block as parent, or one is the parent of the non-bridge block that is parent of the other.

A dynamic data structure for biconnectivity in general graphs is developed in [23, 42, 27]; it maintains an n-vertex graph and handles deletions and insertions of edges in $t(n) = O(\log^3 n \cdot \log^2 \log n)$ amortized time, and answers queries in $O(\log^2 n \cdot \log^2 \log n)$ worst-case time. The data structure is easily modified to give the first cutvertex separating a pair of vertices in $O(\log^2 n \cdot \log^2 \log n)$ time, but even without this modification, one can find the first cutvertex via a binary search along a spanning tree in $O(\log n)$ queries in $O(\log^3 n \cdot \log^2 \log n)$ worst case time. Note however that for our purposes, the original [23] data structure with $O(\log^5 n)$ amortized update- and query time is sufficient. For the rest of this paper, we will just use $t(n)$ to denote the amortized time per operation (queries included) of a fully dynamic biconnectivity structure for general graphs.

For (not necessarily distinct) vertices v, u, w in a tree, we use $v \longleftrightarrow u$ to denote the tree-path connecting v and u, and we use $\mathrm{meet}(u, v, w)$ to denote the unique common vertex of all three tree-paths connecting them.

A *strict* (r, s)*-division* is a set of $O(n/r)$ subgraphs $\mathcal{R} = \{R_1, R_2, \ldots\}$ called *regions*, that partition the edges. Each region $R \in \mathcal{R}$ has at most r vertices, and a set ∂R of at most s *boundary vertices*, such that only boundary vertices appear in more than one region. We denote by $\partial\mathcal{R}$ the set of all boundary vertices $\bigcup_{R\in\mathcal{R}} \partial R$. Note that with these definitions, $\sum_{R\in\mathcal{R}} |\partial R| \leq O(n/r) \cdot O(s) = O(n \cdot \frac{s}{r})$.

An r-division usually means a strict (r, s)-division with $s = O(\sqrt{r})$, but we will be using it more broadly to include any strict (r, s)-division, where $s = O(r^{1-\varepsilon})$ for some $\varepsilon > 0$.

We say that a pair $(\mathcal{A}, \mathcal{R})$ consisting of an r_1-division and an r_2-division are *nested*, if $\partial\mathcal{A} \subseteq \partial\mathcal{R}$, and $\mathcal{R}$ contains an r_2-division of each region of $\mathcal{A}$. With a slight abuse of notation, for any $A \in \mathcal{A}$ we will let $\mathcal{R} \cap A$ denote this r_2-division.

3 Bicapacitated biconnectivity

Consider the BC-forest of a graph. It may be viewed as a *bicapacitated graph*, where non-bridge blocks have capacity 2 and bridge blocks and vertices have capacity 1; then, vertices u and v in G are biconnected exactly when there exists a flow of value 2 from u to v in the BC-forest of G. (Disregarding the capacity of the source and sink vertices.) We denote by *bicapacitated biconnectivity* the query to the existence of such a flow.

Recall that we want to be able to use the framework recursively: we want to build and maintain BC-trees for small graphs and stitch them, or rather, compressed versions of them together, thus obtaining a patchwork graph. So, we need to extend our definitions so that they can handle a bicapacitated input graph corresponding to the BC-trees of an underlying region. The resulting patchwork graphs are always bipartite, with vertices on one side all having capacity 1, and vertices on the other side having capacity either 1 or 2. We will restrict our definition of bicapacitated graph to mean such graphs.

Now, we can introduce the problem of *fully dynamic bicapacitated biconnectivity*, as that of facilitating bicapacitated biconnectivity queries between vertices in a bicapacitated graph as it undergoes insertions and deletions of edges. Note that (fully) dynamic bicapacitated biconnectivity has an easy reduction to (fully) dynamic biconnectivity:

▶ **Lemma 4.** *Given a fully dynamic data structure for biconnectivity in general graphs using amortized $t_u(n)$ time per link or cut and (amortized/worst case) $t_q(n)$ per pairwise biconnectivity or nearest cutvertex query, there is a fully dynamic data structure for bicapacitated graphs that uses $O(t_u(2n))$ amortized time per edge insert/delete, and answers pairwise biconnectivity and nearest-cutvertex queries in (amortized/worst case) $O(t_q(2n))$ time.*

4 The patchwork graph

We are given an r-division $\mathcal{R} = \{R_1, \ldots, R_k\}$ of G, and we want to define a graph $G_\mathcal{R}$ of size $O(|\partial\mathcal{R}|)$ that somehow captures all the biconnectivity relations that cross multiple regions. We call the resulting $G_\mathcal{R}$ a *patchwork graph*, because it is built by stitching together a suitable *patch graph* for each region.

Our patch graph for each region is in turn based on the BC-forest for the region. We *compress* the BC-forest of the region similarly to [12] as follows:

▶ **Definition 5.** *Given a bicapacitated graph $G = (V, E)$, its BC-forest $F = \mathrm{BC}(G)$, and a subset of vertices $S \subseteq V$, define a node*[5] *$x \in T$, where the tree T is a component of F, to be*

- S-critical *if $x = \mathrm{meet}_T(s_1, s_2, s_3)$ for some $s_1, s_2, s_3 \in S$,*
- S-disposable *if $x \notin s_1 \longleftrightarrow_T s_2$ for all $s_1, s_2 \in S$, and*
- S-contractible *otherwise.*

▶ **Definition 6.** *The* compressed BC-forest $\overline{\mathrm{BC}}(G, S)$ *is the forest obtained from its forest of BC-trees by deleting all S-disposable nodes, and replacing each maximal path of S-contractible nodes that start and end in distinct blocks, with a single so-called* pseudoblock *node with capacity 1.*

[5] Throughout the text we consistently denote vertices of G by *vertices*, and vertices of BC-trees and SPQR-trees as *nodes*.

▶ **Definition 7.** *Given an r-division $\mathcal{R} = \{R_1, \dots, R_k\}$ of a graph G, define the* patchwork graph $G_{\mathcal{R}} = \bigcup_{R \in \mathcal{R}} \overline{\mathrm{BC}}(R, \partial R)$ *to be the bicapacitated graph obtained by taking the (non-disjoint) union of compressed BC-forests $\overline{\mathrm{BC}}(R, \partial R)$ for each region $R \in \mathcal{R}$.*

Any vertex of G corresponds to a BC-vertex in $\mathrm{BC}(R)$ for some R. Some of these BC-vertices are either present or represented in $G_{\mathcal{R}}$. We thus want to define the representation of a vertex as the vertex in $G_{\mathcal{R}}$ representing its BC-node, when it exists:

▶ **Definition 8.** *Given a patchwork graph $G_{\mathcal{R}}$ and a vertex v of G, we define the representative $\overline{\mathrm{B}}(v)$ of v as follows:*

- *If v is a vertex of $G_{\mathcal{R}}$, then $\overline{\mathrm{B}}(v) = v$; else*
- *Let $R \in \mathcal{R}$ be the unique region containing v. If v is incident to a block in* $\mathrm{BC}(R)$ *that is not S-disposable, then v is represented either by that block or the pseudoblock representing it.*
- *Otherwise, v is not represented.*

Overloading notation slightly, say that a vertex of the graph is critical, disposable, or contractible, if the BC-node representing it is.

▶ **Observation 9.** *There is a linear time algorithm for building the compressed BC-forest of a graph with respect to a given subset of vertices, and for finding the representatives of the vertices.*

▶ **Lemma 10.** *Distinct vertices u, v are biconnected in G if and only if either*

1. *At least one of u, v is not a boundary vertex, and u, v are biconnected in the at most one region R containing both; or*
2. *$\overline{\mathrm{B}}(u) = \overline{\mathrm{B}}(v)$ is a pseudo-block whose unique neighbours are biconnected in $G_{\mathcal{R}}$; or*
3. *$\overline{\mathrm{B}}(u)$ and $\overline{\mathrm{B}}(v)$ are different and are biconnected in $G_{\mathcal{R}}$.*

Proof. We will show that u and v are *not* biconnected if and only if all three conditions are false. Assume u and v are not biconnected. Then they can clearly not be biconnected within some region R, so condition 1 is false. If $\overline{\mathrm{B}}(u) = \overline{\mathrm{B}}(v)$, then this is a pseudo-block contracted from a chain containing the neighbors of u, v in $\mathrm{BC}(R)$, and a cutpoint c that separates them within R. Consider the neighbors u' and v' to this pseudoblock in $G_{\mathcal{R}}$. If they were biconnected in $G_{\mathcal{R}}$ there would be a u, v path in $G \setminus \{c\}$ contradicting our choice of u, v. Thus u' and v' are not biconnected in $G_{\mathcal{R}}$ and condition 2 is false. Finally, if $\overline{\mathrm{B}}(u)$ and $\overline{\mathrm{B}}(v)$ are different, then any cutvertex c separating u and v in G will either be a cutvertex in $G_{\mathcal{R}}$, or will be in a pseudoblock $\overline{\mathrm{B}}(c)$ with neighbors u' and v'. Since c is a cutvertex in G, $(u', \overline{\mathrm{B}}(c))$ and $(\overline{\mathrm{B}}(c), v')$ are bridges in $G_{\mathcal{R}}$, and $\overline{\mathrm{B}}(u)$ and $\overline{\mathrm{B}}(v)$ will be separated by at least one of them and are therefore not biconnected in $G_{\mathcal{R}}$ and condition 3 is false.

If, on the other hand, none of the three conditions are true, then, if $\overline{\mathrm{B}}(u) = \overline{\mathrm{B}}(v)$ is a pseudoblock whose neighbours in $G_{\mathcal{R}}$ are not biconnected, then any cutvertex separating u from v in their region also separates them in G. If $\overline{\mathrm{B}}(u) \neq \overline{\mathrm{B}}(v)$ are separable by some cutvertex c in $G_{\mathcal{R}}$, then c is also a cutvertex separating u from v in G, and hence they are not biconnected. If $\overline{\mathrm{B}}(u) \neq \overline{\mathrm{B}}(v)$ are the endpoints of a bridge in $G_{\mathcal{R}}$ then one of them must be a pseudoblock containing a cutvertex or a bridge in G separating them. ◀

Lemma 10 above almost enables us to transform a biconnectivity-query in G into a biconnectivity-query in $G_{\mathcal{R}}$ and a biconnectivity inside a region R. However, item 1 is only directly useful when neither of the vertices belong to the boundary; when one is a boundary vertex we do not know which vertex in the region it corresponds to. Fortunately, when the non-boundary vertex is represented, we may query biconnectivity in $G_{\mathcal{R}}$ to obtain the answer. To handle disposable vertices, we introduce the notion of the *nearest represented vertex*:

▶ **Definition 11.** *When an S-disposable vertex v is connected to at least one boundary vertex b, it knows its* nearest represented vertex $\mathrm{nr}(v)$ *which is the first non-disposable node in the BC-tree of the region on the path from b to v (note that this node is one unique cutvertex). When an S-disposable vertex v is not connected to the boundary, it has* $\mathrm{nr}(v) = \mathbf{nil}$.

Note also that item 3 requires the pseudo-block to know its exactly two neighbours.

▶ **Lemma 12.** *Vertices u and v are biconnected if and only if either*

- *u and v are non-boundary vertices of the same region and are biconnected in the region,*
- *u is a non-boundary vertex that is biconnected in its region R with* $\mathrm{nr}(u)$ *and* $\mathrm{nr}(u) = v$*,*
- $\overline{\mathrm{B}}(u) = \overline{\mathrm{B}}(v)$ *is a pseudo-block and its neighbours are biconnected in $G_{\mathcal{R}}$, or*
- $\overline{\mathrm{B}}(u) \neq \overline{\mathrm{B}}(v)$ *are biconnected in $G_{\mathcal{R}}$*

Proof. Follows from Lemma 10 by expanding item 1 into the two cases of whether both or only one vertex is non-boundary. ◀

Note that patchwork graphs are well-behaved and respect sub-divisions of r-divisions in the following sense:

▶ **Lemma 13.** *If $S \subseteq \partial\mathcal{R}$, then* $\overline{\mathrm{BC}}(G, S) = \overline{\mathrm{BC}}(G_{\mathcal{R}}, S)$

Proof. There is a correspondence between the critical, disposable, and contractible BC-nodes.

Consider an S-critical BC-node x of G. It may overlap with several regions. However, in each region, each vertex of x lies on some $r_1 \longleftrightarrow r_2$ path for $r_1, r_2 \in \partial\mathcal{R}$, so they are never disposable. But then, since $S \subseteq \partial\mathcal{R}$, x is also $\partial\mathcal{R}$-critical, and thus, present in $G_{\mathcal{R}}$. Clearly, once the block is present in $G_{\mathcal{R}}$, it is also S-critical in $G_{\mathcal{R}}$.

If a BC-node of G is S-disposable, we only need to observe that its $\partial\mathcal{R}$-contractible and $\partial\mathcal{R}$-critical parts, for each path $r_1 \longleftrightarrow r_2$ they lie on, at most one endpoint is not S-disposable.

Finally, if a BC-node x of G is S-contractible, then it lies on some path $s_1 \longleftrightarrow s_2$, which $\partial\mathcal{R}$ cuts up into subpaths $r_1 \longleftrightarrow r_2 \longleftrightarrow r_3 \longleftrightarrow \ldots$ in (not necessarily different) regions $R_1, R_2, R_3, \ldots$. But then, all parts of x are preserved as either $\partial\mathcal{R}$-critical or $\partial\mathcal{R}$-contractible $\overline{\mathrm{BC}}(R_i)$-vertices, and thus, survive in $\overline{\mathrm{BC}}(G_{\mathcal{R}}, S)$. On the other hand, if a vertex in R_i does not belong in x, then it does not lie on any of the paths $r_j \longleftrightarrow r_{j+1}$, and can thus not be represented by a vertex or a pseudo-block on that path. ◀

The same lines of thought can be used to make the following observation about how nested r-divisions behave with respect to patchwork graphs:

▶ **Observation 14.** *If $\partial\mathcal{R}_1 \subseteq \partial\mathcal{R}_2$, then $G_{\mathcal{R}_1} = (G_{\mathcal{R}_2})_{\mathcal{R}_1}$.*

5 Decremental Biconnectivity in Patchwork Graphs

Given the BC-forest for (the patchwork graph associated with) each region of G in an r-division $\mathcal{R}$, we want to explicitly maintain $G_{\mathcal{R}}$ and $\mathrm{BC}(G_{\mathcal{R}})$.

Let R' be a bicapacitated graph associated with region R, and suppose that $\overline{\mathrm{BC}}(R', \partial R) = \overline{\mathrm{BC}}(R, \partial R)$. We will arrange things so either $R' = R$ (with all vertices having capacity 1), or $R' = R_{\mathcal{R}'}$ for some r-division $\mathcal{R}'$ of R with $\partial R \subseteq \partial\mathcal{R}'$, so the equality follows from Lemma 13.

We will maintain a fully dynamic biconnectivity structure for R' with amortized time $t(n) \in O(\mathrm{poly}(\log n))$ per operation, e.g. using [23][6]. We use this structure to explicitly maintain $\mathrm{BC}(R')$ under the following operations:

[6] A faster algorithm here would just make more pairs of r-divisions suitable.

path deletion – given a path between two vertices of capacity 1, whose internal vertices all have degree 2, deletes all edges and internal vertices on the path.

block split – given a vertex u of capacity 2, and an adjacent vertex v of capacity 1, split u into two vertices u_1, u_2 of capacity 2 connected by a path with 2 edges via v, with u_1, u_2 partitioning the remaining neighbors of u.

pseudoblock contraction – given a path of 3 vertices, all having degree 2 and the middle having capacity 1, contract the path to a single vertex with capacity 1.

The point is that if one of these operations is applied to R', then the change to $\mathrm{BC}(R')$ and $\overline{\mathrm{BC}}(R', \partial R)$ can also be described by a sequence of these operations.[7]

▶ **Lemma 15.** *There is a data structure that explicitly maintains* $\mathrm{BC}(R')$ *that can be initialized, and support any sequence of* $O(|R'|)$ *path deletions, block splits, and pseudoblock contractions, in* $O(|R'|t(n')\log n')$ *total time, where* n' *is the number of vertices in* R'*.*

Proof. Use the data structure from Lemma 4 as a subroutine. Start by inserting all the edges. Each pseudoblock contraction can be simulated using a constant number of edge insertions or deletions. The total number of edges participating in path deletions is upper bounded by $O(n')$. Each block split either takes only a constant number of edge insertions or deletions, or makes a non-trivial partition of the adjacent edges. In the latter case, we still do a constant number of edge insertions and deletions, followed by one edge move (deletion and insertion) for each edge that ends up in a non-largest set in the partition. Each edge is moved in this way $O(\log n')$ times, so the total number of update operations done on the fully dynamic structure is $O(|R'|\log n')$. Once an update to R' has been simulated in the fully dynamic structure, we can use queries in that structure to find any new cutvertices that we need to update $\mathrm{BC}(R')$. If the update in R' was a path deletion, then the corresponding update to $\mathrm{BC}(R')$ is either a path deletion, or a sequence of block splits. Each of these block splits can be found using the cutvertices given by the fully dynamic structure: Do a parallel search from both endpoints, and use the nearest cutvertex-query from Lemma 4 to guide the search and to know when a whole block has been found. If the update in R' was a block split, this will either do nothing in $\mathrm{BC}(R')$ or cause a single block split. If the update in R' is a pseudoblock contraction, the corresponding update to $\mathrm{BC}(R')$ is at most one edge deletion (because the leaf corresponding to the cutvertex disappears), at most one pseudoblock contraction (corresponding to the same pseudoblock contraction), or nothing happens (because the pseudoblocks were disposable). ◀

The point is that we will be using this with $|R'| = O\big(n/\big(t(n)\log n\big)\big)$, where n is the number of vertices in R, which means the total time used on R is $O(n) \subseteq O(|R|)$.

▶ **Lemma 16.** *The data structure of Lemma 15 can be extended to also report the explicit changes needed to* $\overline{\mathrm{BC}}(R', \partial R)$ *in the same asymptotic initialisation and update time. Any sequence of* $O(|R'|)$ *updates to* R' *cause* $O(|\partial R|)$ *updates in* $\overline{\mathrm{BC}}(R', \partial R)$*.*

Proof. For each change to $\mathrm{BC}(R')$, we can update $\overline{\mathrm{BC}}(R', \partial R)$ accordingly. This essentially consists of replaying the same change as in $\mathrm{BC}(R')$, followed by at most two pseudoblock contractions; at most one in each end of the path it possibly unfolds to. Note however, that some operations will end up having no effect on the structure of $\overline{\mathrm{BC}}(R', \partial R)$. For example a trivial block split followed by a pseudoblock contraction will change only which cutvertex

[7] By *explicit maintenance* is meant that each rooted BC-tree is maintained such that finding the parent of a vertex or block takes constant time.

separates the pseudoblock from the block. In this case, rather than doing a split and a contract, we simply update the identity of the cutvertex. With this optimization, the total number of block splits is upper bounded by $O(|\partial R|)$, and so is the number of edges and hence the number of possible path deletions and pseudoblock contractions. ◀

It immediately follows that we are able to efficiently maintain the patchwork graph, by combining the lemma above with the definition of the patchwork graph, $G_{\mathcal{R}} = \bigcup_{R\in\mathcal{R}} \overline{\mathrm{BC}}(R, \partial R)$.

▶ **Lemma 17.** *Given a graph G, and a strict (r, s)-division $\mathcal{R}$ of G, if we can explicitly maintain $\overline{\mathrm{BC}}(R, \partial R)$ for each $R \in \mathcal{R}$ in amortized constant time per update after $O(|R|)$ preprocessing, then we can explicitly maintain $G_{\mathcal{R}}$ in amortized constant time per update after $O(|G|)$ preprocessing. Furthermore, any sequence of $O(|G|)$ updates in G cause $O(|G_{\mathcal{R}}|)$ updates in $G_{\mathcal{R}}$.*

Proof. Let G have n vertices and m edges. The first part follows trivially from $\sum_{R\in\mathcal{R}}|R| \in O(n/r)O(r) + m = O(n + m)$. Each block split in $G_{\mathcal{R}}$ either reduces the degree of some block, or adds a pseudoblock. Since we do not add another pseudoblock when there already is one in a given direction, the maximum total number of splits an initial block vertex v can cause is $O(d(v))$. Thus the maximum number of splits is $\sum_{v\in G_{\mathcal{R}}} O(d(v)) = O(|G_{\mathcal{R}}|)$, and so is the maximum number of edges and hence the number of possible path deletions and pseudoblock contractions. ◀

In order to use Lemma 12 to answer biconnected queries, we need to store some auxiliary information: for each pseudoblock, store its neighbours, and for each disposable vertex, store its nearest represented vertex. Thus, these need to be updated as the graph undergoes dynamic updates.

path deletion When a path from x to y is deleted, all vertices represented by internal nodes on the path become disposable. For each such vertex v, its nearest represented vertex becomes either x or y. Furthermore, each vertex u who had v as its nearest represented vertex, now changes its nearest represented vertex to $\mathrm{nr}(u) = \mathrm{nr}(v)$. Thus, the set of vertices having x (or y) as a representative, is now the union of: vertices on the path, vertices represented by blocks or pseudo-blocks on the path, and the sets that these vertices used to represent. These sets of vertices that have the same representative can be maintained via union find in $O(n \log n)$ total merge-time and $O(1)$ worst-case find-time, using the weighted quick-find algorithm [1]. The endpoints x and y may change status from being represented by themselves to being represented by a block or pseudoblock.

block split A block is never the neighbour of a pseudoblock, nor is it the nearest represented *vertex*, so block splits do not give cause to changes in neighbours and representatives.

pseudoblock contraction does not give rise to changes in the nearest represented vertex - the vertices that were previously represented by a node that is involved in the contraction, are still represented, but now they are represented by the resulting pseudoblock. The set of vertices represented by the resulting pseudoblock is the union of vertices represented by nodes along the contracted path, again, this is done via union-find. Finally, the resulting pseudoblock is updated to remember its two neighbours.

We are now ready to prove:

▶ **Theorem 18** (first part of Theorem 1). *There exists a data structure that given a graph G with n vertices and m edges, and given a suitable pair of r-divisions, preprocesses G in $O(m + n)$ time and handles any series of edge-deletions in $O(m)$ total time while answering queries to pairwise biconnectivity in $O(1)$ time.*

Proof. Given a fine r-division $\mathcal{R}$ of G, build the BC-forest and compressed BC-forest for each region, and build the patchwork graph $G_{\mathcal{R}}$. Given the coarse division $\mathcal{A}$, and given the patchwork graph for $\mathcal{R}$, build the BC-forest and the compressed BC-forest for each region of the patchwork graph, and build the patchwork graph $G_{\mathcal{A}}$. Finally, build the BC-forest for $G_{\mathcal{A}}$. The construction time is linear, due to [38] and Observation 9.

Deletions are handled bottom up: updating the regions of $\mathcal{R}$, then those of $G_{\mathcal{R}}$ induced by $\mathcal{A}$, and then $G_{\mathcal{A}}$. The total time for deletions is linear, due to Lemmata 15, 16, and 17.

In detail: Since $r_2 = O(\text{poly}(\log\log n))$, we can afford to precompute and store a table of all simple graphs on r_2 vertices with s_2 boundary vertices, and how their BC-trees and compressed BC-trees change under any possible edge deletion. Using such a table, the region R in $\mathcal{R}$ containing the deleted edge can be updated in constant time.

The updates to $\overline{\text{BC}}(R, \partial R)$ may cause some updates to the patchwork graph $A_{\mathcal{R}\cap A}$ for the region $A \in \mathcal{A}$ containing the deleted edge. By Lemma 17, we can find these in amortized constant time per edge deletion in A, and there are at most $|A_{\mathcal{R}\cap A}|$ of them. Since

$$|A_{\mathcal{R}\cap A}| \leq \sum_{R\in\mathcal{R}\cap A} |\partial R| \leq s_2 \cdot O\left(\frac{r_1}{r_2}\right) \in O\left(r_1 \frac{s_2}{r_2}\right) \quad\text{and}\quad \frac{r_2}{s_2} = \Omega(t(r_1)\log r_1)$$

we have $|A_{\mathcal{R}\cap A}| \in O(\frac{r_1}{t(r_1)\log r_1})$. By Lemma 15 and 16 we can therefore explicitly maintain $\text{BC}(A_{\mathcal{R}\cap A})$ and $\overline{\text{BC}}(A_{\mathcal{R}\cap A}, \partial A)$ in amortized constant time per edge deletion in A.

The updates to $\overline{\text{BC}}(A_{\mathcal{R}\cap A}, \partial A)$ again trigger some number of updates to $G_{\mathcal{R}}$. By Lemma 17, we can find these in amortized constant time per edge deletion in G, and there are at most $|G_{\mathcal{R}}|$ of them. Since

$$|G_{\mathcal{R}}| \leq \sum_{A\in\mathcal{A}} |\partial A| \leq s_1 \cdot O\left(\frac{n}{r_1}\right) \in O\left(n \frac{s_1}{r_1}\right) \quad\text{and}\quad \frac{r_1}{s_1} = \Omega(t(n)\log n)$$

we have $|G_{\mathcal{R}}| \in O(\frac{n}{t(n)\log n})$. By Lemma 15 we can therefore explicitly maintain $\text{BC}(G_{\mathcal{R}})$ in amortized constant time per edge deletion in G.

To handle biconnected-queries, perform the $O(1)$ queries indicated by Lemma 12. ◀

6 Nearest cutvertex in $O(1)$ worst-case time

We have now shown how we handle queries to biconnectivity in a decremental graph subject to deletions. To answer nearest cutvertex queries, we need more structure. We need to augment our explicit representation of the dynamic BC-tree subject to block-splits so that it answers nearest cutvertex queries (subsection 6.1), and we need to show that we need only a constant number of queries in the patchwork graph together with a constant number of queries in regions, to answer nearest-cutvertex in the graph.

6.1 Navigating a dynamic BC-tree

If the vertices u and v are connected but not biconnected, and we have a BC-tree over the component containing them, the nearest cutvertex to u will be the second internal node on the unique BC-tree-path from u to v. So, in order to answer nearest cutvertex queries, it is enough to answer first-on-path queries on a tree (since second-on-path can be found using two first-on-path queries).

▶ **Lemma 19.** *There is a data structure for representing a dynamic BC-forest that can be initialized on a forest with n nodes and support any sequence of $O(n)$ path-deletions, block-splits, and pseudoblock-contractions, in $O(n\log n)$ total time, while answering* connected *and* first-on-path *queries in worst case constant time.*

Proof. We use the *characteristic ancestor* and *tree connectivity* data structures defined in the full version as a base (preliminary version available as [26]). These both work on rooted trees with black and white nodes, where a white node can be *split* into two white nodes of lower degree, and we can *contract* the endpoints of any edge regardless of color to form a new black node.

First, observe that we can combine these into a single structure, supporting both split, contract, and delete operations and both first-on-path and connected queries. This is because first-on-path(u, v) is only valid when u and v are connected, and the results of valid queries are therefore not affected by edge deletions. So we can maintain the two structures in parallel, and simply ignore deletions in the first-on-path structure, and let each structure answer the query it is designed for.

Second, observe that:

- Each path-deletion can be simulated using contractions and an edge deletion.
- Each block-split can be implemented as two node splits and an edge contraction.
- Each pseudoblock-contraction can be implemented as two edge contractions.

And note that if we color each vertex black and each block white (with pseudoblocks being either black or white depending on their history), then these operations respect the color requirements for our data structures.

Since we do only $O(n)$ operations, and we start with n black nodes, the total time for all updates is $O(n \log n)$. ◀

It follows as a corollary that we can answer nearest cutvertex queries given an explicit representation of the BC-forest:

▶ **Corollary 20.** *Given a dynamic BC-tree over a connected n-vertex graph, we can answer biconnected and nearest cutvertex queries in $O(1)$ time, spending an additional $O(n \log n)$ time on any sequence of updates.*

Proof. Given a pair of connected and different vertices u, v, let w be the second-on-path vertex found by querying first-on-path(first-on-path$(u, v), v)$. If $w = v$, the vertices are biconnected. Otherwise, w is the nearest cutvertex separating u from v. ◀

6.2 The patchwork graph

In the following, recall that each disposable vertex v knows its nearest represented vertex $\mathrm{nr}(v)$, and each pseudoblock knows its two neighbours.

▶ **Lemma 21.** *If u, v are connected and not biconnected, then the nearest cutvertex separating u from v can be determined by at most one nearest cutvertex-query and at most one biconnected query in $G_{\mathcal{R}}$ followed by at most one nearest cutvertex-query and at most one biconnected query within a region.*

Proof. If u and v are not both non-boundary vertices, and they are connected within the region R containing both, then the nearest cutvertex within R is the nearest cutvertex in G.

Otherwise, if u is disposable, then it knows its nearest represented vertex $\mathrm{nr}(u)$. If u and $\mathrm{nr}(u)$ are biconnected, then $\mathrm{nr}(u)$ is the nearest cutvertex separating u from v in G. Otherwise, the nearest cutvertex separating u from $\mathrm{nr}(u)$ in their region R is also the nearest cutvertex separating u from v in G.

If u is represented but v is not represented, then v knows its closest represented vertex $\mathrm{nr}(v)$ within its region. If $\overline{\mathrm{B}}(\mathrm{nr}(v))$ is biconnected with $\overline{\mathrm{B}}(u)$, then $\mathrm{nr}(v)$ is the answer, otherwise, $\mathrm{nr}(v)$ is used in place of v in the following.

For the remaining cases, u and v are both represented, and their representatives are different. If the nearest cutvertex query between $\overline{\mathrm{B}}(u)$ and $\overline{\mathrm{B}}(v)$ in $G_{\mathcal{R}}$ returns a neighbour b of the pseudo-block that either is $\overline{\mathrm{B}}(u)$ or is a neighbour of $\overline{\mathrm{B}}(u)$, then querying nearest cutvertex between u and b in the region of the pseudo-block will return the nearest cutvertex between u and v in G. Note here, that a pseudo-block is only present in one region, and even if u is a boundary vertex that appears in several regions, the pseudo-block knows the identity of both its endpoints within the region.

Finally, in all other cases, the nearest cutvertex separating $\overline{\mathrm{B}}(u)$ and $\overline{\mathrm{B}}(v)$ in $G_{\mathcal{R}}$ is the nearest cutvertex separating u and v in G. ◀

▶ **Theorem 22** (Second part of Theorem 1). *The data structure in Theorem 18 can be augmented to support queries to nearest cutvertex in $O(1)$ worst-case time, while handling any series of edge-deletions in $O(n + m)$ total time.*

Proof. For the patchwork graph of G and for the patchwork graphs of each region, maintain their dynamic BC-forest as indicated in Lemma 19. For the regions of the fine r-division, that is, those of polylog log-size, maintain an explicit table over the answer to nearest cutvertex queries.

Due to Corollary 20, the maintenance of explicit forests of BC-trees over the patchwork graph of G is done in $O(n' \log n')$ total time for $n' = O(n/\log n)$, thus, $O(n)$ total time, while handling intermixed nearest cutvertex-queries in $O(1)$ worst-case time. Same goes for the explicit maintenance of BC-forests of the patchwork graphs in the regions of the coarse r-division.

Finally, to handle nearest-cutvertex(u, v)-queries, perform the $O(1)$ queries indicated by Lemma 21: each of the $O(1)$ queries in the regions of the coarse r-division give rise to $O(1)$ look-ups in the regions of the fine r-division. Thus, the total query-time is constant. ◀

7 Conclusion and implications

We have given a somewhat technical theorem stating that if a graph has suitable r-divisions, there is an efficient data structure for decremental biconnectivity. For minor free graphs, we promised not only that they admit suitable r-divisions, but that such r-divisions can be computed in linear time in the size of the graph (regarding the size of the excluded minor as a constant). In the full version, we prove the following consequence to Theorem 1:

▶ **Corollary 23.** *There exists a data structure that, given a graph G with n vertices and m edges, and given a suitable pair of r-divisions, preprocesses G in $O(m + n)$ time and handles any series of edge-deletions in $O(m)$ total time while answering connectivity queries, 2-edge connectivity queries, and biconnectivity queries in $O(1)$ time.*

Using [37, Lemma 2] and [40, Lemma 3.4], we also show Lemma 2 which is used in the proof of the following theorem, which in combination with Theorem 1 gives Corollary 3.

▶ **Theorem 24.** *Given a graph G with n vertices that does not have a K_ℓ-minor, and any $t(n) \in O(\mathrm{poly}(\log n))$ we can compute a t-suitable pair of r-divisions in $O(n)$ time.*

Finally, since the total number of edges in a minor-free graph is $O(n)$, the data structure above has the optimal amortized update time for edge deletions and vertex deletions, both. The question of whether our data structure generally admits vertex deletions in $O(n)$ total time remains open.

References

1 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1974.

2 Lyudmil Aleksandrov and Hristo Djidjev. Linear algorithms for partitioning embedded graphs of boundedgenus. *Siam Journal on Discrete Mathematics - SIAMDM*, 9:129–150, February 1996. doi:10.1137/S0895480194272183.

3 Lars Arge, Freek van Walderveen, and Norbert Zeh. Multiway simple cycle separators and i/o-efficient algorithms for planar graphs. In *Proceedings of the Twenty-fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, pages 901–918, Philadelphia, PA, USA, 2013. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=2627817.2627882.

4 David Eppstein. Dynamic generators of topologically embedded graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 599–608, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=644108.644208.

5 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *Journal of the ACM*, 44(5):669–696, September 1997. doi:10.1145/265910.265914.

6 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. Separator-based sparsification ii: Edge and vertex connectivity. *SIAM Journal on Computing*, 28(1):341–381, February 1999. doi:10.1137/S0097539794269072.

7 David Eppstein, Giuseppe F. Italiano, Roberto Tamassia, Robert E. Tarjan, Jeffery R. Westbrook, and Moti Yung. Maintenance of a minimum spanning forest in a dynamic planar graph. *Journal of Algorithms*, 13(1):33–54, March 1992. Special issue for 1st SODA.

8 Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM Journal on Computing*, 14(4):781–798, 1985. doi:10.1137/0214055.

9 Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16(6):1004–1022, December 1987. doi:10.1137/0216064.

10 Greg N. Frederickson. Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees. *SIAM Journal on Computing*, 26(2):484–538, 1997. doi:10.1137/S0097539792226825.

11 Michael L. Fredman and Michael E. Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, STOC '89, pages 345–354, New York, NY, USA, 1989. ACM. doi:10.1145/73007.73040.

12 Zvi Galil, Giuseppe F. Italiano, and Neil Sarnak. Fully dynamic planarity testing with applications. *Journal of the ACM*, 46(1):28–91, January 1999. doi:10.1145/300515.300517.

13 Dora Giammarresi and Giuseppe F. Italiano. Decremental 2- and 3-connectivity on planar graphs. *Algorithmica*, 16(3):263–287, 1996. doi:10.1007/BF01955676.

14 Michael T. Goodrich. Planar separators and parallel polygon triangulation. *Journal of Computer and System Sciences*, 51(3):374–389, 1995. doi:10.1006/jcss.1995.1076.

15 Jens Gustedt. Efficient union-find for planar graphs and other sparse graph classes. *Theoretical Computer Science*, 203(1):123–141, 1998. doi:10.1016/S0304-3975(97)00291-0.

16 Frank Harary. *Graph Theory*. Addison-Wesley Series in Mathematics. Addison Wesley, 1969.

17 Monika R Henzinger, Philip Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, 1997. doi:10.1006/jcss.1997.1493.

18 Monika R. Henzinger and Han La Poutré. Certificates and fast algorithms for biconnectivity in fully-dynamic graphs. In Paul Spirakis, editor, *Algorithms — ESA '95*, pages 171–184, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

19 Monika Rauch Henzinger and Valerie King. Fully dynamic 2-edge connectivity algorithm in polylogarithmic time per operation, 1997.

20 Monika Rauch Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *Journal of the ACM*, 46(4):502–516, 1999. Announced at STOC '95. doi:10.1145/320211.320215.

21 Monika Rauch Henzinger and Mikkel Thorup. Sampling to provide or to bound: With applications to fully dynamic graph algorithms. *Random Struct. Algorithms*, 11(4):369–379, 1997. doi:10.1002/(SICI)1098-2418(199712)11:4<369::AID-RSA5>3.0.CO;2-X.

22 John Hershberger, Monika Rauch, and Subhash Suri. Data structures for two-edge connectivity in planar graphs. *Theoretical Computer Science*, 130(1):139–161, 1994. doi:10.1016/0304-3975(94)90156-2.

23 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*, 48(4):723–760, July 2001. doi:10.1145/502090.502095.

24 Jacob Holm, Giuseppe F. Italiano, Adam Karczmarz, Jakub Lacki, and Eva Rotenberg. Decremental SPQR-trees for Planar Graphs. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 46:1–46:16, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ESA.2018.46.

25 Jacob Holm, Giuseppe F Italiano, Adam Karczmarz, Jakub Lacki, Eva Rotenberg, and Piotr Sankowski. Contracting a planar graph efficiently. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 87. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

26 Jacob Holm and Eva Rotenberg. Good r-divisions imply optimal amortised decremental biconnectivity. *CoRR*, abs/1808.02568, 2018. arXiv:1808.02568.

27 Jacob Holm, Eva Rotenberg, and Mikkel Thorup. Dynamic bridge-finding in $\widetilde{O}(\log^2 n)$ amortized time. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 35–52, 2018. doi:10.1137/1.9781611975031.3.

28 Shang-En Huang, Dawei Huang, Tsvi Kopelowitz, and Seth Pettie. Fully dynamic connectivity in $O(\log n (\log\log n)^2)$ amortized expected time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 510–520, 2017. doi:10.1137/1.9781611974782.32.

29 Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, pages 1131–1142, Philadelphia, PA, USA, 2013. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=2627817.2627898.

30 Casper Kejlberg-Rasmussen, Tsvi Kopelowitz, Seth Pettie, and Mikkel Thorup. Faster Worst Case Deterministic Dynamic Connectivity. 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 53:1–53:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ESA.2016.53.

31 Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 505–514, New York, NY, USA, 2013. ACM. doi:10.1145/2488608.2488672.

32 Richard J. Lipton and Robert E. Tarjan. A Separator Theorem for Planar Graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

33 Jakub Łącki and Piotr Sankowski. Min-cuts and shortest cycles in planar graphs in $O(n \log\log n)$ time. In *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, pages 155–166, 2011. doi:10.1007/978-3-642-23719-5_14.

34 Jakub Łącki and Piotr Sankowski. Optimal decremental connectivity in planar graphs. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, pages 608–621, 2015. doi:10.4230/LIPIcs.STACS.2015.608.

35 Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *Proceedings of the 58th Annual Symposium on Foundations of Computer Science, FOCS 2017*, 2017.

36 Mihai Pătraşcu and Erik D Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006.

37 Bruce Reed and David R. Wood. Fast separation in a graph with an excluded minor. In Stefan Felsner, editor, *2005 European Conference on Combinatorics, Graph Theory and Applications (EuroComb '05)*, volume DMTCS Proceedings vol. AE, European Conference on Combinatorics, Graph Theory and Applications (EuroComb '05) of *DMTCS Proceedings*, pages 45–50, Berlin, Germany, 2005. Discrete Mathematics and Theoretical Computer Science. URL: https://hal.inria.fr/hal-01184376.

38 Robert E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. doi:10.1137/0201010.

39 Robert E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, April 1975. doi:10.1145/321879.321884.

40 Siamak Tazari and Matthias Müller-Hannemann. Shortest paths in linear time on minor-closed graph classes, with an application to steiner tree approximation. *Discrete Applied Mathematics*, 157(4):673–684, 2009. doi:10.1016/j.dam.2008.08.002.

41 Mikkel Thorup. Decremental dynamic connectivity. In *SODA '97*, pages 305–313. SIAM, 1997.

42 Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 343–350, New York, NY, USA, 2000. ACM. doi:10.1145/335305.335345.

43 Christian Wulff-Nilsen. Separator theorems for minor-free and shallow minor-free graphs with applications. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 37–46, 2011. doi:10.1109/FOCS.2011.15.

44 Christian Wulff-Nilsen. Faster deterministic fully-dynamic graph connectivity. In *Encyclopedia of Algorithms*, pages 738–741. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

b-Coloring Parameterized by Clique-Width

Lars Jaffke
University of Bergen, Norway

Paloma T. Lima
University of Bergen, Norway

Daniel Lokshtanov
University of California Santa Barbara, CA, USA

Abstract

We provide a polynomial-time algorithm for b-Coloring on graphs of constant clique-width. This unifies and extends nearly all previously known polynomial-time results on graph classes, and answers open questions posed by Campos and Silva [Algorithmica, 2018] and Bonomo et al. [Graphs Combin., 2009]. This constitutes the first result concerning structural parameterizations of this problem. We show that the problem is FPT when parameterized by the vertex cover number on general graphs, and on chordal graphs when parameterized by the number of colors. Additionally, we observe that our algorithm for graphs of bounded clique-width can be adapted to solve the Fall Coloring problem within the same runtime bound. The running times of the clique-width based algorithms for b-Coloring and Fall Coloring are tight under the Exponential Time Hypothesis.

2012 ACM Subject Classification Mathematics of computing → Graph coloring

Keywords and phrases b-Coloring, clique-width, vertex cover, structural parameterization

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.43

Related Version *Full Version*: `https://arxiv.org/abs/2003.04254`

1 Introduction

This paper settles open questions regarding the complexity of the b-Coloring problem on graph classes and initiates the study of its structural parameterizations. A *b-coloring* of a graph G with k colors is a partition of the vertices of G into k independent sets such that each of them contains a vertex that has a neighbor in all of the remaining ones. The *b-chromatic number* of G, denoted by $\chi_b(G)$, is the maximum integer k such that G admits a b-coloring with k colors. This notion was introduced by Irving and Manlove [29] to describe the behavior of the following color-suppressing heuristic for the Graph Coloring problem. We start with some proper coloring of the input graph G and try to iteratively suppress one of its colors. That is, for a given color c, we consider each vertex v of color c, and check if there is another color $c' \neq c$ available that does not appear in its neighborhood. If so, we assign vertex v the color c', observing that the coloring remains proper, and repeat this process for the remaining vertices of color c. If successful, we remove the color c from all vertices of G and decrease the number of colors by one. Once no color can be supressed by this procedure, the coloring at hand is a b-coloring of G, and in the worst case, this heuristic produces a coloring with $\chi_b(G)$ many colors.

Since then, the b-Coloring and b-Chromatic Number problems which given a graph G and an integer k ask whether G has a b-coloring with k colors and whether $\chi_b(G) \geq k$, respectively, have received considerable attention in the algorithms and complexity communities.[1] While these problems have been shown to be NP-complete in the general

[1] We would like to remark that the b-Coloring and b-Chromatic Number problems are not as closely related as the Graph Coloring and Chromatic Number problems: If a graph G has a b-coloring with k colors, then $\chi_b(G) \geq k$, but $\chi_b(G) \geq k$ does not imply the existence of a b-coloring with k colors.

case [29], as well as on bipartite graphs [32], co-bipartite graphs [6], chordal graphs [24], and line graphs [7], a lot of effort has been put into devising polynomial-time algorithms for these problems in various other classes of graphs. These include trees [29], claw-free block graphs [10], tree-cographs [6], and graphs with few P_4s, such as cographs and P_4-sparse graphs [5], P_4-tidy graphs [45], and $(q, q-4)$-graphs for constant q [9]. A common property shared by these graph classes is that they all have bounded *clique-width.*[2]

The main contribution of this work is an algorithm that solves b-Coloring (and b-Chromatic Number) in polynomial time on graphs of constant clique-width. Besides unifying the above mentioned polynomial-time cases, this extends the tractability landscape of these problems to larger graph classes, and answers two open problems stated in the literature.

Over a decade ago, Bonomo et al. [5] asked whether their polynomial-time result for P_4-sparse graphs can be extended to distance-hereditary graphs. Havet et al. [24] answered the question negatively by providing an NP-completeness proof for chordal distance-hereditary graphs. We observe, however, that their proof has a flaw and while it does prove the claimed statement for chordal graphs, it unfortunately fails to do so for distance-hereditary graphs. Our polynomial-time algorithm for graphs of bounded clique-width in fact provides a positive answer to Bonomo et al.'s question, as distance-hereditary graphs have clique-width at most 3 [23]. In recent years, even subclasses of distance-hereditary graphs have received significant attention, for instance in the work of Campos and Silva [10]: they provide a polynomial-time algorithm for claw-free block graphs, and ask whether this result can be generalized to block graphs. Our algorithm provides a positive answer to this question as well. Moreover, it extends the known algorithm for $(q, q-4)$-graphs [9] (for constant q) to all (q,t)-graphs for constants q and t with $q \geq 4$, $t \geq 0$, and either $q \leq 6$ and $t \leq q-4$, or $q \geq 7$ and $t \leq q-3$, by a theorem to due Makowsky and Rotics [36].

Our algorithm runs in time $n^{2^{\mathcal{O}(w)}}$, where n denotes the number of vertices of the input graph which is given together with a clique-width w-expression. As consequences of results due to Fomin et al. [20] and Fomin et al. [21], we observe that b-Coloring parameterized by clique-width is W[1]-hard, and that the exponential dependence on w in the degree of the polynomial cannot be avoided unless the Exponential Time Hypothesis (ETH) fails. Concretely, an algorithm running in time $n^{2^{o(w)}}$ would refute ETH.

From the point of view of parameterized complexity, Panolan et al. [38] showed that b-Chromatic Number parameterized by the number of colors is W[1]-hard. However, this problem may even be harder, since so far no XP-algorithm is known. Recently, Aboulker et al. [1] showed that the more restrictive b-Chromatic Core problem parameterized by the number of colors (which has a brute-force XP-algorithm, see e.g. [18]) remains W[1]-hard.

It is therefore natural to ask which additional restrictions can be imposed to obtain parameterized tractability results. For instance, an open problem posed by Sampaio [41] (see also [43]) asks whether b-Coloring parameterized by the number of colors is FPT on chordal graphs. We answer this question in the affirmative, via Courcelle's Theorem [11] for bounded treewidth graphs. Other restricted cases that have been considered in the literature target specific numbers of colors that depend on the input graph. The Dual b-Coloring problem, which asks if an input n-vertex graph has a b-coloring with $n-k$ colors, is FPT parameterized by k [25]. Moreover, deciding if a graph G has a b-coloring with $k = \Delta(G) + 1$

[2] To the best of our knowledge, the only polynomial-time result for graphs of unbounded clique-width so far concerns graphs of large girth. In particular, Campos et al. [8] showed that b-Chromatic Number is polyomial-time solvable on graphs of girth at least 7.

colors, which is an upper bound on $\chi_b(G)$, is FPT parameterized by k [38, 41], while the case $k = \Delta(G)$ is XP and for every fixed $p \geq 1$, the case $k = \Delta(G) - p$ is NP-complete for $k = 3$ [30].

Another novelty aspect of our XP-algorithm parameterized by clique-width is that it is the first result about *structural parameterizations* of the b-Coloring and b-Chromatic Number problems. In all previously known polynomial-time cases the algorithms only work if the input graph has some prescribed structure. Our algorithm works on all graphs, albeit with a prohibitively slow runtime on graphs of large clique-width. In this vein, we round off our work with an FPT-result for another lead player among structural parameterizations, the *vertex cover number* of a graph; a parameter often referred to as the *Drosophila* of parameterized complexity.

Fall Coloring. A *fall coloring* is a special type of b-coloring where *every* vertex needs to have at least one neighbor in all color classes except its own. In other words, it is a partition of the vertex set of a graph into independent dominating sets. As a standalone notion, fall coloring has been introduced by Dunbar et al. [17]. However, since the corresponding Fall Coloring problem falls in the category of locally checkable vertex partitioning problems, it has been shown in earlier work of Telle and Proskurowski [44] to be FPT parameterized by the treewidth of the input graph, and by Heggernes and Telle [26] to be NP-complete for fixed number of colors. Fall Coloring remains hard further restricted to bipartite [33, 34, 42], chordal [42], or planar [34] graphs. On the other hand, even with unbounded number of colors, it is known to be solvable in polynomial time on strongly chordal graphs [35, 22], threshold graphs and split graphs [37]. In all of these cases, one simply checks whether the chromatic number of the input graph is equal to its minimum degree plus one. To the best of our knowledge, these are the only known polynomial-time cases. We adapt our algorithm for b-Coloring on graphs of bounded clique-width to solve Fall Coloring, and therefore show that the latter problem is as well solvable in time $n^{2^{\mathcal{O}(w)}}$, where w denotes the clique-width of a given decomposition of the input graph. By a simple reduction, we show that Fall Coloring is also W[1]-hard in this parameterization and that an $n^{2^{o(w)}}$-time algorithm for it would refute ETH.

Vertex Coloring Problems Parameterized by Clique-Width. We briefly touch on differences in the complexities of vertex coloring problems of graphs when parameterized by clique-width. While the standard Graph Coloring problem, asking for a proper coloring of the input graph, is XP-time solvable parameterized by clique-width [19, 46], some of its generalizations are NP-complete on graphs of constant clique-width. In the List Coloring problem we are given a graph G and for each of its vertices v a list $L(v)$ of colors, and the question is whether G has a proper coloring such that each vertex is assigned a color from its list. This problem is NP-complete on the (not disjoint) union of two complete graphs [31], and such graphs clearly have constant clique-width. In the related Precoloring Extension problem, we are given a graph, some of whose vertices already received a color, and the question is whether this coloring can be extended to a proper coloring of the entire graph. The following standard reduction from List Coloring, starting with a graph that is the union of two complete graphs, shows that this variant is NP-complete on graphs of constant clique-width as well. Take the graph G together with the lists $L(\cdot)$, and construct a graph H by adding to G, for each vertex $v \in V(G)$ and each color $c \notin L(v)$, a new vertex x_v^c which is adjacent only to v and assigned color c. It is not difficult to see that this precoloring of H can be extended to the remainder of its vertices if and only if G has a list coloring using the lists $L(\cdot)$. Moreover, adding pendant vertices to a graph does not increase its clique-width.

Belmonte et al. [3] recently showed that the Grundy Coloring problem, which asks for a linear order of the vertices that maximizes the number of colors used by the greedy coloring heuristic, is NP-complete on graphs of constant clique-width. This nicely contrasts our XP-algorithm for b-Coloring, since both the b-Coloring and the Grundy Coloring problems are rooted in the theoretical analysis of graph coloring heuristics.

Sketch of the algorithm. Let us discuss how we obtain our XP-algorithm parameterized by clique-width. First, we consider a branch decomposition of the input graph G of bounded *module-width* w which is equivalent to clique-width and has the following property. At each node t of the branch decomposition we have a subgraph G_t of G whose vertex set can be partitioned into at most w equivalence classes with respect to their neighborhood outside of G_t. For the purpose of our dynamic programming algorithm, it suffices to describe colorings by the way each of their color classes interacts with these equivalence classes. In the Graph Coloring problem, it is enough to describe a color class according to its intersection with the equivalence classes of G_t alone [19, 46] (see also [21]). For the b-Coloring problem, we additionally have to ensure that eventually, each color class indeed has a b-vertex. The challenge is to do so without explicitly remembering which color classes a vertex has already seen in its neighborhood – this would result in prohibitively large tables. We overcome this difficulty by a symmetry breaking trick that instead stores, for each color class, a *demand* to the future neighbors of the equivalence classes which – if fulfilled – guarantees that the *other* color classes can have b-vertices in the end.

Due to space restrictions, proofs of statements marked "♣" as well as several discussions are deferred to the full version.

2 Preliminaries

We use standard terminology and assume the reader to be familiar with basic notions in graph theory and parameterized complexity and refer to the books [4, 15] and [14, 16], respectively, for introductions; or to the attached full version. To avoid confusion, we clarify some notation. All graphs considered here are simple and finite. For a graph G we denote by $V(G)$ and $E(G)$ the vertex set and edge set of G, respectively. For a set of vertices $S \subseteq V(G)$, the *subgraph of G induced by S* is $G[S]$. A graph is called *subcubic* if all its vertices have degree at most three. A graph G is *connected* if for all 2-partitions (X, Y) of $V(G)$ with $X \neq \emptyset$ and $Y \neq \emptyset$, there is a pair $x \in X$, $y \in Y$ such that $xy \in E(G)$. A *connected component* of a graph is a maximal connected subgraph. In a tree T, the vertices of degree one are called the *leaves* of T, denoted by $\mathrm{L}(T)$, and the vertices in $V(T) \setminus \mathrm{L}(T)$ are the *internal vertices* of T. The *length* of a path is the number of its edges. For a graph G and a pair of vertices $u, v \in V(G)$, we denote by $\mathrm{dist}_G(u, v)$ the length of the shortest path between u and v in G. A graph G is called *distance-hereditary* if for each connected induced subgraph H of G, and each pair of vertices $u, v \in V(H)$, $\mathrm{dist}_H(u, v) = \mathrm{dist}_G(u, v)$. A tree T is called a *caterpillar* if it contains a path $P \subseteq T$ such that all vertices in $V(T) \setminus V(P)$ are adjacent to a vertex in P.

Let Ω be a set and $\sim$ an equivalence relation over Ω. For an element $x \in \Omega$ the *equivalence class of x*, denoted by $[x]$, is the set $\{y \in \Omega \mid x \sim y\}$. We denote the set of all equivalence classes of $\sim$ by $\Omega/\sim$.

The *Exponential Time Hypothesis (ETH)* is the following conjecture about the 3-SAT problem, which given a boolean formula ϕ in conjunctive normal form with clauses of size at most three asks whether there is a truth assignment to its variables that lets ϕ evaluate to true.

▶ **Conjecture** (ETH [27, 28]). *There is no algorithm that solves each instance of 3-SAT on n variables in time $2^{o(n)}$.*

Clique-Width, branch decompositions, and module-width. We first define clique-width, introduced by Courcelle, Engelfriet, and Rozenberg [12], and then the equivalent measure of *module-width* that we will use in our algorithm. The reason why we choose module-width over clique-width is that at each node of the decomposition it captures information that is very useful for coloring problems:

We keep the definition of clique-width slightly informal and refer to [12, 13] for more details. Let G be a graph. The *clique-width* of G, denoted by $\mathsf{cw}(G)$, is the minimum number of labels $\{1, \ldots, k\}$ needed to obtain G using the following four operations: (1) Create a new graph consisting of a single vertex labeled i. (2) Take the disjoint union of two labeled graphs G_1 and G_2. (3) Add all edges between pairs of vertices of label i and label j. (4) Relabel every vertex labeled i to label j.

▶ **Definition 1** (Branch decomposition). *Let G be a graph. A* branch decomposition *of G is a pair $(T, \mathcal{L})$ of a subcubic tree T and a bijection $\mathcal{L} \colon V(G) \to \mathrm{L}(T)$. If T is a caterpillar, then $(T, \mathcal{L})$ is called* linear branch decomposition. *If T is rooted, then we call $(T, \mathcal{L})$ a* rooted branch decomposition. *In this case, for $t \in V(T)$, we denote by T_t the subtree of T rooted at t, and we define $V_t := \{v \in V(G) \mid \mathcal{L}(v) \in \mathrm{L}(T_t)\}$, $\overline{V_t} := V(G) \setminus V_t$, and $G_t := G[V_t]$.*

▶ **Definition 2** (Module-width, [39, 40]). *Let G be a graph, and $(T, \mathcal{L})$ be a rooted branch decomposition of G. For each $t \in V(T)$, let $\sim_t$ be the equivalence relation on V_t defined as: $\forall u, v \in V_t \colon u \sim_t v \Leftrightarrow N_G(u) \cap \overline{V_t} = N_G(v) \cap \overline{V_t}$. The* module-width *of $(T, \mathcal{L})$ is $\mathsf{mw}(T, \mathcal{L}) := \max_{t \in V(T)} |V_t/{\sim_t}|$. The* module-width of G*, denoted by* $\mathsf{mw}(G)$*, is the minimum module width over all rooted branch decompositions of G.*

Let $(T, \mathcal{L})$ be a rooted branch decomposition of a graph G and let $t \in V(T)$ be a node with children r and s. We now describe an operator associated with t that tells us how the graph G_t is formed from its subgraphs G_r and G_s, and how the equivalence classes of $\sim_t$ are formed from the equivalence classes of $\sim_r$ and $\sim_s$. Concretely, we associate with t a bipartite graph H_t on bipartition $(V_r/{\sim_r}, V_s/{\sim_s})$ such that:

1. $E(G_t) = E(G_r) \cup E(G_s) \cup F$, where $F = \{uv \mid u \in V_r, v \in V_s, \{[u], [v]\} \in E(H_t)\}$, and
2. there is a partition $\mathcal{P} = \{P_1, \ldots, P_h\}$ of $V(H_t)$ such that $V_t/{\sim_t} = \{Q_1, \ldots, Q_h\}$, where for $1 \le i \le h$, $Q_i = \bigcup_{Q \in P_i} Q$. For each $1 \le i \le h$, we call P_i the *bubble* of the resulting equivalence class $\bigcup_{Q \in P_i} Q$ of $\sim_t$.

As auxiliary structures, for $p \in \{r, s\}$, we let $\eta_p \colon V_p/{\sim_p} \to V_t/{\sim_t}$ be the map such that for all $Q_p \in V_p/{\sim_p}$, $Q_p \subseteq \eta_p(Q_p)$, i.e. $\eta_p(Q_p)$ is the equivalence class of $\sim_t$ whose bubble contains Q_p. We call (H_t, η_r, η_s) the *operator* of t.

▶ **Theorem 3** (Rao, Thm. 6.6 in [39]). *For any graph G, $\mathsf{mw}(G) \le \mathsf{cw}(G) \le 2 \cdot \mathsf{mw}(G)$, and given a decomposition of bounded clique-width, a decomposition of bounded module-width, and vice versa, can be constructed in time $\mathcal{O}(n^2)$, where $n = |V(G)|$.*

Colorings. Let G be a graph. An ordered partition $\mathcal{C} = (C_1, \ldots, C_k)$ of $V(G)$ is called a *coloring* of G (with k colors). (Observe that for $i \in \{1, \ldots, k\}$, C_i may be empty.) For $i \in \{1, \ldots, k\}$, we call C_i the *color class* i, and say that the vertices in C_i *have color* i. $\mathcal{C}$ is called *proper* if each C_i is an independent set in G. The *restriction* of a coloring $\mathcal{C} = (C_1, \ldots, C_k)$ to a vertex set $S \subseteq V(G)$, is $\mathcal{C}|_S := (C_1 \cap S, \ldots, C_k \cap S)$. In this case we

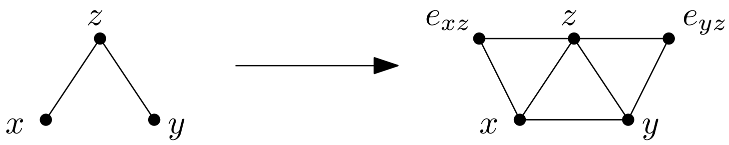


Figure 1 A gem created following the reduction in [24].

say conversely that $\mathcal{C}$ *extends* $\mathcal{C}|_S$. A proper coloring $(C_1, \ldots, C_k)$ is called a *b-coloring*, if for all $i \in \{1, \ldots, k\}$, there is a vertex $v_i \in C_i$, called *b-vertex of color i*, such that for all $j \in \{1, \ldots, k\} \setminus \{i\}$, $N_G(v_i) \cap C_j \neq \emptyset$.

b-Coloring

Input: Graph G, integer k
Question: Does G have a *b*-coloring with k colors?

Distance-hereditary graphs and chordal graphs. In their work on P_4-sparse graphs, Bonomo et al. [5] asked whether *b*-Coloring is polynomial-time solvable on the class of distance-hereditary graphs. Havet et al. [24] claimed to answer this question in the negative way, showing that *b*-Coloring is NP-complete on chordal distance-hereditary graphs. Their proof, however, contains a flaw and the graph constructed in their reduction, even though indeed chordal, fails to be distance-hereditary. In what follows, we briefly describe their reduction and argue that the graph constructed is not distance-hereditary. The reduction presented in [24] is from 3-Edge Coloring restricted to the class of 3-regular graphs. Given an instance G for 3-Edge Coloring with $V(G) = \{v_1, \ldots, v_n\}$, they construct a graph H as follows. The vertex set of H contains a copy of $V(G)$ plus one vertex associated with each edge of G. We denote by e_{xy} the vertex corresponding to the edge xy. The vertices of $V(G)$ form a clique in H, the vertices corresponding to edges form an independent set, and for each edge $xy \in E(G)$, the vertex e_{xy} is adjacent to the copy of x and y in H. The connected component of H induced by these vertices is therefore a split graph. Finally, they add three disjoint copies of $K_{1,n+2}$ to H. It is thus easy to see that H is a chordal graph. However, let xz and yz be two edges of G sharing one endpoint. Then the subgraph of H induced by $\{x, y, z, e_{xz}, e_{yz}\}$ is isomorphic to a gem (see Figure 1). As shown by Bandelt and Mulder [2], distance-hereditary graphs are gem-free graphs. This shows that the graph H is not a distance-hereditary graph.

Via monadic second order logic and Courcelle's Theorem [11], we can show the following result for chordal graphs.

▶ **Proposition 4** (♣)**.** *b-Coloring parameterized by k is* FPT *on chordal graphs.*

3 Parameterized by Clique-Width

In this section, we consider the *b*-coloring problem parameterized by the clique-width of the input graph. We will work with decompositions of bounded *module-width*, which is equivalent for our purposes, see Theorem 3.

The main contribution of this section is an algorithm that given a graph G on n vertices and one of its rooted branch decompositions of module-width w, and an integer k, decides whether G has a *b*-coloring with k colors in time $n^{2^{\mathcal{O}(w)}}$. Before we proceed, we observe that *b*-Coloring is W[1]-hard in this parameterization, and that the exponential dependence on w of the degree of the polynomial in the runtime is probably difficult to avoid.

▶ **Proposition 5** (♣). *The b-*Coloring *problem on graphs on n vertices parameterized by their module-width w is* W[1]-hard and cannot be solved in time $n^{2^{o(w)}}$, unless ETH fails. Moreover, the hardness holds even when a linear branch decomposition of width w is provided.

3.1 Outline of the Algorithm

Throughout the following, we are given a graph G and one of its rooted branch decompositions $(T, \mathcal{L})$ of module-width $w = \mathsf{mw}(T, \mathcal{L})$ and we want to find a b-coloring of G with k colors, if it exists. In particular, our algorithm will find a b-coloring $\mathcal{C}$ together with a set of *witness b-vertices*, containing precisely one b-vertex for each color class of $\mathcal{C}$, if it exists. This will be done via dynamic programming along T, and for each node $t \in V(T)$, the partial solutions associated with t are partial b-colorings of G_t.

▶ **Definition 6** (Partial b-Coloring). *Let G be a graph and $k \in \mathbb{N}$. For an induced subgraph H of G, a* partial b-coloring *of H is a pair $(\mathcal{C}, B)$ of a* proper coloring *$\mathcal{C} = (C_1, \ldots, C_k)$ of H and a subset $B \subseteq V(H)$ such that for all $i \in [k]$, $|C_i \cap B| \le 1$. We call the vertices in B the* partial b-vertices*.*

To obtain an efficient algorithm, we require a compact representation of the partial b-colorings of each subgraph G_t associated with a node $t \in V(T)$. To that end, we introduce the notion of a *t-signature* of a partial b-coloring. Two partial b-colorings with the same t-signature will be interchangeable for the sake of our algorithm, therefore the number of table entries at each node t will be bounded by the number of t-signatures.

Let $(\mathcal{C}, B)$ be a partial b-coloring of G_t. For $(\mathcal{C}, B)$ to be extended to a b-coloring $(\mathcal{C}', B')$ of the entire graph G, we have to ensure that two things happen for each color class $C \in \mathcal{C}$:

(I) The extension of C in $\mathcal{C}'$ is an independent set in G.

(II) There is a witness b-vertex in B' for the extension of C in $\mathcal{C}'$.

The t-signature has to represent a partial b-coloring faithfully enough so that we can keep track of all the ways in which the above two conditions can be satisfied for each of its color classes 'in the future'. At the same time, its definition has to enable us to significantly compress the information about partial b-colorings of G_t. This happens in the following way. We categorize color classes of partial b-colorings of G_t according to *t-types*. If two color classes C_1, C_2 of a partial b-coloring $(\mathcal{C}, B)$ have the same t-type, then the above two conditions can be satisfied for C_1 and C_2 by extensions of $(\mathcal{C}, B)$ in the exact same ways. This allows us to forget about the "names" of the color classes in a partial b-coloring, but instead to only remember for each t-type how many color classes with that type there are. This is precisely the information that is stored in a t-signature.

Now, if we can bound the number of t-types by some function of the module-width w, say $f(w)$, then the number of t-signatures is upper bounded by $k^{f(w)} \le n^{f(w)}$. (There are at most k colors, so in particular there are at most k colors with a given t-type.) This translates directly to an upper bound on the number of table entries in the dynamic programming algorithm, which, up to some constants in the degree of the polynomial, bounds the runtime of the resulting algorithm.

Let us discuss the information that goes into the definition of a t-type. Let C be a color class in a partial b-coloring $(\mathcal{C}, B)$ of G_t. To keep track of which vertices from $\overline{V_t}$ can be added to C without introducing a coloring conflict, it suffices to store which equivalence classes of $\sim_t$ have vertices in C,[3] since all vertices in a given equivalence class have the same neighbors in $\overline{V_t}$. This way we can ensure that condition (I) is satisfied.

[3] This is similar to the algorithm of Wanke for Graph Coloring on graphs of bounded NLC-width [46].

To verify if condition (II) is satisfied we have to store some information about the partial b-vertices. Naturally, we record whether or not B contains a partial b-vertex of C, but we need to store more information. Suppose that B contains the partial b-vertex v of C. In a straightforward approach, we would simply keep track of the color classes that already appear in the neighborhood of v. This way we could easily decide at which point during the execution of the algorithm, a partial b-vertex turns into a b-vertex. However, this results in prohibitively large table entries, since there are 2^{k-1} subsets of colors that we would have to consider, which for our purpose is no better than 2^n.

We overcome this issue with the following symmetry breaking trick: We do *not* record which color classes the partial b-vertex of C already sees/still needs to see. Instead, we record for which equivalence classes $Q \in V_t/{\sim_t}$ we need to add a *future neighbor of* Q, i.e. a vertex from $N(Q) \cap \overline{V_t}$, to C, such that the partial b-vertex from *some other color* C' sees color C in its neighborhood. More concretely, suppose that some equivalence class $Q \in V_t/{\sim_t}$ contains the partial b-vertex $w \in B$ of another color class $C' \neq C$, such that w has no neighbor of color C in V_t. For w to become a b-vertex of its color, color class C must be extended with a neighbor of w in the future, i.e. in $\overline{V_t}$. The neighborhood of w in $\overline{V_t}$ is precisely $N_G(Q) \cap \overline{V_t}$, therefore we can concisely model this situation as color class C requiring to contain a vertex among the future neighbors of Q. In this situation, we say that

color class C has demand to the future neighbors of Q.

The t-type records for each equivalence class Q of $\sim_t$, if a color class contains vertices of Q, or if it has demand to the future of Q, or none of the two. Note that if a color class both contains a vertex from Q and has demand to the future of Q, we already know that we can disregard the corresponding partial b-coloring: In the corresponding color class, we cannot add any future neighbors of Q without creating a coloring conflict, and if we do not add a future neighbor of Q, then there is some color class whose partial b-vertex will never become a b-vertex. Now, if we have a partial b-coloring in which every color class has a partial b-vertex, and all demands have been fulfilled, meaning that there is no color class that has demand to the future of some equivalence class of $\sim_t$, then we know that we actually have a b-coloring. Moreover, the number of t-types is $2^{\mathcal{O}(w)}$, so the resulting algorithm runs in time $n^{2^{\mathcal{O}(w)}}$.

3.2 t-Types and t-Signatures

In this section we introduce the basic concepts that we alluded to in the above description, namely the notion of a *t-type* and of a *t-signature*, where t is some node in the given branch decomposition. A t-type is meant to capture the necessary information of a color class in a partial b-coloring of G_t. However, we cannot give the definition of a t-type as a property of a vertex set alone: a color class C may have demand to the future of an equivalence class, which is because there is a partial b-vertex of *another* color $C' \neq C$ that has no neighbor of color C yet. Therefore, we first give the definition of a t-type abstractly, i.e. absent of any partial b-coloring or color class, and then define what it means for a color class to be of a certain t-type *within a partial b-coloring.* This is illustrated in Figure 2.

The t-type is a pair of a bit that is meant to tell us whether or not a coloring contains a partial b-vertex of that color, and a map that tells us for each equivalence class, whether there is a vertex of the color in the equivalence class (via the value cont), or if the color has demand to the future neighbors of the equivalence class (via the value dem), or none of the two (via the value none).

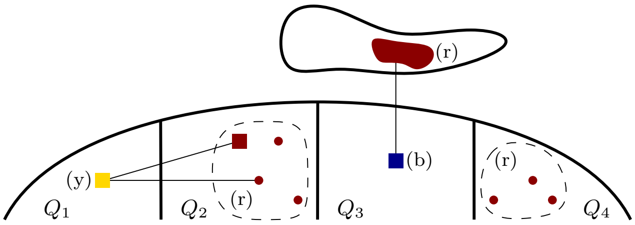


Figure 2 Illustration of the definition of a color class being of a certain t-*type* inside a partial b-coloring of G_t. The large square vertices are partial b-vertices for their color. The type of the red (r) color in the coloring is as follows. Since it has a b-vertex (the one in Q_2), we have that $\xi = 1$. Since Q_2 and Q_4 have red vertices, $\phi(Q_2) = \phi(Q_4) = \mathsf{cont}$. Q_1 and Q_3 do not have red vertices. Q_1 contains the b-vertex of color yellow (y), but this vertex already has a red neighbor. Therefore, $\phi(Q_1) = \mathsf{none}$. Finally, Q_3 has the b-vertex of color blue (b), and this vertex does not have a red neighbor yet. Therefore, there has to be a red vertex among the future neighbors of Q_3. Hence, $\phi(Q_3) = \mathsf{dem}$.

▶ **Definition 7** (t-Type). *Let G be a graph with rooted branch decomposition $(T, \mathcal{L})$ and let $t \in V(T)$. A t-*type *is a pair (ϕ, ξ) of a map $\phi\colon Q_t/{\sim_t} \to \{\mathsf{none}, \mathsf{cont}, \mathsf{dem}\}$ and a bit $\xi \in \{0,1\}$. We denote the set of all t-types by* types_t.

▶ **Observation 8.** *Let $(T, \mathcal{L})$ be a rooted branch decomposition of module-width $w = \mathsf{mw}(T, \mathcal{L})$. For each $t \in V(T)$, $|\mathsf{types}_t| = 2 \cdot 3^{|V_t/{\sim_t}|} \le 2 \cdot 3^w$.*

▶ **Definition 9.** *Let G be a graph with rooted branch decomposition $(T, \mathcal{L})$ and let $t \in V(T)$. Let $(\mathcal{C}, B)$ be a partial b-coloring of G_t, let $C \in \mathcal{C}$ be a color class, and let $\tau = (\phi, \xi) \in \mathsf{types}_t$ be a t-type. We say that C* has t-type τ in $(\mathcal{C}, B)$ *if*

(i) *$\xi = |C \cap B|$ and*

(ii) *for each $Q \in V_t/{\sim_t}$,*

  (a) *if $Q \cap C \neq \emptyset$, and $\nexists v \in (B \setminus C) \cap Q$ such that $N(v) \cap C = \emptyset$, then $\phi(Q) = \mathsf{cont}$;*

  (b) *if $Q \cap C = \emptyset$ and $\exists v \in (B \setminus C) \cap Q$ such that $N(v) \cap C = \emptyset$, then $\phi(Q) = \mathsf{dem}$; and*

  (c) *if $Q \cap C = \emptyset$, and $\nexists v \in (B \setminus C) \cap Q$ such that $N(v) \cap C = \emptyset$, then $\phi(Q) = \mathsf{none}$.*

The reader may have observed that (ii) does not cover all the possibilities. The situation that is not covered is when $Q \cap C \neq \emptyset$ and there is some $v \in (B \setminus C) \cap Q$ such that $N(v) \cap C = \emptyset$. A priori, we can of course not exclude this as a possibility, but there is a simple reason that partial b-colorings that contain a color class in which this situation arises can be disregarded: For the vertex v to become a b-vertex for its color, we have to add a future neighbor of Q to C; but since Q already contains a vertex from C this means that the resulting set is not independent anymore.

▶ **Definition 10** (t-Signature). *Let G be a graph with rooted branch decomposition $(T, \mathcal{L})$, and let $t \in V(T)$. A t-*signature *is a map $\mathsf{sig}_t\colon \mathsf{types}_t \to \{0, 1, \ldots, k\}$ with $\sum_{\tau \in \mathsf{types}_t} \mathsf{sig}_t(\tau) = k$.*

The following bound on the number of t-signatures immediately follows from Observation 8: for each t-type, the function takes one of $k + 1 \le n + 1$ values.

▶ **Observation 11.** *Let G be a graph on n vertices and $(T, \mathcal{L})$ be one of its branch decompositions of module-width $w = \mathsf{mw}(T, \mathcal{L})$. For each $t \in V(T)$, there are at most $n^{2^{\mathcal{O}(w)}}$ many t-signatures.*

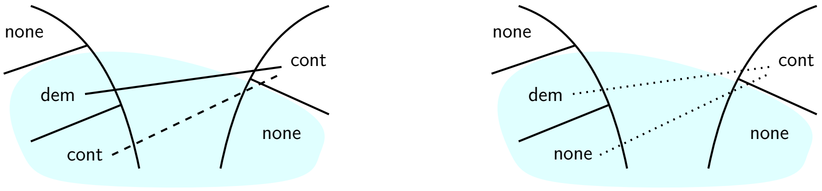


Figure 3 Illustration of Definition 13. The shaded area shows a bubble and the labels on the equivalence classes correspond to type labelings. For the left hand side, note that between a pair of classes that are both labeled "cont", there can be no edge in the operator. Moreover, since the bubble contains a class labeled cont and one labeled dem, the demand of the latter has to be fulfilled at this node, i.e. there has to be an edge from this class to a "cont"-class. The right side shows the situation when the "cont"-class in the bubble is changed to "none", in which case the dotted edges may or may not be present in the operator.

▶ **Definition 12.** *Let G be a graph with rooted branch decomposition $(T, \mathcal{L})$, and let $t \in V(T)$. Let furthermore* sig_t *be a t-signature and $(\mathcal{C}, B)$ a partial b-coloring in G_t. We say that* sig_t represents *$(\mathcal{C}, B)$ if for each t-type $\tau \in \mathsf{types}_t$, there are precisely $\mathsf{sig}_t(\tau)$ color classes in $(\mathcal{C}, B)$ that have t-type τ in $(\mathcal{C}, B)$. We call a partial b-coloring $(\mathcal{C}, B)$ of G_t* representable *if and only if there is a t-signature that represents it.*

3.3 Compatibility

Let $t \in V(T) \setminus \mathrm{L}(T)$ be an internal node of the given rooted branch decomposition, let r and s be its children, and let (H_t, η_r, η_s) be the operator of t. In our algorithm, we want to combine information about partial b-colorings of G_r and G_s to obtain information about partial b-colorings of G_t. We will try to obtain a color class of a partial b-coloring of G_t by taking the union of a color class C_r of a partial b-coloring of G_r and a color class C_s of a partial b-coloring of G_s.

However, in some cases this is not possible. For instance, when C_r contains vertices from some equivalence class $Q_r \in V_r/{\sim_r}$ and C_s contains vertices from some equivalence class $Q_s \in V_s/{\sim_s}$, and in the graph H_t of the operator of t, we have that $Q_rQ_s \in E(H_t)$. Then, in G_t all edges between the set Q_r and Q_s are present which means that $C_r \cup C_s$ is not an independent set in G_t. Another condition is necessary to ensure that several demands that *have to be* met at node t are indeed met. Let $C_t = C_r \cup C_s$ and suppose there is an equivalence class $Q_t \in V_t/{\sim_t}$ that contains a vertex of C_t. Suppose furthermore that there is another equivalence class $Q_r \in V_r/{\sim_r}$ contained in the bubble of Q_t such that C_r has demand to the future neighbors of Q_r. Then, this demand must be fulfilled by a neighbor of Q_r in C_s for otherwise, the equivalence class Q_t both contains vertices of C_t and C_t has demand to the future neighbors of Q_t. The resulting partial b-coloring would not be representable. We illustrate the notion of compatibility in Figure 3.

▶ **Definition 13** (Compatible types)**.** *Let G be a graph with rooted branch decomposition $(T, \mathcal{L})$. Let furthermore $t \in V(T) \setminus \mathrm{L}(T)$ with children r and s, and let (H_t, η_r, η_s) be the operator of t. Let $(\phi_r, \xi_r) \in \mathsf{types}_r$ and $(\phi_s, \xi_s) \in \mathsf{types}_s$. We say that (ϕ_r, ξ_r) and (ϕ_s, ξ_s) are* compatible *if the following conditions hold.*

(i) $\xi_r + \xi_s \leq 1$.

(ii) *There is no pair $Q_r \in V_r/{\sim_r}$, $Q_s \in V_s/{\sim_s}$ such that $Q_rQ_s \in E(H_t)$ and $\phi_r(Q_r) = \phi_s(Q_s) = \mathsf{cont}$.*

(iii) *For each $Q \in V_t/{\sim_t}$ such that there exists a $p \in \{r, s\}$ and a $Q_p \in \eta_p^{-1}(Q)$ with $\phi_p(Q_p) = \mathsf{cont}$, the following holds.*

  - **(a)** *For all $Q_r \in \eta_r^{-1}(Q)$ with $\phi_r(Q_r) = \mathsf{dem}$, there is a $Q_s \in V_s/{\sim_s}$ with $\phi_s(Q_s) = \mathsf{cont}$ and $Q_rQ_s \in E(H_t)$.*
  - **(b)** *Similarly, for all $Q_s \in \eta_s^{-1}(Q)$ with $\phi_s(Q_s) = \mathsf{dem}$, there is a $Q_r \in V_r/{\sim_r}$ with $\phi_r(Q_r) = \mathsf{cont}$ and $Q_sQ_r \in E(H_t)$.*

Given a pair of a color class C_r of a partial b-coloring of G_r and a color class C_s of a partial b-coloring of G_s whose types in the respective colorings are compatible, $C_r \cup C_s$, considered as a color class in a partial b-coloring of G_t, has a fixed type, which can formally be constructed as follows.

▶ **Definition 14** (Merge Type). *Let G be a graph with rooted branch decomposition $(T, \mathcal{L})$. Let furthermore $t \in V(T) \setminus \mathrm{L}(T)$ with children r and s, and let (H_t, η_r, η_s) be the operator of t. Let $\rho = (\phi_r, \xi_r) \in \mathsf{types}_r$ and $\sigma = (\phi_s, \xi_s) \in \mathsf{types}_s$ be a pair of compatible types. The* merge type *of ρ and σ, denoted by $\mu(\rho, \sigma)$, is the following t-type (ϕ_t, ξ_t).*

(i) *$\xi_t = \xi_r + \xi_s$.*

(ii) *For each $Q \in V_t/{\sim_t}$:*

  - **(a)** *If for some $p \in \{r, s\}$, $\exists Q_p \in \eta_p^{-1}(Q)$ with $\phi_p(Q_p) = \mathsf{cont}$, then $\phi_t(Q) = \mathsf{cont}$.*
  - **(b)** *If (iia) does not apply and for some $p \in \{r, s\}$, $\exists Q_p \in \eta^{-1}(Q)$ with $\phi_p(Q_p) = \mathsf{dem}$ and for all $Q_pQ_o \in E(H_t)$ we have that $\phi_o(Q_o) \neq \mathsf{cont}$, then $\phi_t(Q) = \mathsf{dem}$.*
  - **(c)** *If neither (iia) nor (iib) applies, then $\phi_t(Q) = \mathsf{none}$.*

Towards a notion of compatibility of signatures, we first define a structure we call *merge skeleton*. Given a node $t \in V(T)$ with children r and s, the merge skeleton is an edge-labeled bipartite graph whose vertices are the r-types and the s-types, with the merge type of a compatible pair of types $\rho \in \mathsf{types}_r$, $\sigma \in \mathsf{types}_s$ written on the edge $\rho\sigma$. Such an edge is meant to represent the fact that taking the union of a color class C_r that has r-type ρ in a partial b-coloring of G_r with a color class C_s that has s-type σ in a partial b-coloring of G_s results in a color class of t-type $\mu(\rho, \sigma)$ in a partial b-coloring of G_t.

▶ **Definition 15** (Merge skeleton). *Let G be a graph and $(T, \mathcal{L})$ one of its rooted branch decompositions. Let $t \in V(T) \setminus \mathrm{L}(T)$ with children r and s. The* merge skeleton *of r and s is an edge-labeled bipartite graph $(\mathfrak{J}, \mathfrak{m})$ where*

- *$V(\mathfrak{J}) = \mathsf{types}_r \cup \mathsf{types}_s$,*
- *for all $\rho \in \mathsf{types}_r$, $\sigma \in \mathsf{types}_s$, $\rho\sigma \in E(\mathfrak{J})$ if and only if ρ and σ are compatible, and*
- *$\mathfrak{m} \colon E(\mathfrak{J}) \to \mathsf{types}_t$ is such that for all $\rho\sigma \in E(\mathfrak{J})$, $\mathfrak{m}(\rho\sigma)$ is the merge type of ρ and σ.*

Now, any pair of an r-signature sig_r and an s-signature sig_s can "flesh out" the merge skeleton $(\mathfrak{J}, \mathfrak{m})$ of r and s, in the following sense. We can consider the union of sig_r and sig_s as a map labeling the vertices of $\mathfrak{J}$. Then, an edge-labeling $\mathfrak{n}$ of $\mathfrak{J}$ with integers from $\{0, 1, \ldots, k\}$, such that for each vertex of $\mathfrak{J}$, the sum over its incident edges e of $\mathfrak{n}(e)$ is equal to its vertex label, produces a t-signature sig_t. We can read off how many color classes of each type there are from the edge labeling $\mathfrak{n}$.

▶ **Definition 16** (Compatible signatures). *Let $(T, \mathcal{L})$ be a rooted branch decomposition. Let furthermore $t \in V(T) \setminus \mathrm{L}(T)$ with children r and s. Let sig_t be a t-signature, let sig_r be an r-signature and sig_s be a s-signature. We say that $(\mathsf{sig}_t, \mathsf{sig}_r, \mathsf{sig}_s)$ is* compatible *if there is a triple $(\mathfrak{J}, \mathfrak{m}, \mathfrak{n})$ such that $(\mathfrak{J}, \mathfrak{m})$ is the merge skeleton of r and s, and $\mathfrak{n} \colon E(\mathfrak{J}) \to \{0, 1, \ldots, k\}$ is a map with the following properties.*

(i) *For all $p \in \{r, s\}$ and all $\pi \in \mathsf{types}_p$, $\sum_{e \in E(\mathfrak{J})\colon \pi \in e} \mathfrak{n}(e) = \mathsf{sig}_p(\pi)$.*

(ii) *For all $\tau \in \mathsf{types}_t$, $\sum_{e \in E(\mathfrak{J})\colon \mathfrak{m}(e) = \tau} \mathfrak{n}(e) = \mathsf{sig}_t(\tau)$.*

▶ **Lemma 17 (♣).** *Let G be a graph on n vertices and let $(T, \mathcal{L})$ be one of its rooted branch decomposition of module-width $w = \mathsf{mw}(T, \mathcal{L})$. Let $t \in V(T) \setminus \mathrm{L}(T)$ with children r and s. Let sig_t be a t-signature, sig_r be an r-signature, and sig_s be an s-signature. One can decide in time $n^{2^{\mathcal{O}(w)}}$ whether or not $(\mathsf{sig}_t, \mathsf{sig}_r, \mathsf{sig}_s)$ is compatible.*

3.4 Merging and Splitting Partial *b*-Colorings

In this section we state the lemmas that show that the notions introduced above work as desired, and the technical lemmas we prove here will be the cornerstone of the correctness proof of the resulting algorithm.

▶ **Lemma 18 (♣).** *Let G be a graph with rooted branch decomposition $(T, \mathcal{L})$ and let $t \in V(T) \setminus \mathrm{L}(T)$ be an internal node with children r and s. Let sig_r be an r-signature, sig_s be an s-signature, and sig_t be a t-signature such that:*

- *For all $p \in \{r, s\}$, there is a partial b-coloring $(\mathcal{C}_p, B_p)$ in G_p represented by sig_p, and*
- *$(\mathsf{sig}_t, \mathsf{sig}_r, \mathsf{sig}_s)$ is compatible.*

Then, there is a partial b-coloring $(\mathcal{C}_t, B_t)$ of G_t that is represented by sig_t.

▶ **Lemma 19 (♣).** *Let G be a graph with rooted branch decomposition $(T, \mathcal{L})$ and let $t \in V(T) \setminus \mathrm{L}(T)$ be an internal node with children r and s. Let sig_t be a t-signature, and suppose there is a partial b-coloring $(\mathcal{C}_t, B_t)$ of G_t which is represented by sig_t. Then, there exists an r-signature sig_r and an s-signature sig_s such that*

- *for all $p \in \{r, s\}$ there is a partial b-coloring $(\mathcal{C}_p, B_p)$ represented by sig_p, and*
- *$(\mathsf{sig}_t, \mathsf{sig}_r, \mathsf{sig}_s)$ is compatible.*

3.5 The Algorithm

▶ **Definition of the table entries.** *For a node $t \in V(T)$ and a t-signature sig_t, we let $\mathsf{tab}[t, \mathsf{sig}_t] = 1$ if and only if there exists a partial b-coloring of G_t that is represented by sig_t.*

We now show that if all table entries have been computed correctly, then the solution can be read off the table entries stored at the root $\mathfrak{r}$ of the given rooted branch decomposition. Observe that since $V_\mathfrak{r} = V(G)$ and therefore $\overline{V_t} = \emptyset$, the equivalence relation $\sim_\mathfrak{r}$ has one equivalence class, namely $V(G)$.

▶ **Lemma 20 (♣).** *Let G be a graph with rooted branch decomposition $(T, \mathcal{L})$ and let $\mathfrak{r} \in V(T)$ be the root of T. Let ρ be the $\mathfrak{r}$-type $(\phi_\mathfrak{r}, \xi_\mathfrak{r})$ with $\xi_\mathfrak{r} = 1$ and $\phi_\mathfrak{r}(V(G)) = \mathsf{cont}$. Let $\mathsf{sig}_\mathfrak{r}$ be the $\mathfrak{r}$-signature letting $\mathsf{sig}_\mathfrak{r}(\rho) = k$. Then, G has a b-coloring with k colors if and only if $\mathsf{tab}[\mathfrak{r}, \mathsf{sig}_\mathfrak{r}] = 1$.*

The table entries at the leaves are computed by brute force, and we defer the details to the full version. We compute the table entries at the internal nodes as follows.

▶ **Internal nodes of T.** *Now let $t \in V(T) \setminus \mathrm{L}(T)$ with children r and s. For each t-signature sig_t, we let $\mathsf{tab}[t, \mathsf{sig}_t] = 1$ if and only if there exists a pair $(\mathsf{sig}_r, \mathsf{sig}_s)$ of an r-signature sig_r and an s-signature sig_s such that*

(J1) $\mathsf{tab}[r, \mathsf{sig}_r] = 1$ *and* $\mathsf{tab}[s, \mathsf{sig}_s] = 1$, *and*
(J2) $(\mathsf{sig}_t, \mathsf{sig}_r, \mathsf{sig}_s)$ *is compatible.*

▶ **Lemma 21 (♣).** *For each $t \in V(T)$ and t-signature sig_t, the above algorithm computes the table entry $\mathsf{tab}[t, \mathsf{sig}_t]$ correctly.*

We wrap up. By Lemma 21, the algorithm computes all table entries correctly, and by Lemma 20, the solution to the instance can be determined upon inspecting the table entries associated with the root of the given branch decomposition. Correctness of the algorithm follows. The runtime follows essentially from Observation 11 and Lemma 17. We give the details in the full version.

▶ **Theorem 22.** *There is an algorithm that solves b-*Coloring *in time $n^{2^{\mathcal{O}(w)}}$, where n denotes the number of vertices of the input graph, and w denotes the module-width of a given rooted branch decomposition of the input graph.*

3.6 Fall Coloring

Recall that a *fall coloring* is a special type of b-coloring where *every* vertex is a b-vertex for its color. We adapt our algorithm for b-Coloring on graphs of bounded clique-width to solve Fall Coloring, and therefore obtain the following theorem.

▶ **Theorem 23 (♣).** *There is an algorithm that solves* Fall Coloring *in time $n^{2^{\mathcal{O}(w)}}$, where n denotes the number of vertices of the input graph, and w denotes the module-width of a given rooted branch decomposition of the input graph.*

▶ **Proposition 24 (♣).** *The* Fall Coloring *problem on graphs on n vertices parameterized by the module-width w of the input graph is* W[1]-hard and cannot be solved in time $n^{2^{o(w)}}$, unless ETH fails. Moreover, the hardness holds even when a linear branch decomposition of width w is provided.

4 Parameterized by Vertex Cover

We conclude by stating that b-Coloring parameterized by the size of a vertex cover of the input graph is FPT.

▶ **Theorem 25 (♣).** *There is an algorithm that solves b-*Coloring *in time $2^{\mathcal{O}(\ell^2 \log \ell)} \cdot n^{\mathcal{O}(1)}$, where ℓ denotes the vertex cover number of the input graph.*

References

1 Pierre Aboulker, Édouard Bonnet, Eun Jung Kim, and Florian Sikora. Grundy coloring & friends, half-graphs, bicliques. In *STACS 2020*, volume 154 of *LIPIcs*, pages 58:1–58:18, 2020.

2 Hans-Jürgen Bandelt and Henry Martin Mulder. Distance-hereditary graphs. *Journal of Combinatorial Theory Series B*, 41:182–208, 1986.

3 Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, and Yota Otachi. Grundy distinguishes treewidth from pathwidth. In *ESA 2020*, volume 173 of *LIPIcs*, pages 14:1–14:19, 2020.

4 J. Adrian Bondy and Uppaluri S. R. Murty. *Graph Theory*, volume 244 of *Graduate Texts in Mathematics*. Springer, 2008.

5 Flavia Bonomo, Guillermo Durán, Frederic Maffray, Javier Marenco, and Mario Valencia-Pabon. On the b-coloring of cographs and P_4-sparse graphs. *Graphs and Combinatorics*, 25(2):153–167, 2009.

6 Flavia Bonomo, Oliver Schaudt, Maya Stein, and Mario Valencia-Pabon. b-Coloring is NP-hard on co-bipartite graphs and polytime solvable on tree-cographs. *Algorithmica*, 73(2):289–305, 2015.

7 Victor A. Campos, Carlos V. Lima, Nicolas A. Martins, Leonardo Sampaio, Marcio C. Santos, and Ana Silva. The b-chromatic index of graphs. *Discrete Mathematics*, 338(11):2072–2079, 2015.

8 Victor A. Campos, Carlos Vinicius G. C. Lima, and Ana Silva. Graphs of girth at least 7 have high b-chromatic number. *European Journal of Combinatorics*, 48:154–164, 2015.

9 Victor A. Campos, Cláudia Linhares-Sales, Rudini Sampaio, and Ana Karolinna Maia. Maximization coloring problems on graphs with few P_4. *Discrete Applied Mathematics*, 164:539–546, 2014.

10 Victor A. Campos and Ana Silva. Edge-b-coloring trees. *Algorithmica*, 80(1):104–115, 2018.

11 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

12 Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46(2):218–270, 1993.

13 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

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

15 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics.* Springer, Heidelberg, fifth edition, 2016.

16 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Springer, 2013.

17 Jean E. Dunbar, Sandra M. Hedetniemi, S.T. Hedetniemi, David P. Jacobs, J. Knisely, R.C. Laskar, and Douglas F. Rall. Fall colorings of graphs. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 33:257–274, 2000.

18 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.

19 Wolfgang Espelage, Frank Gurski, and Egon Wanke. How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. In *WG 2001*, pages 117–128, 2001.

20 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM Journal on Computing*, 39(5):1941–1956, 2010.

21 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Clique-width III: Hamiltonian cycle and the odd case of graph coloring. *ACM Transactions on Algorithms*, 15(1):9:1–9:27, 2019.

22 Wayne Goddard and Michael A. Henning. Independent domination in graphs: A survey and recent results. *Discrete Mathematics*, 313(7):839–854, 2013.

23 Martin Charles Golumbic and Udi Rotics. On the clique-width of some perfect graph classes. *International Journal of Foundations of Computer Science*, 11(03):423–443, 2000.

24 Frédéric Havet, Claudia Linhares Sales, and Leonardo Sampaio. b-coloring of tight graphs. *Discrete Applied Mathematics*, 160(18):2709–2715, 2012.

25 Frédéric Havet and Leonardo Sampaio. On the Grundy and *b*-chromatic numbers of a graph. *Algorithmica*, 65:885–899, 2013.

26 Pinar Heggernes and Jan Arne Telle. Partitioning graphs into generalized dominating sets. *Nordic Journal on Computing*, 5(2):128–142, 1998.

27 Russell Impagliazzo and Ramamohan Paturi. On the complexity of *k*-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

28 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

29 Robert W. Irving and David F. Manlove. The b-chromatic number of a graph. *Discrete Applied Mathematics*, 91(1-3):127–141, 1999.

30 Lars Jaffke and Paloma T. Lima. A complexity dichotomy for critical values of the b-chromatic number of graphs. *Theoretical Computer Science*, 815:182–196, 2020.

31 Klaus Jansen. Complexity results for the optimum cost chromatic partition problem, 1997.

32 Jan Kratochvíl, Zsolt Tuza, and Margit Voigt. On the b-chromatic number of graphs. In *WG 2002*, pages 310–320, 2002.

33 Renu Laskar and Jeremy Lyle. Fall colouring of bipartite graphs and cartesian products of graphs. *Discrete Applied Mathematics*, 157(2):330–338, 2009.

34 Juho Lauri and Christodoulos Mitillos. Complexity of fall coloring for restricted graph classes. In *30th IWOCA*, pages 352–364. Springer, 2019.

35 Jeremy Lyle, Nate Drake, and Renu Laskar. Independent domatic partitioning or fall coloring of strongly chordal graphs. *Congressus Numerantium*, 172:149–159, 2005.

36 Johann A. Makowsky and Udi Rotics. On the clique-width of graphs with few P_4's. *International Journal of Foundations of Computer Science*, 10(03):329–348, 1999.

37 Christodoulos Mitillos. *Topics in Graph Fall-Coloring*. PhD thesis, Illinois Institute of Technology, 2016.

38 Fahad Panolan, Geevarghese Philip, and Saket Saurabh. On the parameterized complexity of b-chromatic number. *Journal of Computer and System Sciences*, 84:120–131, 2017.

39 Michaël Rao. *Décompositions de graphes et algorithmes efficaces*. PhD thesis, University of Metz, 2006.

40 Michaël Rao. Clique-width of graphs defined by one-vertex extensions. *Discrete Mathematics*, 308(24):6157–6165, 2008.

41 Leonardo Sampaio. *Algorithmic aspects of graph colourings heuristics*. PhD thesis, Université Nice Sophia Antipolis, 2012.

42 Ana Silva. Graphs with small fall-spectrum. *Discrete Applied Mathematics*, 254:183–188, 2019.

43 Ana Shirley Ferreira da Silva. *The b-chromatic number of some tree-like graphs*. PhD thesis, Université Joseph-Fourier - Grenoble I, 2010.

44 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.

45 Clara Inés Betancur Velasquez, Flavia Bonomo, and Ivo Koch. On the b-coloring of P_4-tidy graphs. *Discrete Applied Mathematics*, 159(1):60–68, 2011.

46 Egon Wanke. k-NLC graphs and polynomial algorithms. *Discrete Applied Mathematics*, 54:251–266, 1994.

A Ramsey Theorem for Finite Monoids

Ismaël Jecker
Institute of Science and Technology, Klosterneuburg, Austria

Abstract

Repeated idempotent elements are commonly used to characterise iterable behaviours in abstract models of computation. Therefore, given a monoid M, it is natural to ask how long a sequence of elements of M needs to be to ensure the presence of consecutive idempotent factors. This question is formalised through the notion of the *Ramsey function* R_M associated to M, obtained by mapping every $k \in \mathbb{N}$ to the minimal integer $\mathsf{R}_M(k)$ such that every word $u \in M^*$ of length $\mathsf{R}_M(k)$ contains k consecutive non-empty factors that correspond to the same idempotent element of M.

In this work, we study the behaviour of the Ramsey function R_M by investigating the *regular $\mathcal{D}$-length* of M, defined as the largest size $L(M)$ of a submonoid of M isomorphic to the set of natural numbers $\{1, 2, \dots, L(M)\}$ equipped with the max operation. We show that the regular $\mathcal{D}$-length of M determines the degree of R_M, by proving that $k^{L(M)} \leq \mathsf{R}_M(k) \leq (k|M|^4)^{L(M)}$.

To allow applications of this result, we provide the value of the regular $\mathcal{D}$-length of diverse monoids. In particular, we prove that the full monoid of $n \times n$ Boolean matrices, which is used to express transition monoids of non-deterministic automata, has a regular $\mathcal{D}$-length of $\frac{n^2+n+2}{2}$.

2012 ACM Subject Classification Theory of computation → Algebraic language theory

Keywords and phrases Semigroup, monoid, idempotent, automaton

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.44

Related Version *Full Version*: https://arxiv.org/abs/2101.05895

Funding This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 754411.

Acknowledgements I wish to thank Michaël Cadilhac, Emmanuel Filiot and Charles Paperman for their valuable insights concerning Green's relations.

1 Introduction

The algebraic approach to language theory was initiated by Schützenberger with the definition of the syntactic monoid associated to a formal language [14]. This led to several parallels being drawn between classes of languages and varieties of monoids, the most famous being that rational languages are characterised by finite syntactic monoids [12], and that star-free languages are characterised by finite aperiodic syntactic monoids [15]. These characterisations motivate the study of finite monoids as a way to gain some insight about automata. In this work, we focus on the following problem:

> *Given a finite monoid M and $k \in \mathbb{N}$, what is the minimal integer $R_M(k)$ such that every word $u \in M^*$ of length $R_M(k)$ contains k consecutive factors corresponding to the same idempotent element of M?*

The interest of this problem lies in the fact that when we model the behaviours of an abstract machine as elements of a monoid, repeated idempotent factors often characterise the behaviours that have good properties with respect to iteration. This can be used, for instance, to obtain pumping lemmas, as seen in [8] for weighted automata.

A partial answer to this problem is obtained by using Ramsey's Theorem [13] or Simon's Factorisation Forest Theorem [16] (these techniques are detailed in the full version), as both approaches provide upper bounds for $\mathsf{R}_M(k)$. However, neither approximation is precise:

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 44; pp. 44:1–44:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Ramsey's theorem disregards the monoid structure, and the Factorisation Forest Theorem guarantees much more than what is required here. We prove a version of Ramsey's Theorem adapted to monoids, or, equivalently, a weaker version of the Forest Factorisation Theorem, that yields an improved bound relying on a parameter of monoids called the regular $\mathcal{D}$-length. We now present some examples, followed with an overview of the main concepts studied in this paper: the Ramsey function associated to a monoid and the regular $\mathcal{D}$-length.

1.1 Examples

We describe three families of monoids, along with the corresponding idempotent elements.

Max monoid. The max monoid H_n is the set $\{1, 2, \dots, n\}$, equipped with the max operation. In this monoid, every element i is idempotent since $\max(i, i) = i$.

Transformation monoid. The (full) transformation monoid T_n is the set of all (partial) functions from a set of n elements into itself, equipped with the composition. See [3] for a detailed definition of T_n and its properties. Transformation monoids contain a wide range of idempotent elements. For instance, the identity function, mapping each element to itself, or the constant function f_i, mapping all elements to one fixed element i, are idempotent. In general, a function f is idempotent if and only if each element i of its range satisfies $f(i) = i$. Transformations are commonly used to express transition monoids of deterministic finite state automata, as in this setting each input letter acts as a function over the set of states.

Relation monoid. For non-deterministic automata, transition monoids are more complex: functions fail to model the behaviour of the input letters since a single state can transition towards several distinct states. We use the (full) relation monoid B_n of all $n \times n$ Boolean matrices (matrices with values in $\{0, 1\}$), equipped with the usual matrix composition (considering that $1 + 1 = 1$). There are plenty of idempotent matrices, for instance every diagonal matrix, or the full upper triangular matrix. Idempotent Boolean matrices are characterised in [9], they correspond to specific orders over the subsets of $\{1, 2, \dots, n\}$.

1.2 Ramsey function

Given a finite monoid M, the *Ramsey function* R_M associated to M maps each $k \in \mathbb{N}$ to the minimal integer $\mathsf{R}_M(k)$ such that every sequence of elements of M of length $\mathsf{R}_M(k)$ contains k non-empty consecutive factors that all correspond to the same idempotent element of M.

Related work. There are several known methods to approximate the Ramsey function R_M of a monoid M. Ramsey's Theorem and Simon's Factorisation Forest Theorem are commonly used, however, as stated before, these approaches are too general to obtain a precise bound. The value of $\mathsf{R}_M(k)$ is studied in [4] in the particular case $k = 1$. The authors prove that for a monoid M that contains N non-idempotent elements, $\mathsf{R}_M(1) \leq 2^N - 1$. No general related lower bound is proved, but they show that for every $N \in \mathbb{N}$, there exists a monoid M_N with N non-idempotent elements that actually reaches the upper bound: $\mathsf{R}_{M_N}(1) = 2^N - 1$.

Our contributions. We prove new bounds for R_M by following a different approach: instead of focusing on the non-idempotent elements of M, we study its idempotent elements, and the way in which they interact. In Section 3, we start by considering two specific cases where the exact value of the Ramsey function is easily obtained. First, for a group $\mathcal{G}$, the

Ramsey function is polynomial with respect to the size: $\mathsf{R}_{\mathcal{G}}(k) = k|\mathcal{G}|$. Second, we call *max monoid* H_n the set $\{1, 2, \dots, n\}$ equipped with the max operation, and we show that here the Ramsey function is exponential with respect to the size: $\mathsf{R}_{H_n}(k) = k^n$. The later result implies that k^n is a lower bound for the Ramsey function of every monoid M that has H_n as a submonoid. We prove a related upper bound: We define the *regular $\mathcal{D}$-length* $L(M)$ of M as the size of the largest max monoid $H_{L(M)}$ embedded in M, and show the following:

▶ **Theorem 1.** *Every monoid M of regular $\mathcal{D}$-length L satisfies $k^L \leq \mathsf{R}_M(k) \leq (k|M|^4)^L$.*

Stated differently: every word $u \in M^*$ of length $(k|M|^4)^L$ contains k consecutive non-empty factors corresponding to the same idempotent element of M, and, conversely, there exists a word $u_M \in M^*$ of length $k^L - 1$ that does not contain k consecutive non-empty factors corresponding to the same idempotent element. Note that while the gap between the lower and upper bound is still wide, this shows that the degree of the Ramsey function R_M is determined by the regular $\mathcal{D}$-length of M.

1.3 Regular $\mathcal{D}$-length

Theorem 1 states that the degree of the Ramsey function of a monoid M is determined by the regular $\mathcal{D}$-length of M, which is the size of the largest max monoid embedded in M. We now show that for transformation monoids and relation monoids, the regular $\mathcal{D}$-length is exponentially shorter than the size. Let us begin by mentioning an equivalent definition of the regular $\mathcal{D}$-length in terms of Green's relations. While this alternative definition is not used in the proofs presented in this paper, it allows us to immediately obtain the regular $\mathcal{D}$-length of monoids whose Green's relations are known.

Alternative definition. The regular $\mathcal{D}$-length of a monoid M is the size of its largest chain of regular $\mathcal{D}$-classes. A $\mathcal{D}$-class of M is an equivalence class of the preorder $\leq_{\mathcal{D}}$ defined by $m \leq_{\mathcal{D}} m'$ if $m = s \cdot m' \cdot t$ for some $s, t \in M$, and it is called regular if it contains at least one idempotent element (see [11] for more details). The equivalence between both definitions is proved in the full version.

Computing the regular $\mathcal{D}$-length. The following table compares the size and the regular $\mathcal{D}$-length of the monoids mentioned earlier. The entries corresponding to the sizes are considered to be general knowledge. We detail below the row listing the regular $\mathcal{D}$-lengths.

Monoid	G	H_n	T_n	B_n
Size	$\lvert G\rvert$	n	$(n+1)^n$	$2^{(n^2)}$
Regular $\mathcal{D}$-length	1	n	$n+1$	$\frac{n^2+n+2}{2}$

First, every group $\mathcal{G}$ contains a single idempotent element (the neutral element), hence its regular $\mathcal{D}$-length is 1. Then, using the definition of the regular $\mathcal{D}$-length in terms of embedded max monoid, we immediately obtain that $L(H_n)$ is equal to n. We get the next entry using the definition of the regular $\mathcal{D}$-length in terms of chain of $\mathcal{D}$-classes: The transformation monoid T_n is composed of a single chain of $n+1$ $\mathcal{D}$-classes that are all regular [3], hence its regular $\mathcal{D}$-length is $n+1$.

Finally, for the relation monoid B_n, the situation is not as clear: the $\mathcal{D}$-classes do not form a single chain, and some of them are not regular. Determining the exact size of the largest chain of $\mathcal{D}$-classes (note the absence of "regular") is still an open question, yet it is known to grow exponentially with respect to n: a chain of $\mathcal{D}$-classes whose size is the Fibonacci number $F_{n+3} - 1$ is constructed in [2], and, conversely, the upper bound $2^{n-1} + n - 1$ is proved in [6] (and slightly improved in [7, 17, 5]). Our second main result is that, as long as we only consider chains of *regular* $\mathcal{D}$-classes, we can obtain the precise value of the maximal length, and, somewhat surprisingly, it is only quadratic in n:

▶ **Theorem 2.** *The regular $\mathcal{D}$-length of the monoid of $n \times n$ Boolean matrices is $\frac{n^2+n+2}{2}$.*

Therefore, the regular $\mathcal{D}$-length of a transformation monoid is exponentially smaller than its size, and the regular $\mathcal{D}$-length of a relation monoid is even exponentially smaller than its largest chain of $\mathcal{D}$-classes. For such kind of monoids, Theorem 1 performs considerably better than previously known methods to find idempotent factors. For instance, it was used in [10] to close the complexity gap left in [1] for the problem of deciding whether the function defined by a given two-way word transducer is definable by a one-way transducer.

2 Definitions and notations

We define in this section the notions that are used throughout the paper. We denote by $\mathbb{N}$ the set $\{0, 1, 2, \ldots\}$, and for all $i \leq j \in \mathbb{N}$ we denote by $[i, j]$ the interval $\{i, i+1, \ldots, j\}$.

Monoids. A (finite) *semigroup* $(S, \cdot)$ is a finite set S equipped with a binary operation $\cdot : S \times S \to S$ that is *associative*: $(s_1 \cdot s_2) \cdot s_3 = s_1 \cdot (s_2 \cdot s_3)$ for every $s_1, s_2, s_3 \in S$. A *monoid* is a semigroup $(M, \cdot)$ that contains a *neutral element* 1_M: $m \cdot 1_M = m = 1_M \cdot m$ for all $m \in M$. A *group* is a monoid $(\mathcal{G}, \cdot)$ in which every element $g \in \mathcal{G}$ has an *inverse element* $g^{-1} \in \mathcal{G}$: $g \cdot g^{-1} = 1_{\mathcal{G}} = g^{-1} \cdot g$. We always denote the semigroup operation with the symbol $\cdot$. As a consequence, we identify a semigroup $(S, \cdot)$ with its set of elements S.

An element e of a semigroup S is called *idempotent* if it satisfies $e \cdot e = e$. Note that whereas a finite semigroup does not necessarily contain a neutral element, it always contains at least one idempotent element: iterating any element $s \in S$ eventually yields an idempotent element, called the *idempotent power* of s, and denoted $s^{\#} \in S$.

A *homomorphism* between two monoids M and M' is a function $\varphi : M \to M'$ preserving the monoid structure: $\varphi(m_1 \cdot m_2) = \varphi(m_1) \cdot \varphi(m_2)$ for all $m_1, m_2 \in M$ and $\varphi(1_M) = \varphi(1_{M'})$. A *monomorphism* is an injective homomorphism, an *isomorphism* is a bijective homomorphism.

Ramsey decomposition. Let M be a monoid. A *word* over M is a finite sequence $u = m_1 m_2 \ldots m_n \in M^*$ of elements of M. The *length* of u is its number of symbols $|u| = n \in \mathbb{N}$. We enumerate the positions between the letters of u starting from 0 before the first letter, until $|u|$ after the last letter. A *factor* of u is a subsequence of u composed of the letters between two such positions i and j: $u[i, j] = m_{i+1} m_{i+2} \ldots m_j \in M^*$ for some $0 \leq i \leq j \leq |u|$ (where $u[i, j] = \varepsilon$ if $i = j$). We denote by $\pi(u)$ the element $1_M \cdot m_1 \cdot m_2 \cdot \ldots \cdot m_n \in M$, and we say that u *reduces* to $\pi(u)$. For every integer $k \in \mathbb{N}$, a *k-decomposition* of u is a decomposition of u in $k+2$ factors such that the k middle ones are non-empty:

$$u = x y_1 y_2 \ldots y_k z, \text{ where } x, z \in M^*, \text{ and } y_i \in M^+ \text{ for every } 1 \leq i \leq k.$$

A k-decomposition is called *Ramsey* if all the middle factors $y_1, y_2, \ldots, y_k$ reduce to the same idempotent element $e \in M$. For instance, a word has a Ramsey 1-decomposition if and only if it contains a factor that reduces to an idempotent element. The *Ramsey function* $\mathsf{R}_M : \mathbb{N} \to \mathbb{N}$ associated to M is the function mapping each $k \in \mathbb{N}$ to the minimal $\mathsf{R}_M(k) \in \mathbb{N}$ such that every word $u \in M^*$ of length $\mathsf{R}_M(k)$ has a Ramsey k-decomposition.

3 Ramsey decompositions

In this section, we bound the Ramsey function R_M associated to a monoid M. As a first step we consider two basic cases for which the exact value of the Ramsey function is obtained: in Subsection 3.1 we show that every group $\mathcal{G}$ satisfies $\mathsf{R}_{\mathcal{G}}(k) = k|\mathcal{G}|$, and in Subsection 3.2 we show that every max monoid H_n (obtained by equipping the first n positive integers with the max operation) satisfies $\mathsf{R}_{H_n}(k) = k^n$. Finally, in Subsection 3.3, we prove bounds in the general case by studying the submonoids of M isomorphic to a max monoid.

3.1 Group: prefix sequence algorithm

We show that in a group, the Ramsey function is polynomial with respect to the size.

▶ **Proposition 3.** *For every group $\mathcal{G}$, $\mathsf{R}_{\mathcal{G}}(k) = k|\mathcal{G}|$ for all $k \in \mathbb{N}$.*

We fix for this subsection a group $\mathcal{G}$ and $k \in \mathbb{N}$. We begin by proving an auxiliary lemma, which we then apply to prove matching bounds for $\mathsf{R}_{\mathcal{G}}(k)$: First, we define an algorithm that extracts a Ramsey k-decomposition out of every word of length $k|\mathcal{G}|$. Then, we present the construction of a witness $u_{\mathcal{G}} \in \mathcal{G}^*$ of length $k|\mathcal{G}| - 1$ that has no Ramsey k-decompositions.

Key lemma. In a group, the presence of inverse elements allows us to establish a correspondence between the factors of a word $u \in \mathcal{G}^*$ that reduce to the neutral element, and the pairs of prefixes of u that both reduce to the same element.

▶ **Lemma 4.** *Two prefixes $u[0,i]$ and $u[0,j]$ of a word $u \in \mathcal{G}^*$ reduce to the same element if and only if $u[i,j]$ reduces to the neutral element of $\mathcal{G}$.*

Proof. Let $u \in \mathcal{G}^*$ be a word. The statement is a direct consequence of the fact that for every $0 \leq i \leq j \leq |u|$, $\pi(u[0,i]) \cdot \pi(u[i,j]) = \pi(u[0,j])$: If $\pi(u[0,i]) = \pi(u[0,j])$, then

$$\pi(u[i,j]) = \pi(u[0,i])^{-1} \cdot \pi(u[0,j]) = \pi(u[0,i])^{-1} \cdot \pi(u[0,i]) = 1_{\mathcal{G}}.$$

Conversely, if $\pi(u[i,j]) = 1_{\mathcal{G}}$, then

$$\pi(u[0,i]) = \pi(u[0,i]) \cdot 1_{\mathcal{G}} = \pi(u[0,i]) \cdot \pi(u[i,j]) = \pi(u[0,j]).$$ ◀

Algorithm. We define an algorithm constructing Ramsey k-decompositions.

```
ALG_1: Start with $u \in \mathcal{G}^*$ of length $k|\mathcal{G}|$;
    a. Compute the $k|\mathcal{G}| + 1$ prefixes $\pi(u[0,0])$, $\pi(u[0,1])$, ..., $\pi(u[0,|u|])$ of $u$;
    b. Find $k + 1$ indices $i_0$, $i_1$, ..., $i_k$ such that all the $\pi(u[0,i_j])$ are equal;
    c. Return the Ramsey $k$-decomposition $u = u[i_0,i_1]u[i_1,i_2] \ldots u[i_{k-1},i_k]$.
```

Since Lemma 4 ensures that every pair of elements i_j, i_{j+1} identified at step 2 satisfies $\pi(u[i_j,i_{j+1}]) = 1_{\mathcal{G}}$, we are guaranteed that the returned k-decomposition is Ramsey.

Witness. We build a word $u_{\mathcal{G}} \in \mathcal{G}^*$ of length $k|\mathcal{G}| - 1$ that has no Ramsey k-decompositions. Let $v = a_1 a_2 \ldots a_{k|\mathcal{G}|} \in \mathcal{G}^*$ be a word of length $k|\mathcal{G}|$, starting with the letter $1_{\mathcal{G}}$, and containing exactly k times each element of $\mathcal{G}$. For instance, given an enumeration $g_1, g_2, \ldots, g_{|\mathcal{G}|}$ of the elements of $\mathcal{G}$ starting with $g_1 = 1_{\mathcal{G}}$, we can simply pick $v = g_1^k g_2^k \ldots g_{|\mathcal{G}|}^k$. Now let $u_{\mathcal{G}} = b_1 b_2 \ldots b_{k|\mathcal{G}|-1}$ be the word whose sequence of reduced prefixes is v: for every $1 \leq i \leq k|\mathcal{G}| - 1$, the letter b_i is equal to $a_i^{-1} \cdot a_{i+1}$. Then for every k-decomposition of $u_{\mathcal{G}}$, at least one of the factors do not reduce to the neutral element of $\mathcal{G}$, since otherwise Lemma 4 would imply the existence of $k+1$ identical letters in v, which is not possible by construction. As a consequence, $u_{\mathcal{G}}$ has no Ramsey k-decompositions.

3.2 Max monoid: divide and conquer algorithm

Given an integer $n \in \mathbb{N}$, the *max monoid*, denoted H_n, is the monoid over the set $\{1, 2, \dots, n\}$ with the associative operation $i \cdot j = \max(i, j)$. Whereas in a group only the neutral element is idempotent, each element i of the max monoid H_n is idempotent since $\max(i, i) = i$. As a result of this abundance of idempotent elements, an exponential bound is required to ensure the presence of consecutive factors reducing to the same idempotent element.

▶ **Proposition 5.** *For every max monoid H_n, $R_{H_n}(k) = k^n$ for all $k \in \mathbb{N}$.*

The proof is done in two steps: we first define an algorithm that extracts a Ramsey k-decomposition out of every word of length k^n, and then we present the construction of a witness u_n of length $k^n - 1$ that has no Ramsey k-decompositions.

Algorithm. We define an algorithm that extracts a Ramsey k-decompositions out of each word $u \in H_n^*$ of length k^n. It is a basic divide and conquer algorithm: we divide the initial word u into k equal parts. If each of the k parts reduces to n, they form a Ramsey k-decomposition since n is an idempotent element. Otherwise, one part does not contain the maximal element $n \in H_n$, and we start over with it. Formally,

```
ALG2: Start with $u \in H_n^*$ of length $k^n$, initialize $j$ to $n$. While $j > 0$, repeat the following:
    a. Split $u$ into $k$ factors $u_1$, $u_2$, ..., $u_k$ of length $k^{j-1}$;
    b. If every $u_i$ contains the letter $j$, return the Ramsey $k$-decomposition $u = u_1 u_2 \dots u_k$;
    c. If $u_i$ does not contain $j$ for some $1 \leq i \leq j$, decrement $j$ by 1 and set $u := u_i \in H_{j-1}^*$.
```

The algorithm is guaranteed to eventually return a Ramsey k-decomposition: if the n^{th} cycle of the algorithm is reached, it starts with a word of length k whose letters are in the monoid H_1, which only contains the letter 1, hence the algorithm will go to step b.

Witness. We construct an infinite sequence of words $u_1, u_2, \dots \in \mathbb{N}^*$ such that for all $n \in \mathbb{N}$,
(a) $u_n \in H_n$ satisfies $|u_n| = k^n - 1$ and
(b) u_n has no Ramsey k-decompositions.
Let

$$\begin{aligned} u_1 &= 1^{k-1} \in H_1^*, \\ u_n &= (u_{n-1} n)^{k-1} u_{n-1} \in H_n^* \quad \text{for every } n > 1. \end{aligned}$$

For every $n > 1$, the word u_n is defined as k copies of u_{n-1} separated by the letter n. We prove by induction that the two conditions are satisfied by each word of the sequence. The base case is immediate: the word u_1 has length $k - 1$, and as a consequence has no decomposition into k nonempty factors. Now suppose that $n > 1$, and that u_{n-1} satisfies the two properties. Then u_n has the required length:

$$|u_n| = (k-1)(|u_{n-1}| + 1) + |u_{n-1}| = (k-1)k^{n-1} + k^{n-1} - 1 = k^n - 1.$$

To conclude, we show that every k-decomposition

$$u_n = x y_1 y_2 \dots y_k z, \text{ with } y_i \in H_n^+ \text{ for all } 1 \leq i \leq k \tag{1}$$

is not Ramsey. Let y be the factor $y_1 y_2 \dots y_k$ of u_n, and consider the two following cases:

- If $\pi(y) \neq n$, none of the y_i contains the letter n, hence y is factor of one of the factors u_{n-1} of u_n. Therefore, by the induction hypothesis, Decomposition (1) is not Ramsey.
- If $\pi(y) = n$, since u_n contains only $k - 1$ copies of the letter n, one of the factors y_i does not contain n for $1 \leq i \leq k$. Then $\pi(y) \neq \pi(y_i)$, hence Decomposition (1) is not Ramsey.

▶ **Example 6.** Here are the first three words of the sequence in the cases $k = 2$ and $k = 3$:

$$\begin{aligned} k = 2: \quad & u_1 = 1 \quad && u_2 = 121 \quad && u_3 = 1213121, \\ k = 3: \quad & u_1 = 11 \quad && u_2 = 11211211 \quad && u_3 = 11211211311211211311211211. \end{aligned}$$

3.3 General setting

We saw in the previous subsection that for the max monoid H_n, words of length exponential with respect to n are required to guarantee the presence of Ramsey decompositions (Proposition 5). Note that the same lower bound applies to every monoid M that contains a copy of H_n as submonoid. We now show that we can also obtain an upper bound for $\mathsf{R}_M(k)$ by studying the submonoids of M isomorphic to a max monoid. We formalise this idea through the notion of regular $\mathcal{D}$-length of a monoid.

Regular $\mathcal{D}$-length. The *regular $\mathcal{D}$-length* of a monoid M, denoted $L(M)$, is the size of the largest max monoid embedded in M. Formally, it is the largest $\ell \in \mathbb{N}$ such that there exists a monomorphism (i.e. injective monoid homomorphism) $\varphi : H_\ell \to M$. We now present the main theorem of this section, which states that for every monoid M, the degree of $\mathsf{R}_M(k)$ is determined by the regular $\mathcal{D}$-length of M.

▶ **Theorem 1.** *Every monoid M of regular $\mathcal{D}$-length L satisfies $k^L \leq \mathsf{R}_M(k) \leq (k|M|^4)^L$.*

Let us fix for the whole subsection a monoid M of regular $\mathcal{D}$-length $L(M)$ and an integer $k \in \mathbb{N}$. The lower bound is a corollary of Proposition 5: the max monoid $H_{L(M)}$ has a witness $u_{L(M)}$ of length $k^{L(M)} - 1$ that has no Ramsey k-decompositions (its construction is presented in the previous subsection). Then, by definition of the regular $\mathcal{D}$-length, there exists a monomorphism $\varphi : H_{L(M)} \to M$, and applying φ to $u_{L(M)}$ letter by letter yields a witness $u'_{L(M)} \in M^*$ of length $k^{L(M)} - 1$ that has no Ramsey k-decompositions.

The rest of the subsection is devoted to the proof of the upper bound. We begin by defining an auxiliary algorithm that extracts from each long enough word a decomposition where the prefix and suffix absorb the middle factors. Then, we define our main algorithm which, on input $u \in M^*$ of length $(k|M|^4)^n$ for some $n \in \mathbb{N}$, either returns a Ramsey k-decomposition of u, or a copy of the max monoid H_{n+1} embedded in M. In particular, if n is equal to the regular $\mathcal{D}$-length $L(M)$ of M, we are guaranteed to obtain a Ramsey k-decomposition.

Auxiliary algorithm. We define an algorithm which, on input $u \in M^*$ of length $k|M|^2$, returns a k-decomposition

$$u = xy_1y_2\dots y_kz, \text{ where } x, z \in M^*, \text{ and } y_i \in S^+ \text{ for every } 1 \leq i \leq k$$

such that for every $1 \leq i \leq k$, both x and z are able to absorb the factor y_i: $\pi(xy_i) = \pi(x)$ and $\pi(y_iz) = \pi(z)$. This is done as follows: since u is a word of length $k|M|^2$, it can be split into $k|M|^2 + 1$ distinct prefix-suffix pairs. Then $k + 1$ of these pairs reduce to the same pair of elements of M, which immediately yields the desired decomposition. Formally,

```
ALG3: Start with $u \in M^*$ of length $k|M|^2$;

1. a. Compute the $k|M|^2 + 1$ prefixes $\pi(u[0,0]), \pi(u[0,1]), \dots, \pi(u[0,|u|]) \in M$ of $u$,
   b. Compute the $k|M|^2 + 1$ suffixes $\pi(u[0,|u|]), \pi(u[1,|u|]), \dots, \pi(u[|u|,|u|]) \in M$ of $u$,
   c. Identify $k + 1$ indices $s_0, s_1, \dots, s_k$ such that
      (1) all the $\pi(u[0,s_i])$ are equal,
      (2) all the $\pi(u[s_i,|u|])$ are equal;
2. Set $x = u[0,s_0]$, $z = u[s_k,|u|]$, and $y_i = u[s_{i-1},s_i]$ for every $1 \leq i \leq k$;

3. Return the $k$-decomposition $xy_1y_2\dots y_kz$ of $u$.
```

Main algorithm. We define an algorithm extracting Ramsey k-decompositions. Over an input $u \in M^*$ of length $(k|M|^4)^n$ for $n \in \mathbb{N}$, the algorithm works by defining gradually shorter words $u_n, u_{n-1}, \ldots \in M^*$, where each u_j has length $(k|M|^4)^j$, along with a sequence of idempotent elements $e_{n+1}, e_n, \ldots \in M$. Starting with $u_n = u$, we define e_{n+1} as the idempotent power of some well chosen factors of u_n. We then consider k consecutive factors of u_n. If all of them reduce to e_{n+1}, they form a Ramsey k-decomposition, and we are done. Otherwise, we pick a factor u_{n-1} that does not reduce to e_{n+1}, and we start over. This continues until either a Ramsey k-decomposition is found, or n cycles are completed. In the later case, we show that the function $\varphi : H_{n+1} \to M$ mapping i to e_i is a monomorphism.

```
ALG4: Start with $u \in M^*$ of length $(k|M|^4)^n$. Initialize $u_n$ to $u$ and $j$ to $n$.

- While $j > 0$, repeat the following:
  1. a. Call ALG3 to get an $m$-decomposition $u_j = xy_1y_2 \ldots y_m z$, where $m = k^j|M|^{4j-2}$;
     b. Set $v := \pi(y_1)\pi(y_2)\ldots\pi(y_m) \in M^*$;
  2. a. Call ALG3 to get an $m'$-decomposition $v = x'y'_1y'_2 \ldots y'_{m'}z'$, where $m' = k^j|M|^{4j-4}$;
     b. Set $w := \pi(y'_1)\pi(y'_2)\ldots\pi(y'_{m'}) \in M^*$, and set $e_{j+1} := (\pi(z'x'))^{\#}$;
  3. a. Split $w$ into $k$ factors $y''_1$, $y''_2$, ..., $y''_k$ of length $(k|M|^4)^{j-1}$;
     b. If every $y''_i$ satisfies $\pi(y_i) = e_{j+1}$, then $w = y''_1y''_2 \ldots y''_k$ is a Ramsey decomposition.
        Return the corresponding Ramsey $k$-decomposition of $u$;
     c. If $\pi(y''_i) \neq e_{j+1}$ for some $1 \leq i \leq n$, set $u_{j-1} := y''_i$, and decrement $j$ by 1.
- Set $e_1 = 1_M$, and return the idempotent elements $e_1, e_2, \ldots, e_{n+1} \in M$.
```

Step 1. We use the auxiliary algorithm to obtain a decomposition $u_j = xy_1y_2 \ldots y_m z$, and we build v by concatenating the reductions of the y_i. Since both x and z absorb each y_i, and in step 2b we define e_{j+1} as the idempotent power of reduced factors of v:

The word u_j, its prefix x and its suffix z satisfy $\pi(u_j) = \pi(xz) = \pi(x) \cdot e_{j+1} \cdot \pi(z)$. (1)

Step 2. We use the auxiliary algorithm to get a decomposition $u' = x'y'_1y'_2 \ldots y'_{m'}z'$, we build w by concatenating the reductions of the y'_i, and we set e_{j+1} as the idempotent power of $\pi(z'x')$. As both x' and z' absorb each y'_i, and in step 3c we define u_{j-1} as a factor of w:

For every factor y of u_{j-1}, $e_{j+1} \cdot \pi(y) = e_{j+1} = \pi(y) \cdot e_{j+1}$. (2)

Step 3. We divide w into k factors of equal length. If each of them reduces to e_{j+1}, they form a Ramsey k-decomposition of w. As w is obtained form u by iteratively reducing factors and dropping prefixes and suffixes, this decomposition can be transferred back to a Ramsey k-decomposition of $u = u_n$. If one factor does not reduce to e_{j+1}, we assign its value to u_{j-1}. Therefore:

The word u_{j-1} does not reduce to e_{j+1}. (3)

Proof of correctness. To prove that the algorithm behaves as intended, we show that if it completes n cycles without returning a Ramsey k-decomposition, then the function $\varphi : H_{n+1} \to M$ defined by $\varphi(j) = e_j$ is a monomorphism. Since e_j is the idempotent power of reduced factors of u_{j-1} for all $1 \leq j \leq n$, Equation (2) yield that $e_{j+1} \cdot e_j = e_{j+1} = e_j \cdot e_{j+1}$.

Therefore φ is a homomorphism. We conclude by showing that it is injective. Suppose, towards building a contradiction, that $\varphi(j) = e_j = e_i = \varphi(i)$ for some $1 \leq j < i \leq n$. Since φ is a homomorphism, all the intermediate elements collapse: in particular $e_j = e_{j+1}$. Then

$$\pi(u_{j-1}) \underset{(1)}{=} \pi(x) \cdot e_j \cdot \pi(z) = \pi(x) \cdot e_{j+1} \cdot \pi(z) \underset{(2)}{=} e_{j+1},$$

which cannot hold by Equation (3).

4 Regular $\mathcal{D}$-length of the monoid of Boolean matrices

A Boolean matrix is a matrix A whose components are Boolean elements: $A_{ij} \in \{0,1\}$. The (full) *Boolean matrix monoid* B_n is the set of all $n \times n$ Boolean matrices, equipped with the matrix composition defined as follows: $(A \cdot B)_{ik} = 1$ if and only if there exists $j \in [1,n]$ satisfying $A_{ij} = B_{jk} = 1$. This fits the standard matrix multiplication if we consider that $1 + 1 = 1$: addition of Boolean elements is the OR operation, and multiplication is the AND operation. The main contribution of this section is the following theorem.

▶ **Theorem 2.** *The regular $\mathcal{D}$-length of the monoid of $n \times n$ Boolean matrices is $\frac{n^2+n+2}{2}$.*

The proof is split in two parts. We prove the upper bound by studying the structure of the idempotent elements of B_n (Subsection 4.1). Then, we prove the lower bound by constructing a monomorphism from the max monoid of size $\frac{n^2+n+2}{2}$ into B_n (Subsection 4.2). We begin by introducing definitions tailored to help us in the following demonstrations.

Stable matrix. A Boolean matrix $A \in B_n$ is called *stable* if for each component A_{ik} equal to 1, there exists $j \in [1,n]$ satisfying $A_{ij} = A_{jj} = A_{jk} = 1$. Idempotent matrices are stable (see the full version).

Positive set. A (maximal) *positive set* of an idempotent matrix $A \in B_n$ is a maximal set $I \subseteq [1,n]$ such that all the corresponding components of A are 1: $A_{ij} = 1$ for all $i,j \in I$, and for every $k \in [1,n] \setminus I$, there exists $i \in I$ such that $A_{ik} = 0$ or $A_{ki} = 0$. The positive sets of an idempotent matrix are disjoint (see the full version), hence A has at most n positive sets.

Free pair. For each idempotent matrix $A \in B_n$ we define the relation $\to_A$ on $[1,n]$ as follows: given $i,j \in [1,n]$, we have $i \to_A j$ if for all $i_2, j_2 \in [1,n]$, $A_{i_2 i} = 1 = A_{j j_2}$ implies $A_{i_2 j_2} = 1$. A *free pair* of A is a set of two distinct elements $i,j \in [1,n]$ incomparable by $\to_A$: $i \not\to_A j$ and $j \not\to_A i$. Note that A has at most $\frac{n(n-1)}{2}$ free pairs (all sets of two distinct elements in $[1,n]$). Let us state some observations concerning $\to_A$ that follow immediately from the definition (see the full version for the proofs). First, as A is idempotent, $\to_A$ is reflexive. However, it might not be transitive. Moreover, for every component A_{ij} of A equal to 1, we have that $i \to_A j$. The converse implication is not true, as shown by the following example. Finally, for every $i \in [1,n]$, if the i^{th} row contains no 1, i.e., $A_{ik} = 0$ for all $k \in [1,n]$, then $i \to_A j$ for every $j \in [1,n]$. Conversely, if the i^{th} column contains no 1, then $j \to_A i$ for every $j \in [1,n]$.

▶ **Example 7.** We depict below a submonoid of B_4 generated by two matrices A and B. The six elements of this submonoid, including the identity matrix $D \in B_n$, are all idempotent. Under each matrix, we list its positive sets. We then compute the corresponding free pairs.

$\begin{matrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{matrix}$	$\begin{matrix}1&0&1&0\\0&1&0&1\\1&0&1&0\\0&1&0&1\end{matrix}$	$\begin{matrix}1&1&1&1\\0&0&0&1\\0&0&0&1\\0&0&0&1\end{matrix}$	$\begin{matrix}1&1&1&1\\0&0&0&1\\1&1&1&1\\0&0&0&1\end{matrix}$	$\begin{matrix}1&1&1&1\\0&1&0&1\\0&1&0&1\\0&1&0&1\end{matrix}$	$\begin{matrix}1&1&1&1\\0&1&0&1\\1&1&1&1\\0&1&0&1\end{matrix}$
D	A	B	$A \cdot B$	$B \cdot A$	$A \cdot B \cdot A$
$\{1\}, \{2\}, \{3\}, \{4\}$	$\{1, 3\}, \{2, 4\}$	$\{1\}, \{4\}$	$\{1, 3\}, \{4\}$	$\{1\}, \{2, 4\}$	$\{1, 3\}, \{2, 4\}$

Every pair is free in D since the relation $\rightarrow_D$ is the identity: given two distinct elements $i, j \in [1, n]$, we have $D_{ii} = 1 = D_{jj}$, yet $D_{ij} = 0$, hence $i \not\rightarrow_D j$. On the contrary, the four matrices B, $A \cdot B$, $B \cdot A$ and $A \cdot B \cdot A$ has no free pairs: the relation $\rightarrow_B$ only lacks $(4, 1)$, $\rightarrow_{A \cdot B}$ only lacks $(4, 1)$ and $(4, 3)$, $\rightarrow_{B \cdot A}$ only lacks $(2, 1)$ and $(4, 1)$, $\rightarrow_{A \cdot B \cdot A}$ only lacks $(2, 1)$, $(4, 1)$ and $(4, 3)$. Finally, for A, the relation $\rightarrow_A$ is the union of the identity and the four pairs $\{(1, 3), (3, 1), (2, 4), (4, 2)\}$, which yields the free pairs $\{1, 2\}$, $\{1, 4\}$, $\{2, 3\}$ and $\{3, 4\}$.

4.1 Upper bound

To prove the upper bound of Theorem 2, we show that every monomorphism $\varphi : H_m \rightarrow B_n$ satisfies $m \leq \frac{n^2+n+2}{2}$. To this end, we study the sequence of matrices $s_\varphi = A_1, A_2, \ldots, A_m$ obtained by listing the elements $\varphi(i) = A_i$ of the image of φ. Note that all the elements of s_φ are distinct as φ is injective, and $A_i \cdot A_{i+1} = A_{i+1} = A_{i+1} \cdot A_i$ for all $1 \leq i < m$ as φ is a homomorphism. We introduce three lemmas that imply interesting properties of every pair A_i, A_{i+1} of successive matrices of s_φ. First, Lemma 8 shows that every positive set of A_{i+1} contains a positive set of A_i. Therefore, since positive sets are disjoint, the number of positive sets can never increase along s_φ. Second, Lemma 9 shows that every free pair of A_{i+1} is also a free pair of A_i. As a consequence, the number of free pairs can never increase along s_φ. Finally, Lemma 10 shows that either the number of positive sets or free pairs differs between A_i and A_{i+1}, as otherwise these two matrices would be equal.

Combining the three lemmas yields that between each pair of successive matrices of s_φ, neither the number of positive sets nor the number of free pairs increases, and at least one decreases. This immediately implies the desired upper bound: as the number of positive sets of matrices of B_n ranges from 0 to n and the number of free pairs ranges from 0 to $\frac{n(n-1)}{2}$, s_φ contains at most $n + \frac{n(n-1)}{2} + 1 = \frac{n^2+n+2}{2}$ matrices. To conclude, we now proceed with the formal statements and the proofs of the three lemmas.

▶ **Lemma 8.** *Let A and B be two idempotent matrices of B_n satisfying $A \cdot B = B = B \cdot A$. Then every positive set of B contains a positive set of A.*

Proof. Let us pick two idempotent matrices $A, B \in B_n$ satisfying $A \cdot B = B = B \cdot A$. If B has no positive sets, the statement is trivially satisfied. Now let us suppose that B has at least one positive set $I \subseteq [1, n]$. We show the existence of a positive set $J \subseteq I$ of A.

Since I is not empty by definition, it contains an element i, and $B_{ii} = 1$. Then, as $B = B \cdot A$, there exists $k \in [1, n]$ satisfying $B_{ik} = A_{ki} = 1$. Moreover, as A is stable, there exists $j \in [1, n]$ satisfying $A_{kj} = A_{jj} = A_{ji} = 1$. In particular, $A_{jj} = 1$, hence A has a positive set J containing j. Then, for every $i_2 \in I$ and every $j_2, j_3 \in J$, we obtain

$$\begin{aligned}
&B_{j_2 i_2} = (A \cdot A \cdot B)_{j_2 i_2} = 1 \text{ since } A_{j_2 j} = A_{ji} = B_{i i_2} = 1,\\
&B_{i_2 j_3} = (B \cdot B \cdot A \cdot A)_{i_2 j_3} = 1 \text{ since } B_{i_2 i} = B_{ik} = A_{kj} = A_{j j_3} = 1,\\
&B_{j_2 j_3} = (B \cdot B)_{j_2 j_3} = 1 \text{ since } B_{j_2 i_2} = B_{i_2 j_3} = 1.
\end{aligned}$$

As a consequence, J is a subset of I since positive sets are maximal by definition. ◀

▶ **Lemma 9.** *Let A and B be two idempotent matrices of B_n satisfying $A \cdot B = B = B \cdot A$. Then every free pair of B is a free pair of A.*

Proof. Let us pick two idempotent matrices $A, B \in B_n$ satisfying $A \cdot B = B = B \cdot A$. We prove the lemma by contraposition: we show that for every pair of elements $i, j \in [1, n]$, $i \to_A j$ implies $i \to_B j$ (hence if i and j are incomparable by $\to_B$, so are they by $\to_A$).

Let us pick $i, j \in [1, n]$ satisfying $i \to_A j$, and $i_2, j_2 \in [1, n]$ satisfying $B_{i_2 i} = 1 = B_{jj_2}$. To conclude, we show that $B_{i_2 j_2} = 1$. To this end, we introduce two new elements $i_1, j_1 \in [1, n]$: First, as $(B \cdot A)_{i_2 i} = B_{i_2 i} = 1$, there exists $i_1 \in [1, n]$ such that $B_{i_2 i_1} = 1$ and $A_{i_1 i} = 1$; Second, as $(A \cdot B)_{jj_2} = B_{jj_2} = 1$, there exists $j_1 \in [1, n]$ such that $A_{jj_1} = 1$ and $B_{j_1 j_2} = 1$. Then, as $i \to_A j$ by supposition, we get that $A_{i_1 j_1} = 1$, which implies

$$B_{i_2 j_2} = (B \cdot A \cdot B)_{i_2 j_2} = 1, \text{ since } B_{i_2 i_1} = A_{i_1 j_1} = B_{j_1 j_2} = 1.$$

Since this holds for every $i_2, j_2 \in [1, n]$ satisfying $B_{i_2 i} = 1 = B_{jj_2}$, we obtain that $i \to_B j$. ◀

▶ **Lemma 10.** *Let A and B be two idempotent matrices of B_n satisfying $A \cdot B = B = B \cdot A$. If A and B have the same number of positive sets and free pairs, then they are equal.*

Proof. Let us pick two idempotent elements $A, B \in B_n$ such that $A \cdot B = B = B \cdot A$. Suppose that A and B have the same number of positive sets. By Lemma 8, each positive set of B contains at least one positive set of A. Since the positive sets of B are disjoint, the pigeonhole principle yields the two following claims.

▷ Claim 1. Each positive set of A is contained in a positive set of B.

▷ Claim 2. Each positive set of B contains exactly one positive set of A.

Moreover, suppose that A and B have the same number of free pairs. By Lemma 9 every free pair of B is a free pair of A. This yields the following claim.

▷ Claim 3. The free pairs of A and B are identical.

We now prove that $A = B$. First, we show that for every component A_{ik} equal to 1, the corresponding component B_{ik} is also equal to 1. Since A is stable, there exists $j \in [1, n]$ satisfying $A_{ij} = A_{jj} = A_{jk} = 1$. Then j is contained in a positive set of A, which is itself contained in a positive set of B by Claim 1. Therefore we obtain that $B_{jj} = 1$, which yields

$$B_{ik} = (A \cdot B \cdot A)_{ik} = 1, \text{ since } A_{ij} = B_{jj} = A_{jk} = 1.$$

To conclude, we show that for every component B_{ij} equal to 1, the corresponding component A_{ij} is also equal to 1. To this end, we introduce four new elements i_1, i_2, j_1, j_2 in $[1, n]$: First, as $(A \cdot B \cdot A)_{ij} = B_{ij} = 1$, there exist $i_2, j_2 \in [1, n]$ such that $A_{ii_2} = B_{i_2 j_2} = A_{j_2 j} = 1$. Second, as A is stable, there exist $i_1, j_1 \in [1, n]$ such that $A_{ii_1} = A_{i_1 i_1} = A_{i_1 i_2} = 1$ and $A_{j_2 j_1} = A_{j_1 j_1} = A_{j_1 j} = 1$. These definitions ensure that

$$B_{i_1 j_1} = (A \cdot B \cdot A)_{i_1 j_1} = 1, \text{ since } A_{i_1 i_2} = B_{i_2 j_2} = A_{j_2 j_1} = 1.$$

Note that, as observed after the definition of the relation induced by an idempotent matrix, this implies that $i_1 \to_B j_1$. We derive from this that either $i_1 \to_A j_1$ or $j_1 \to_A i_1$: if $i_1 = j_1$ this follows from the fact that $\to_A$ is reflexive, and if $i_1 \neq j_1$ this follows from Claim 3. We show that both possibilities lead to $A_{ij} = 1$.

- If $i_1 \to_A j_1$, then we obtain $A_{i_1 j_1} = 1$ as $A_{i_1 i_1} = 1 = A_{j_1 j_1}$. Therefore,

$$A_{ij} = (A \cdot A \cdot A)_{ij} = 1 \text{ since } A_{ii_1} = A_{i_1 j_1} = A_{j_1 j} = 1.$$

- If $j_1 \rightarrow_A i_1$, then we obtain $A_{j_1 i_1} = 1$ as $A_{j_1 j_1} = 1 = A_{i_1 i_1}$. Therefore,

  $$B_{j_1 i_1} = (A \cdot B \cdot A)_{j_1 i_1} = 1 \text{ since } A_{j_1 i_1} = B_{i_1 j_1} = A_{j_1 i_1} = 1.$$

  As a consequence, i_1 and j_1 are in the same positive set of B. Moreover, as $A_{i_1 i_1} = A_{j_1 j_1} = 1$, both i_1 and j_1 are elements of positive sets of A. Combining these two statements with Claim 2 yields that i_1 and j_1 are in the same positive set of A. Therefore $A_{i_1 j_1} = 1$, which implies that $i_1 \rightarrow_A j_1$, and we can conclude as in the previous point.

Since we successfully showed that every 1 of A corresponds to a 1 of B, and reciprocally, we obtain that $A = B$, which proves the statement. ◀

4.2 Lower bound

We construct a monomorphism φ between the max monoid $H_{f(n)}$, where $f(n) = \frac{n^2+n+2}{2}$, and the monoid of Boolean matrices B_n. The construction is split in two steps. First, we define φ over the domain $[1, g(n)+1]$, where $g(n) = \frac{n(n-1)}{2}$ is the number of pairs of elements $i < j$ in $[1, n]$. Then, we complete the definition over the domain $[g(n)+1, f(n)]$.

Diagonal to triangular. Let us define φ over $[1, g(n)+1]$. We map the neutral element $1 \in H_{f(n)}$ to the neutral element $D_n \in B_n$: the identity matrix. Then, we map $g(n)+1 \in H_{f(n)}$ to the full upper triangular matrix $U_n \in B_n$. Note that U_n contains $g(n)$ more 1's than D_n does. We define the images of the elements between 1 and $g(n)+1$ by gradually adding to D_n the 1's of U_n it lacks. Formally, we order the indices corresponding to the components above the diagonal $p_1 < p_2 < \ldots < p_{g(n)} \in [1, n] \times [1, n]$ according to the lexicographic order: (i, j) comes before (i', j') if either $i < i'$, or $i = i'$ and $j < j'$. Then, for every $m \in [1, g(n)+1]$, we construct the image $\varphi(m) \in B_n$ as follows:

- Every component $(\varphi(m))_{ii}$ of the diagonal is 1;
- Every component $(\varphi(m))_{ij}$ below the diagonal is 0;
- Every component $(\varphi(m))_{ij}$ above the diagonal is 1 if $(i, j) < p_m$, and 0 otherwise.

Triangular to empty. Let us define φ over $[g(n)+1, f(n)]$. To fit the first part of the definition, we map $g(n)+1 \in H_{f(n)}$ to the upper diagonal matrix $U_n \in B_n$. Then, we map the absorbing element $f(n) = g(n)+1+n \in H_{f(n)}$ to the absorbing element $0_n \in B_n$: the null matrix. Finally, for $m \in [0, n]$, we construct $\varphi(g(n)+1+m)$ by replacing the last m rows of U_n with 0's. Formally, we have:

- Every component $(\varphi(g(n)+1+m))_{ij}$ is 1 if $i \leq j$ and $i \leq n-m$, and 0 otherwise.

Proof of correctness. We prove that the function φ just defined is a monomorphism.

We show that φ is a homomorphism: $\varphi(m) \cdot \varphi(m') = \varphi(m') = \varphi(m') \cdot \varphi(m)$ for all $1 \leq m \leq m' \leq f(n)$. First, note that if $(\varphi(m'))_{ij} = 1$, then $(\varphi(m) \cdot \varphi(m'))_{ij} = (\varphi(m') \cdot \varphi(m))_{ij} = 1$: if $m \leq g(n)+1$, this follows from the fact that the diagonal of $\varphi(m)$ is filled with 1's, and if $m > g(n)+1$, since $m \leq m'$ we obtain that $(\varphi(m))_{ii} = (\varphi(m'))_{ij} = 1 = (\varphi(m'))_{ii} = (\varphi(m))_{ij}$. It remains to show that if $(\varphi(m) \cdot \varphi(m'))_{ik} = 1$ or $(\varphi(m') \cdot \varphi(m))_{ik} = 1$, then $(\varphi(m'))_{ik} = 1$. If $m' \leq g(n)+1$, this holds since for every triple $i \leq j \leq k \in [1, n]$, the pair (i, k) is lexicographically smaller than or equal to (j, k). If $m' > g(n)+1$, this holds since for every triple $i \leq j \leq k \in [1, n]$, trivially i is smaller than or equal to both i and j.

We conclude by showing that φ is injective: between $\varphi(1)$ and $\varphi(g(n)+1)$ a new 1 is added at each step, and between $\varphi(g(n)+1)$ and $\varphi(f(n))$ we remove at each step a 1 of the diagonal that was present in all the previous images.

▶ **Example 11.** We depict the monomorphism $\varphi : H_{f(n)} \to B_n$ in the case $n = 4$ by listing the $f(4) = 11$ elements of its image in B_4. Under each element, we state its number of positive sets followed by its number of free pairs.

$$\underset{(4,6)}{\begin{matrix}\mathsf{I}&\circ&\circ&\circ\\\circ&\mathsf{I}&\circ&\circ\\\circ&\circ&\mathsf{I}&\circ\\\circ&\circ&\circ&\mathsf{I}\end{matrix}}\quad\underset{(4,5)}{\begin{matrix}\mathsf{I}&\mathsf{I}&\circ&\circ\\\circ&\mathsf{I}&\circ&\circ\\\circ&\circ&\mathsf{I}&\circ\\\circ&\circ&\circ&\mathsf{I}\end{matrix}}\quad\underset{(4,4)}{\begin{matrix}\mathsf{I}&\mathsf{I}&\mathsf{I}&\circ\\\circ&\mathsf{I}&\circ&\circ\\\circ&\circ&\mathsf{I}&\circ\\\circ&\circ&\circ&\mathsf{I}\end{matrix}}\quad\underset{(4,3)}{\begin{matrix}\mathsf{I}&\mathsf{I}&\mathsf{I}&\mathsf{I}\\\circ&\mathsf{I}&\circ&\circ\\\circ&\circ&\mathsf{I}&\circ\\\circ&\circ&\circ&\mathsf{I}\end{matrix}}\quad\underset{(4,2)}{\begin{matrix}\mathsf{I}&\mathsf{I}&\mathsf{I}&\mathsf{I}\\\circ&\mathsf{I}&\mathsf{I}&\circ\\\circ&\circ&\mathsf{I}&\circ\\\circ&\circ&\circ&\mathsf{I}\end{matrix}}\quad\underset{(4,1)}{\begin{matrix}\mathsf{I}&\mathsf{I}&\mathsf{I}&\mathsf{I}\\\circ&\mathsf{I}&\mathsf{I}&\mathsf{I}\\\circ&\circ&\mathsf{I}&\circ\\\circ&\circ&\circ&\mathsf{I}\end{matrix}}\quad\underset{(4,0)}{\begin{matrix}\mathsf{I}&\mathsf{I}&\mathsf{I}&\mathsf{I}\\\circ&\mathsf{I}&\mathsf{I}&\mathsf{I}\\\circ&\circ&\mathsf{I}&\mathsf{I}\\\circ&\circ&\circ&\mathsf{I}\end{matrix}}\quad\underset{(3,0)}{\begin{matrix}\mathsf{I}&\mathsf{I}&\mathsf{I}&\mathsf{I}\\\circ&\mathsf{I}&\mathsf{I}&\mathsf{I}\\\circ&\circ&\mathsf{I}&\mathsf{I}\\\circ&\circ&\circ&\circ\end{matrix}}\quad\underset{(2,0)}{\begin{matrix}\mathsf{I}&\mathsf{I}&\mathsf{I}&\mathsf{I}\\\circ&\mathsf{I}&\mathsf{I}&\mathsf{I}\\\circ&\circ&\circ&\circ\\\circ&\circ&\circ&\circ\end{matrix}}\quad\underset{(1,0)}{\begin{matrix}\mathsf{I}&\mathsf{I}&\mathsf{I}&\mathsf{I}\\\circ&\circ&\circ&\circ\\\circ&\circ&\circ&\circ\\\circ&\circ&\circ&\circ\end{matrix}}\quad\underset{(0,0)}{\begin{matrix}\circ&\circ&\circ&\circ\\\circ&\circ&\circ&\circ\\\circ&\circ&\circ&\circ\\\circ&\circ&\circ&\circ\end{matrix}}$$

Starting with the identity matrix D_4, we gradually add 1's, reaching the triangular matrix U_4 in $g(4) = 6$ steps. Then, we erase line after line, reaching the null matrix 0_4 in 4 steps.

References

1 Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. One-way definability of two-way word transducers. *Logical Methods in Computer Science*, 14, 2018. doi:10.23638/LMCS-14(4:22)2018.

2 Michael Breen. A maximal chain of principal ideals in the semigroup of binary relations on a finite set. In *Semigroup Forum*, volume 43, pages 63–76. Springer, 1991.

3 Olexandr Ganyushkin and Volodymyr Mazorchuk. *Classical finite transformation semigroups: an introduction*, volume 9. Springer Science & Business Media, 2008.

4 T. E. Hall and Mark V. Sapir. Idempotents, regular elements and sequences from finite semigroups. *Discrete Mathematics*, 161:151–160, 1996. doi:10.1016/0012-365X(95)00223-J.

5 Shaofang Hong. Distribution of cardinalities of row spaces of boolean matrices of order n. *Southeast Asian Bulletin of Mathematics*, 24:51–64, 2000.

6 Janusz Konieczny. On cardinalities of row spaces of boolean matrices. In *Semigroup Forum*, volume 44, pages 393–402. Springer, 1992.

7 Wen Li and Mou-Cheng Zhang. On konieczny's conjecture of boolean matrices. In *Semigroup Forum*, volume 50, pages 37–58. Springer, 1995.

8 Filip Mazowiecki and Cristian Riveros. Pumping lemmas for weighted automata. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018*, volume 96 of *LIPIcs*, pages 50:1–50:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.50.

9 A. Mukherjea and R. Chaudhuri. Idempotent boolean matrices. *Semigroup forum*, 21:273–282, 1980. URL: http://eudml.org/doc/134452.

10 Anca Muscholl and Gabriele Puppis. The many facets of string transducers (invited talk). In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019*, volume 126 of *LIPIcs*, pages 2:1–2:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.2.

11 Jean-Éric Pin. Mathematical foundations of automata theory. *Lecture notes LIAFA, Université Paris*, 7, 2010.

12 Michael O. Rabin and Dana S. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:114–125, 1959. doi:10.1147/rd.32.0114.

13 F. P. Ramsey. On a problem of formal logic. *Proceedings of the London Mathematical Society*, s2-30(1):264–286, 1930. doi:10.1112/plms/s2-30.1.264.

14 Marcel-Paul Schützenberger. Une théorie algébrique du codage. *Séminaire Dubreil. Algebre et théorie des nombres*, 9:1–24, 1955.

15 Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965. doi:10.1016/S0019-9958(65)90108-7.

16 Imre Simon. Factorization forests of finite height. *Theor. Comput. Sci.*, 72:65–94, 1990. doi:10.1016/0304-3975(90)90047-L.

17 M-C Zhang, S-F Hong, and H-B Kan. On the cardinalities of the row spaces of non-full rank boolean matrices. In *Semigroup Forum*, volume 59, pages 152–154. Springer, 1999.

An Improved Sketching Algorithm for Edit Distance

Ce Jin
MIT, Cambridge, MA, USA

Jelani Nelson
University of California at Berkeley, CA, USA

Kewen Wu
University of California at Berkeley, CA, USA

Abstract

We provide improved upper bounds for the simultaneous sketching complexity of edit distance. Consider two parties, Alice with input $x \in \Sigma^n$ and Bob with input $y \in \Sigma^n$, that share public randomness and are given a promise that the edit distance $\mathsf{ed}(x, y)$ between their two strings is at most some given value k. Alice must send a message sx and Bob must send sy to a third party Charlie, who does not know the inputs but shares the same public randomness and also knows k. Charlie must output $\mathsf{ed}(x, y)$ precisely as well as a sequence of $\mathsf{ed}(x, y)$ edits required to transform x into y. The goal is to minimize the lengths $|sx|, |sy|$ of the messages sent.

The protocol of Belazzougui and Zhang (FOCS 2016), building upon the random walk method of Chakraborty, Goldenberg, and Koucký (STOC 2016), achieves a maximum message length of $\tilde{O}(k^8)$ bits, where $\tilde{O}(\cdot)$ hides $\mathrm{poly}(\log n)$ factors. In this work we build upon Belazzougui and Zhang's protocol and provide an improved analysis demonstrating that a slight modification of their construction achieves a bound of $\tilde{O}(k^3)$.

2012 ACM Subject Classification Theory of computation → Sketching and sampling

Keywords and phrases edit distance, sketching

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.45

Related Version *Full Version*: `https://arxiv.org/abs/2010.13170`

Funding *Ce Jin*: Supported by Akamai Presidential Fellowship.
Jelani Nelson: Supported by NSF award CCF-1951384, ONR grant N00014-18-1-2562, ONR DORECG award N00014-17-1-2127, an Alfred P. Sloan Research Fellowship, and a Google Faculty Research Award.

Acknowledgements We thank Qin Zhang for answering several questions about [7]. C. J. thanks Virginia Vassilevska Williams for several helpful discussions. We thank anonymous reviewers for their helpful comments.

1 Introduction

The edit distance $\mathsf{ed}(x, y)$ between two strings is defined to be the minimum number of character insertions, deletions, or substitutions required to transform x into y. It is one of the most well-studied distance measures on strings, with applications in information retrieval, natural language processing, and bioinformatics. If x, y are each at most length n, the textbook Wagner-Fischer algorithm computes $\mathsf{ed}(x, y)$ exactly in $O(n^2)$ time, with the only improvement since being by a $\log n$ factor due to Masek and Paterson [23]. It has since been shown that an $O(n^{2-\epsilon})$ time algorithm does not exist for any constant $\epsilon > 0$ unless the *Strong Exponential Time Hypothesis* fails [5]. Since the work of [23], several subsequent works have considered setups beyond offline exact algorithms for edit distance, such as faster approximation algorithms [4, 2, 12, 11, 21, 3], metric embeddings [26, 14, 19, 22], smoothed complexity [1, 9], quantum algorithms [8], sublinear time algorithms

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 45; pp. 45:1–45:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

for gap versions [6, 17, 10, 20], and communication complexity and sketching/streaming [13, 7, 15, 18, 16]. In this work we focus on communication complexity, and specifically *simultaneous communication complexity.*

In the communication model, Alice has input string x and Bob has y. They, or a third party, would like to compute $\mathsf{ed}(x, y)$ as well as a minimum length sequence of edits for transforming x into y. We consider the setting of shared public randomness amongst all parties. The one-way setting in which Alice sends a single message to Bob, who must then output $\mathsf{ed}(x, y)$, is known as the *document exchange problem* and has a long history. In the promise version of the problem for which we are promised $\mathsf{ed}(x, y) \leq k$, Orlitsky [25] gave a deterministic protocol in which Alice only sends $O(k \log(n/k))$ bits in the case of binary strings, which is optimal, with the downside that Bob's running time to process her message is exponential. Haeupler recently used public randomness to improve Bob's running time to polynomial with the same asymptotic message length, and it is now known that a polynomial-time recovery algorithm is achievable deterministically if one increases the message length to $O(k \log^2(n/k))$ [15, 18]. Belazzougui and Zhang [7] studied the harder *simultaneous communication* model in which Alice and Bob each send messages to a third party Charlie, who knows neither string but shares knowledge of the public randomness, and Charlie must output $\mathsf{ed}(x, y)$ as well as the edits required to transform x into y. In this model they gave a protocol in which each player sends $O(k^8 \log^5 n) = \tilde{O}(k^8)$ bits.[1]

▶ **Definition 1** (Problem $\mathcal{Q}_{n,k,\delta}$). *Alice and Bob and a referee share public randomness. Alice (resp., Bob) gets a length-n input string x (resp., y) over alphabet Σ, and then sends a "sketch" $sx \in \{0, 1\}^*$ (resp., sy) to the referee. We say the size of the sketch is maximum length of strings sx and sy. After receiving the sketches sx and sy,*

- *if $\mathsf{ed}(x, y) \leq k$, the referee needs to compute $\mathsf{ed}(x, y)$ as well as an optimal edit sequence from x to y, with success probability at least $1 - \delta$;*
- *if $\mathsf{ed}(x, y) > k$, the referee needs to report "error", with success probability at least $1 - \delta$.*

Main contribution

We build upon and improve techniques developed in [7] to show that a very slight modification of their protocol needs a sketch size of only $\tilde{O}(k^3)$ bits to solve problem $\mathcal{Q}_{n,k,\delta}$. More precisely, the bound is $O(k^3 \log^2(n/\delta) \log n)$ bits.[2]

1.1 Proof Overview

We provide a high-level description of the previous results [13, 7] that we build on, and then briefly describe our new ideas.

CGK random walk

The previous sketching result [7] uses a random walk technique developed in [13]. Given two input strings x, y of length n, we append them with infinitely many zeros and initialize two pointers $i = 1, j = 1$. In each step t, we first append $x[i]$ to Alice's output tape (and append $y[j]$ to Bob's output tape), and then increment i by $r_t(x[i]) \in \{0, 1\}$, and increment j by $r_t(y[j]) \in \{0, 1\}$, where $r_t \colon \Sigma \to \{0, 1\}$ is a random function. The process continues for

[1] We use $\tilde{O}(f)$ throughout this paper to denote $f \cdot \mathrm{polylog}(n)$.

[2] We remark that both the algorithm of [7] and our improved algorithm are time-efficient, and work in the more restrictive setting where Alice and Bob have only $\mathrm{poly}(k \log(n/\delta))$ memory and receive the input strings in a *streaming* fashion.

$3n$ steps and we consider the evolution of $i - j$, i.e., the distance between the two pointers during this random process. Observe that when $x[i] \neq y[j]$, the change of $i - j$ is a mean-zero random variable in $\{-1, 0, +1\}$ (and we call this a *progress step*); while when $x[i] = y[j]$, the difference $i - j$ will not change.

The main result of [13] shows that the number of progress steps in this random process is at least $\mathtt{ed}(x, y)/2$, and at most $O(\mathtt{ed}(x, y)^2)$ with constant probability. This property was used to design a sketching protocol (with public randomness for generating r_t) for estimating $\mathtt{ed}(x, y)$ up to a quadratic factor error by applying an approximate Hamming distance sketching protocol to the two strings generated by the random walk (where a progress step corresponds to a Hamming mismatch between Alice's and Bob's output strings).

Previous algorithm

The key idea of [7] is the following. A CGK random walk naturally induces a non-intersecting matching between the input strings: we view x and y as a bipartite graph, where if (i, j) is an edge then $x[i] = y[j]$ and i, j are the pointers in some step of the walk. In particular, this matching can be viewed as an edit sequence where a character is unchanged if it is matched.

Using an exact Hamming sketch protocol (with sketch size near-linear in the number of Hamming errors), the referee can recover this matching, as well as all the unmatched characters. Although this matching may not correspond to an edit sequence of optimal length, [7] shows that if we obtain *multiple* matchings by running i.i.d. CGK random walks, then

(a) if the *intersection* of their matched edges is contained in an optimal matching, then one can extract enough information from the matchings and unmatched characters to recover an optimal edit sequence using dynamic programming;

(b) if we generate $\mathrm{poly}(\mathtt{ed}(x, y), \log n)$ many i.i.d. CGK random walks, then the precondition of Item (a) is satisfied with constant probability.

Our improvements

We obtain our result by improving the dependence on $\mathtt{ed}(x, y)$ in Item (b) described above. In particular, we reduce the number of required random walks. Our improvements come from two parts.

To obtain the first improvement, we observe that the algorithm of [7] relies on the following two events happening. The first is that, for every edge that does appear in a (fixed) optimal matching, there should be one of the sampled CGK random walks that misses this edge. The second is that the CGK random walks should have few progress steps. In [7], they pay a union bound over the two events to make sure *all CGK random walks* are good for the decoder. This introduces a large dependence on $\mathtt{ed}(x, y)$, mainly due to the fact that the number of Hamming errors in a CGK random walk has a heavy-tailed distribution. We manage to avoid this by arguing that these two events happen simultaneously (see Lemma 15) with decent probability, and then modifying the decoding algorithm to only consider *those good CGK random walks.*

The second improvement comes from improved analysis for Item (b), which depends on the following property of the CGK random walk [7, Lemma 16] (see Subsection 3.3 for how this property can be used): informally, if a string $X[1..L]$ has a certain kind of self-similarity (for example, it is periodic), then with some nontrivial probability, a CGK random walk on X itself starting with two pointers $i = 2, j = 1$ will not pass through the state $(i = L, j = L)$. To be more precise, if there is a non-intersecting matching between $X[1..L]$

and itself, where every matched edge (I, J) satisfies $I > J$, and the number of singletons (unmatched characters) is at most K, then the CGK random walk will miss $(i = L, j = L)$ with $\Omega(1/K^2)$ probability.[3]

We use a more technical analysis to improve the bound to $\Omega(1/K)$ (see Proposition 18). Now we informally describe our main idea. Starting from the state $(i = 2, j = 1)$, with at least $\Omega(1/K)$ probability it will first reach a state (i, j) with $i - j > d_0 = \Theta(K)$ before reaching $i - j = 0$ (note that $i - j$ can never become negative). Then we will show that with good probability $i - j$ will remain in the range $[d_0/2, 3d_0/2]$. To do this, we show an $O(d_0^2)$ upper bound on the expected total number of progress steps, and use the fact that the expected deviation produced by a P-step one-dimensional random walk is $O(\sqrt{P})$.

To bound the expected total number of progress steps, we divide the evolution of the state (i, j) into several phases, where in each phase the pointers move from a *stable state* (i, j) to another *stable state* (i', j'), satisfying $j' \geq i$ and $i' \geq 2j - i$. Here, a *stable state* (i, j) informally means that we have a good upper bound of $\mathsf{ed}(X[j..i-1], X[i..2i-j-1])$ in terms of the number of singletons in the range $[j..2i-j-1]$ (for example, if X is "close" to a string with period p, and $i - j$ is approximately a multiple of p, then (i, j) is a stable state). We will bound the expected number of progress steps in one phase by $O\left((i-j) \cdot S + S^2\right)$, where S denotes the number of singletons in the range $[j..i'-1]$. We can see the sum of S over all phases is at most $2K$ since each singleton is counted at most twice. Hence, summing up over all phases would give the desired $O(K^2)$ upper bound, if we assume $i - j = \Theta(d_0)$. Although this assumption may lead to circular reasoning, we can get around this issue by a more careful argument.

Organization

We give several needed definitions in Section 2. In Section 3 we state and analyze our sketching algorithm, which as mentioned, is mostly similar to [7] but with small modifications. Section 4 is devoted to our main technical lemma. In comparison with the proof overview, Section 3 is for the first improvement and Section 4 is for the second improvement. Then we discuss limits on our approach and further problems in Section 5. The lower bounds and missing proofs can be found in full version.

2 Preliminaries

In this section we introduce formal definitions.

2.1 Notations

Let $[n]$ denote $\{1, 2, \ldots, n\}$, and let $[l..r]$ denote $\{l, l+1, \ldots, r\}$. Let $\circ$ denote string concatenation. Let $\mathbb{N}$ denote the set of natural numbers $\{0, 1, \ldots\}$. We consider sketching protocols for strings in Σ^n in this work, where Σ denotes the alphabet. We assume $|\Sigma| \leq \mathrm{poly}(n)$ and $0 \in \Sigma$.[4]

[3] There is a subtle gap in the proof of [7, Lemma 16]. On page 18 of their full version, they bounded the number of progress steps in two cases: (1) at least one of the pointers is not in any cluster, and (2) both of the two pointers are in the same cluster. (Their terminology *cluster* refers to a contiguous sequence of matched edges with no singletons in-between.) However, they did not analyze the case where the two pointers are separated in different clusters, and it was not clear to us how to repair that gap using the techniques developed in [7].

[4] For larger alphabet the algorithm still works but some $\log n$ terms in the bounds become $\log|\Sigma|$. For example, the sketch size will be $O\left(k^3 \log(n|\Sigma|/\delta) \log(n/\delta) \log n\right)$. Alternatively, the parties can hash Σ into a new alphabet of size $O(n^2/\gamma)$ and have no hash collisions on the characters appearing in x, y with probability at least $1 - \gamma$.

For a string $s \in \Sigma^n$ and index $1 \le i \le n$, $s[i]$ (or sometimes s_i) denotes the i-th character of s. For $1 \le i \le j \le n$, $s[i..j]$ denotes the substring $s[i] \circ s[i+1] \circ \cdots \circ s[j]$. If $i > j$ then $s[i..j]$ is the empty string.

2.2 Edit Distance

▶ **Definition 2** (Edit distance $\mathsf{ed}(\cdot,\cdot)$)**.** *The* edit distance *between two strings x and y, denoted by $\mathsf{ed}(x,y)$, is the minimum number of edits (insertions, deletions, and substitutions*[5]*) required to transform x to y.*

▶ **Definition 3** (Matching induced by edit sequence $\mathcal{M}(S)$)**.** *Given strings x, y and an edit sequence S, we can construct a bipartite graph between x and y, where every character in x that is not substituted nor deleted is connected by an edge to its counterpart in y. These edges form a non-intersecting matching, which we denote by $\mathcal{M}(S)$. Moreover, when S achieves optimal edit distance, we say $\mathcal{M}(S)$ is an* optimal matching.

Though there may be multiple optimal matchings, the following definition specifies a canonical one.

▶ **Definition 4** (Greedy optimal matching $\mathcal{M}$, [7])**.** *Let x, y be two strings. For each edit sequence S achieving optimal edit distance, let $\mathcal{M}(S)$ be the matching induced by S. Then the* greedy optimal matching *$\mathcal{M}$ is defined to be the smallest $\mathcal{M}(S)$ in lexicographical order. Specifically, we represent $\mathcal{M}(S)$ as a sequence of (i,j) pairs then sort the sequence lexicographically, and the greedy optimal matching is such that this sorted sequence is as lexicographically small as possible.*

2.3 The CGK Random Walk

We review a useful random process called the *CGK random walk*, which was first introduced by Chakraborty, Goldenberg, and Koucký [13], and played a central role in the sketching algorithm of [7].

▶ **Definition 5** (CGK random walk $\lambda_r(s)$, [13])**.** *Given a string $s \in \Sigma^n$, an integer $m \ge 0$, and a sequence of $m \cdot |\Sigma|$ random coins interpreted as a random function $r\colon [m] \times \Sigma \to \{0,1\}$, the m-step CGK random walk is a length-m string $\lambda_r(s) \in \Sigma^m$ defined by the following process:*

- *Append s with infinitely many zeros.*
- *Initialize the pointer $p \leftarrow 1$ and the output string $s' \leftarrow \emptyset$.*
- *For each step $i = 1, \ldots, m$:*
  - *Append $s[p]$ to s'.*
  - *Update $p \leftarrow p + r(i, s[p])$.*
- *Output $s' =: \lambda_r(s)$.*

For a contiguous segment of the output string $\lambda_r(s)$, the pre-image *of this segment refers to the corresponding substring in the original input string s (which may also include the appended trailing zeros if the walk extends beyond s).*

[5] There is another definition of edit distance, denoted by $\mathsf{ed}'(x,y)$, where only insertions and deletions are allowed. We have $\mathsf{ed}(x,y) \le \mathsf{ed}'(x,y) \le 2 \cdot \mathsf{ed}(x,y)$, and $\mathsf{ed}'(x,y) = |x| + |y| - \mathsf{LCS}(x,y)$, where LCS stands for *longest common subsequence*. The algorithm in [7], as well as our modification of it, can be easily adapted to work for this variant of edit distance as well.

Due to its usefulness in the two-party setting with public randomness, we also frequently use the term CGK random walk *to refer to a* pair *of random walks (as defined in Definition 5) performed on two input strings x, y using the* shared *random string r.*

Consider a CGK random walk λ on two input strings x, y. We use p_i (resp., q_i) to denote the pointer on string x (resp., y) at the beginning of step i. We refer to the pair (p_i, q_i) as the state *of λ at the i-th step, and we write $(p, q) \in \lambda$ if λ passes through the state (p, q), i.e., there exists some i for which $p_i = p$ and $q_i = q$. We say the i-th step of λ is a* progress step *if the i-th characters of the output strings $\lambda_r(x)$ and $\lambda_r(y)$ differ, or equivalently, $x[p_i] \neq y[q_i]$.*[6] *We say λ* walks through *x, y, if in the end the two pointers satisfy $p_m \geq |x|$ and $q_m \geq |y|$.*

The following theorem establishes the connection between CGK random walks and edit distance. Informally, when $\mathsf{ed}(x, y)$ is small, with good probability the number of progress steps in λ is also small (or equivalently, the Hamming distance between the output strings $\lambda_r(x), \lambda_r(y)$ is small). We provide a simpler proof for its Item (3) in full version.

▶ **Theorem 6** ([13, Theorem 4.1]). *Let λ be an m-step CGK random walk on x, y. Then*

(1) *if $m \geq 3 \cdot \max\{|x|, |y|\}$, then λ walks through x, y with probability at least $1 - e^{\Omega(m)}$;*

(2) *given $\lambda_r(x)$ and r, we can reconstruct the pre-image of $\lambda_r(x)$;*

(3) $\mathbf{Pr}\left[\#\textit{progress steps in } \lambda \geq (T \cdot \mathsf{ed}(x, y))^2\right] \leq O(1/T)$.

2.4 Random Walks

We frequently relate the CGK random walk to the following one-dimensional random walk.

▶ **Definition 7** (One-dimensional unbiased and self-looped random walk). *A stochastic process $X = (X_t)_{t \in \mathbb{N}}$ on integers is a one-dimensional unbiased and self-looped random walk if its transition satisfies*

$$X_i = \begin{cases} X_{i-1} - 1 & \textit{w.p.}, 1/4, \\ X_{i-1} & \textit{w.p.}, 1/2, \\ X_{i-1} + 1 & \textit{w.p.}, 1/4. \end{cases}$$

▶ Remark 8. Let λ be a CGK random walk on two strings and (p, q) be its state. Define $\Delta = p - q$. Then Δ can be viewed as a one-dimensional unbiased and self-looped random walk, which makes a transition when and only when λ makes a progress step.

By Remark 8 and the martingale property, we have the following lemma, the proof of which can be found in full version.

▶ **Lemma 9.** *Consider an ∞-step CGK random walk λ on x, y, where p, q are the pointers on x, y respectively. Let u be an index and let $U, V \geq u - 1$ be any integers. Then the following holds.*

(1) *Let T_0 be the first time that $p_{T_0} \geq u$. Then we have $\mathbb{E}\left[|p_{T_0} - q_{T_0}|\right] \leq 4 \cdot \mathsf{ed}(x[1..U], y[1..V])$.*

(2) *Let T_1 be the first time that $(p_{T_1} \geq u) \wedge (q_{T_1} \geq u)$. Then we have $\mathbb{E}\left[|p_{T_1} - q_{T_1}|\right] \leq 4 \cdot \mathsf{ed}(x[1..U], y[1..V])$.*

[6] Our definition of "progress step" is different from that of [7], which additionally requires at least one of the two pointers moves forward in that step.

3 Sketches for Edit Distance

For the rest of the paper, we use the following notational conventions:

- n is the length of the input strings; $m := 3n$ is the number of steps in a CGK random walk.
- x, y are the input strings of length n, which is appended with infinitely many zeros; we are promised $\mathsf{ed}(x, y) \le k$.[7]
- when we use $(\cdot, \cdot)$ to denote a CGK state or an edge between x, y, the first coordinate is a pointer on x and the second is on y.
- $\mathcal{M}$ is the greedy optimal matching of x, y.

Our goal is to prove the following theorem.

▶ **Theorem 10.** *There is a sketching algorithm for $\mathscr{Q}_{n,k,\delta}$ of sketch size $O\left(k^3 \log^2(n/\delta) \log n\right)$ bits. Moreover, the algorithm has the following properties.*

- *The encoding algorithm used by Alice (resp., Bob) only assumes one-pass streaming access to the input string x (resp., y). The time complexity per character is $\mathrm{poly}(k \log(n/\delta))$, and the space complexity is $O\left(k^3 \log^2(n/\delta) \log n\right)$ bits.* [8]
- *The decoding algorithm used by the referee has time complexity $\mathrm{poly}(k \log(n/\delta))$.*

In Subsection 3.1, we review the general framework of [7]'s sketching protocol, and highlight our key improvement in Lemma 15. We will prove this key lemma in Subsection 3.2 and Subsection 3.3.

3.1 General Framework

We adopt the definition of *effective alignments* from [7]. Intuitively, an effective alignment between two strings x, y contains the information of an edit sequence from x to y, but does not contain the information of unchanged characters.

▶ **Definition 11** (Effective alignment $\mathcal{A}$, [7]). *For two strings $x, y \in \Sigma^n$, an* effective alignment *$\mathcal{A}$ between x and y is a triplet (G, g_x, g_y), where*

- *$G = (V_x, V_y, E)$ is a bipartite matching where nodes $V_x = [n], V_y = [n]$ correspond to indices of characters in x and y respectively, and every matched edge $(i, j) \in E$ satisfies $x[i] = y[j]$. Moreover, the matched edges are non-intersecting, i.e., for every pair of distinct edges $(i, j), (i', j') \in E$, we have $i < i'$ iff $j < j'$.*
- *g_x (resp., g_y) is a partial function defined on the set of unmatched nodes $U_x \subseteq V_x$ (resp., $U_y \subseteq V_y$). For each $i \in U_x$ (resp., $j \in U_y$), define $g_x(i) = x[i]$ (resp., $g_y(j) = y[j]$).*

▶ **Definition 12** (Effective alignments consistent with a CGK random walk, [7]). *Let λ be a CGK random walk on x, y, where p, q are the pointers on x and y respectively. If λ walks through x, y, then we say an effective alignment $\mathcal{A} = (G, g_x, g_y)$ is* consistent *with λ if for every matched edge $(p, q) \in G$, we have $(p, q) \in \lambda$.*

As mentioned in Subsection 1.1, Alice and Bob use public randomness to instantiate $\tau = O(k \log(n/\delta))$ independent CGK random walks $\lambda_1, \ldots, \lambda_\tau$ on x, y. Then, for each CGK random walk λ_i, Alice constructs a sketch sx_i based on her part of the random walk $\lambda_i(x)$,

[7] We also analyze the behaviour of our algorithms when $\mathsf{ed}(x, y) > k$ in full version.

[8] The algorithm may use a large number of shared random bits, which can be reduced using Nisan's generator [24]. The main cost, as we can see from the proof, comes from the CGK random walk. We refer readers to [13] for more details on reducing randomness for the CGK random walk.

and Bob similarly constructs sy_i based on his part of the random walk $\lambda_i(y)$. The referee receives sx_i, sy_i, and tries to extract an effective alignment $\mathcal{A}_i$ from the sketches. Each sx_i (and sy_i) has length $O(k^2 \log(n/\delta) \log n)$. The properties of this protocol are summarized as follows.

▶ **Construction 13** (Sketch for each random walk, adapting [7]). Let $C \geq 1$ be some large constant and $\eta \in (0, 1)$. There exists an efficient sketching algorithm such that the following holds. Let λ be an m-step CGK random walk on x (and y). Then,

- the sketch size and encoding space are $O\left(k^2 \log(n/\eta) \log n\right)$ bits;
- the encoding time per character and decoding time are both $\mathrm{poly}(k \log(n/\eta))$;
- for fixed λ, x, y the following hold with success probability at least $1 - \eta$:
  - the decoder either (a) reports "error", or (b) outputs an effective alignment $\mathcal{A}$ consistent with λ;
  - when λ walks through x, y and contains at most $C \cdot k^2$ progress steps, (b) occurs.

The formal proof of Construction 13 can be found in full version.

The final sketches are simply $sx = sx_1 \circ \cdots \circ sx_\tau$ and $sy = sy_1 \circ \cdots \circ sy_\tau$. The referee tries to obtain an effective alignment from every (sx_i, sy_i), and then uses the following lemma to compute $\mathsf{ed}(x, y)$ and recover an optimal edit sequence.

▶ **Lemma 14** ([7, Lemma 14 and Lemma 19]). *There exists a deterministic algorithm taking (sx, sy) as input such that the following holds.*

- *The running time of the algorithm is $\mathrm{poly}(|sx| + |sy|) = \mathrm{poly}(k \log(n/\delta))$.*
- *Let $\mathcal{A}_{i_1}, \ldots, \mathcal{A}_{i_w}$ be the effective alignments[9] decoded from $(sx_1, sy_1), \ldots, (sx_\tau, sy_\tau)$. If $w \geq 1$ and each $\mathcal{A}_{i_j}$ is consistent with λ_{i_j}, then the algorithm outputs a valid edit sequence. If, additionally, $\mathcal{M}$ goes through all edges that are common to $\mathcal{A}_{i_1}, \ldots, \mathcal{A}_{i_w}$, then the edit sequence is optimal.*

Now we state our key lemma.

▶ **Lemma 15** (Key Lemma). *There exist some large constants $C_1, C_2 \geq 1$ such that the following holds. Let λ be an ∞-step CGK random walk on x, y. Then for any fixed $(u, v) \notin \mathcal{M}, x[u] = y[v]$, we have*

$$\mathbf{Pr}\left[(u, v) \notin \lambda \bigwedge \#\textit{progress steps in } \lambda \leq C_1 \cdot k^2\right] \geq \frac{1}{C_2 \cdot k}.$$

Here we reiterate that Lemma 15 summarizes our improvement over the previous work of [7] in two aspects (as mentioned in Subsection 1.1): (1) The previous work only gave a lower bound on $\mathbf{Pr}[(u, v) \notin \lambda]$, while we bound the probability of two events happening simultaneously; (2) The previous work only gave a bound of $\Omega(1/k^2)$, while we give an $\Omega(1/k)$ bound. The proof of this Lemma 15 is divided into two parts in Subsection 3.2 and Subsection 3.3, in which a technical proposition that leads to the improvement in Item (2) will be proved in Section 4.

Assuming Lemma 15, we can prove Theorem 10.

[9] Although we can check if $\mathcal{A}_{i_j}$ is an effective alignment, we cannot verify (without knowing λ_{i_j}) if $\mathcal{A}_{i_j}$ is an effective alignment *consistent with* λ_{i_j}. This subtle difference comes from that in Construction 13 we do not give any guarantee outside the $1 - \eta$ success probability, where the decoder might provide some effective alignment that is *not* consistent with λ_{i_j}.

Proof of Theorem 10. Let C_3 be a large constant.

For the encoding part, we instantiate $\tau = C_2 k \cdot C_3 \log(n/\delta) = O(k \cdot \log(n/\delta))$ independent m-step CGK random walks $\lambda_i, i \in [\tau]$; and construct each sx_i, sy_i using Construction 13 with parameter $C = C_1, \eta = \delta/(2\tau)$.

For the decoding part, we run the decoding procedure in Construction 13 to obtain $\mathcal{A}_{i_1}, \ldots, \mathcal{A}_{i_w}$ for Lemma 14. If $w = 0$ or the edit sequence from Lemma 14 has more than k edits, we report "error"; otherwise we output the edit sequence together with the corresponding edit distance.

Bounds on the parameters. By constructing each sx_i (and sy_i) in parallel, the final sketch size and encoding space are $\tau \cdot O\left(k^2 \log(n/\eta) \log n\right) = O\left(k^3 \log^2(n/\delta) \log n\right)$ (we omit the space for storing auxiliary information (e.g., pointers) in the calculation, since these are minor terms). The encoding time per character is $\tau \cdot \mathrm{poly}\left(k \log(n/\eta)\right) = \mathrm{poly}(k \log(n/\delta))$. The decoding time follows immediately from Lemma 14.

Analysis of the algorithm when $\mathrm{ed}(x, y) \leq k$. Let $S = \left\{(u, v) \in [n]^2 \mid (u, v) \notin \mathcal{M}, x[u] = y[v]\right\}$ and define events

- $\mathcal{E}_i$: λ_i walks through x, y.
- $\mathcal{E}'_i(u, v)$ for $(u, v) \in S$: $(u, v) \notin \lambda_i \bigwedge \#$progress steps in $\lambda_i \leq C_1 \cdot k^2$.

Then

$$\begin{aligned}
&\mathbf{Pr}\left[\forall (u,v) \in S,\ \exists i \in [\tau],\ \mathcal{E}_i \wedge \mathcal{E}'_i(u,v)\right] \\
&\geq 1 - \sum_{(u,v)\in S} \left(1 - \mathbf{Pr}\left[\neg \mathcal{E}_1\right] - \mathbf{Pr}\left[\neg \mathcal{E}'_1(u,v)\right]\right)^{\tau} \\
&\geq 1 - n^2 \cdot \left(1 - e^{\Omega(n)} - \frac{1}{C_2 \cdot k}\right)^{\tau} && \text{(due to Theorem 6 and Lemma 15)} \\
&\geq 1 - \frac{\delta}{2}. && (1)
\end{aligned}$$

Let $\lambda_{i_1}, \ldots, \lambda_{i_w}$ be the random walks walking through x, y and containing at most $C_1 \cdot k^2$ progress steps. Since $\eta = \delta/(2\tau)$ in Construction 13 and by union bound, the decoder, with probability at least $1 - \delta/2$, for each (sx_i, sy_i) either reports "error", or outputs an effective alignment $\mathcal{A}_i$ consistent with λ_i. Conditioning on this, Construction 13 must at least obtain effective alignments $\mathcal{A}_{i_j}, \ldots, \mathcal{A}_{i_w}$ that are consistent with the corresponding random walks. Combined with (1), with probability at least $1 - \delta$, for any $(u, v) \in S$ there exists some λ_{i_j} missing it. Then the edit sequence from Lemma 14 is optimal. ◀

3.2 Proof of Lemma 15: Case $|u - v| > 100 \cdot k$

Proof of Lemma 15: Case $|u - v| > 100 \cdot k$. Assume without loss of generality $u > v$. We stop λ when it meets u. Then by Item (1) in Lemma 9, at this time the state (p, q) satisfies $\mathbb{E}[|p - q|] \leq 4 \cdot k$. Hence by Markov's inequality,

$$\mathbf{Pr}\left[(u,v) \notin \lambda\right] \geq \mathbf{Pr}\left[p - q \leq 100 \cdot k\right] = 1 - \mathbf{Pr}\left[p - q > 100 \cdot k\right] \geq 1 - \frac{4 \cdot k}{100 \cdot k} = 0.96. \quad (2)$$

On the other hand, by setting C_1 large enough we know from Theorem 6

$$\mathbf{Pr}\left[\#\text{progress steps in } \lambda \leq C_1 \cdot k^2\right] \geq 0.99.$$

Hence, by setting C_2 large enough, we have

$$\mathbf{Pr}\left[(u,v) \notin \lambda \bigwedge \#\text{progress steps in } \lambda \leq C_1 \cdot k^2\right] \geq 0.96 + 0.99 - 1 \geq \frac{1}{C_2 \cdot k}.$$ ◀

3.3 Proof of Lemma 15: Case $|u - v| \leq 100 \cdot k$

First we need the following definition.

▶ **Definition 16** (Stable zone $\mathcal{Z}$, [7]). *The* stable zone $\mathcal{Z}$ *of* (u, v) *consists of substrings* $x[u'..u], y[v'..v]$ *of equal length* $L = u - u' + 1 = v - v' + 1$*, where* $L \leq \min\{u, v\}$ *is the maximum possible length satisfying* $x[u'..u] = y[v'..v]$*. In particular,* $u - v = u' - v'$*; and* $(u', v') \neq (1, 1)$ *as* $(u, v) \notin \mathcal{M}$*.*

Moreover, we say a state (p, q) enters $\mathcal{Z}$ *if* $p \geq u'$ *and* $q \geq v'$*.*

We will find Claim 17 useful. For completeness we include its proof in full version.

▷ Claim 17 ([7, Claim 21]). Consider an ∞-step CGK random walk λ on x, y, where p, q are the pointers on x, y respectively. Let T be the first time that λ enters $\mathcal{Z}$, i.e., $(p_T \geq u') \wedge (q_T \geq v')$. Then $\mathbf{Pr}[p_T - q_T \neq u - v] = \mathbf{Pr}[p_T - q_T \neq u' - v'] \geq 2/3$.

We will also rely on the following technical result, the proof of which is in Section 4.

▶ **Proposition 18.** *There exists a universal constant* $C_4 \geq 1$ *such that the following holds. Assume* X, Y *are two identical length-*L *strings over alphabet* Σ*. Assume there exists a size-*M *matching* $(i_1, j_1), \ldots, (i_M, j_M) \in [L]^2$ *such that*

- $i_t > j_t$ *and* $X[i_t] = Y[j_t]$ *hold for all* $t \in [M]$*;*
- $i_1 < i_2 < \cdots < i_M$ *and* $j_1 < j_2 < \cdots < j_M$*.*

Let $\rho = C_4 \cdot (L - M)$ *and* $(\hat{I}, \hat{J})$ *be any state satisfying* $\hat{I} - \hat{J} \geq \rho$*. Then a CGK random walk on* X, Y *starting from* $(\hat{I}, \hat{J})$ *will miss* (L, L) *with probability at least* 0.5*.*

By symmetry, we derive the following corollary.

▶ **Corollary 19.** *Let* $C_4 \geq 1$ *be the same constant in Proposition 18. Assume* X, Y *are two identical length-*L *strings over alphabet* Σ*. Assume there exists a size-*d *matching* $(i_1, j_1), \ldots, (i_M, j_M) \in [L]^2$ *such that*

- $i_t > j_t$ *holds for all* $t \in [M]$*, or* $i_t < j_t$ *holds for all* $t \in [M]$*;*
- $X[i_t] = Y[j_t]$ *holds for all* $t \in [M]$*;*
- $i_1 < i_2 < \cdots < i_M$ *and* $j_1 < j_2 < \cdots < j_M$*.*

Let $\rho = C_4 \cdot (L - M)$ *and* $(\hat{I}, \hat{J})$ *be any state satisfying* $|\hat{I} - \hat{J}| \geq \rho$*. Then a CGK random walk on* X, Y *starting from* $(\hat{I}, \hat{J})$ *will miss* (L, L) *with probability at least* 0.5*.*

Proof of Lemma 15: Case$|u - v| \leq 100 \cdot k$. Let $C_5 \geq 1$ be a large constant. We will apply Proposition 18 with parameter $M \geq L - 103 \cdot k$; and let $\rho = C_4 \cdot 103k$ be the corresponding bound in it.

We expect λ to have the following three phases:

- $\mathcal{E}_1$: λ enters $\mathcal{Z}$ in a state (p_1, q_1) within $C_5 \cdot k^2$ progress steps, where $0 < |(p_1 - q_1) - (u - v)| \leq 200 \cdot k$.
- $\mathcal{E}_2$: Starting from (p_1, q_1) and within $2 \cdot \rho^2$ progress steps, λ reaches a state (p_2, q_2) where either $(p_2, q_2) > (n, n)$ or $|(p_2 - q_2) - (u - v)| \geq \rho$. Also, during the walk from (p_1, q_1) to (p_2, q_2), λ never reaches some state (p, q) satisfying $(p - q) - (u - v) = 0$.
- $\mathcal{E}_3$: $(u, v) \notin \lambda$ and #progress steps in $\lambda \leq 2 \cdot \rho^2 + C_5 \cdot \left(k^2 + (\rho + 301 \cdot k)^2\right)$.

In fact we have the following claim, the proof of which can be found in full version.

▷ Claim 20. $\mathbf{Pr}[\mathcal{E}_1] \geq 0.5$, $\mathbf{Pr}[\mathcal{E}_2 \mid \mathcal{E}_1] \geq 1/(2 \cdot \rho)$, and $\mathbf{Pr}[\mathcal{E}_3 \mid \mathcal{E}_1 \wedge \mathcal{E}_2] \geq 1/4$.

Assuming Claim 20, we have the following desired bound

$$\mathbf{Pr}\left[(u, v) \notin \lambda \bigwedge \#\text{progress steps in } \lambda \leq C_1 \cdot k^2\right] \geq \mathbf{Pr}[\mathcal{E}_3] \geq \mathbf{Pr}[\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3] \geq \frac{1}{C_2 \cdot k}$$

by setting $C_1 = 2 \cdot (103 \cdot C_4)^2 + C_5 \cdot \left(1 + (301 + 103 \cdot C_4)^2\right)$ and $C_2 = 16$. ◀

4 CGK Random Walks on Self-similar Strings

This section is devoted for Proposition 18. It characterizes CGK random walks on strings of certain self-similarity, which may be interesting on its own.

▶ **Proposition** (Proposition 18 restated). *There exists a universal constant $C_4 \geq 1$ such that the following holds. Assume X, Y are two identical length-L strings over alphabet Σ. Assume there exists a size-M matching $(i_1, j_1), \ldots, (i_M, j_M) \in [L]^2$ such that*

- *$i_t > j_t$ and $X[i_t] = Y[j_t]$ hold for all $t \in [M]$;*
- *$i_1 < i_2 < \cdots < i_M$ and $j_1 < j_2 < \cdots < j_M$.*

Let $\rho = C_4 \cdot (L - M)$ and $(\hat{I}, \hat{J})$ be any state satisfying $\hat{I} - \hat{J} \geq \rho$. Then a CGK random walk on X, Y starting from $(\hat{I}, \hat{J})$ will miss (L, L) with probability at least 0.5.

We will first provide necessary definitions and state basic properties in Subsection 4.1. Then present the proof in Subsection 4.2. All missing proofs can be found in full version.

4.1 Stable States

We fix the matching in Proposition 18, so when we say (i, j) is a matched edge it means (i, j) is an edge in the matching. We extend X, Y to $X[-\infty..\infty], Y[-\infty..\infty]$ by adding dummy characters $X[i] = Y[i] = X[L]$ for all $i > L$, and $X[i] = Y[i] = X[1]$ for all $i < 1$. We also add matched edges $(i, i-1)$ for all $i > L$ as well as $i \leq 1$. Note that all the edges are still non-intersecting. Though the added characters may not be consistent with the original input strings x, y, it does not change the probability of the walk missing (L, L). Since $X = Y$ and the initial state satisfies $\hat{I} \geq \hat{J} + \rho \geq \hat{J}$, any future state (I, J) must still satisfy $I \geq J$.

We introduce the notion of *stable segment*.

▶ **Definition 21** (Stable segment). *We say $[l..r]$ is a* stable segment*, if for every matched edge (I, J) (where we must have $I > J$), exactly one of the following two conditions hold:*

- *$J < l$ and $I \leq r$.*
- *$J \geq l$ and $I > r$.*

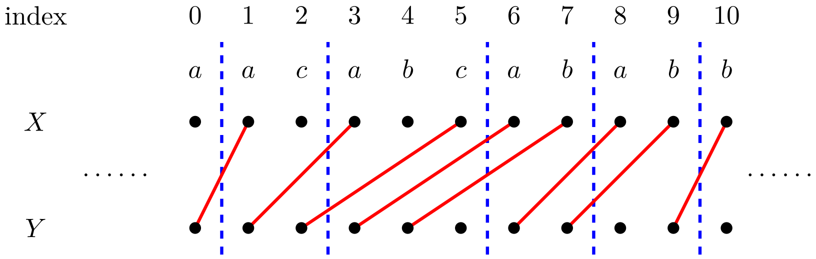


Figure 1 A stable partition for $X[1..L] = Y[1..L] = acabcabab$ $(L = 9)$.

For example in Figure 1, every segment separated by blue dashed lines is a stable segment.

▶ Remark 22. To gain a better intuition of the definition, consider the special case where the string $X[1..L]$ has period p and every matched edge (I, J) inside segment $[1..L]$ satisfies $I - J = p$. In this periodic case, a segment contained in $[2..L-1]$ is stable if and only if its length is p.

Our motivation is that, when there are few unmatched characters, using our more generalized definition we can approximately preserve the nice properties of periodic strings. For example, when X has period p, the strings $X[i..i + tp - 1]$ and $X[i + tp..i + 2tp - 1]$

must be identical. In a non-periodic case, we can similarly prove that $X[i..j-1]$ and $X[j..j+(j-i)-1]$ have small edit distance if $[i..j-1]$ can be divided into several stable segments. In the remaining part of the section, readers are encouraged to use the periodic case for a more intuitive understanding.

▶ **Definition 23** (Stable partition $\mathcal{P}$ and stable states). *Consider a partition $\mathcal{P} = (\mathcal{P}_i)_i$ of the integers into segments, where $\mathcal{P}_i = [p_i..p_{i+1}-1]$ and $p_i < p_{i+1}$. We say $\mathcal{P}$ is a* stable partition *if every $\mathcal{P}_i$ is a stable segment. Then we say*

- *state (I, J) is a* $(\mathcal{P}, b)$-stable state, *if there exists some i such that $J = p_i$ and $I = p_{i+b}$;*
- *state (I, J) is a* b-stable state, *if there exists a stable partition $\mathcal{P}$ such that (I, J) is a $(\mathcal{P}, b)$-stable state;*
- *state (I, J) is a* stable state, *if there exists some $b \geq 0$ such that (I, J) is a b-stable state. In particular, when $I \geq J > L$, (I, J) is always a stable state.*

Given a stable partition $\mathcal{P}$, we can define a predecessor function for $\mathcal{P}$ as follows.

▶ **Lemma 24** (Stable predecessor for a stable partition). *Let $\mathcal{P} = (\mathcal{P}_i)_i$ be a stable partition where $\mathcal{P}_i = [p_i..p_{i+1}-1]$ and $p_i < p_{i+1}$. Then there exists a non-decreasing function $\mathsf{pred}_{\mathcal{P}} : \mathbb{Z} \to \mathbb{Z}$ such that the following holds:*

- *For every i, $\mathsf{pred}_{\mathcal{P}}(p_{i+1}) = p_i$.*
- *For every I, we have $\mathsf{pred}_{\mathcal{P}}(I) \leq I - 1$, and $[\mathsf{pred}_{\mathcal{P}}(I)..I-1]$ is a stable segment.*

Now we extend the definition to b-stable predecessor.

▶ **Definition 25** (b-stable predecessors $\mathsf{pred}^{(b)}_{\mathcal{P}}(\cdot)$). *Let $\mathcal{P}$ be a stable partition. Define*

$$\mathsf{pred}^{(b)}_{\mathcal{P}}(I) = \begin{cases} I & b = 0, \\ \mathsf{pred}_{\mathcal{P}}(\mathsf{pred}^{(b-1)}_{\mathcal{P}}(I)) & b \geq 1. \end{cases}$$

As an example, in Figure 1 $\mathsf{pred}^{(3)}_{\mathcal{P}}(8) = 1$.

We will bound the edit distance between stable states using the number of singletons.

▶ **Definition 26** (Singleton). *Every unmatched $X[i]$ or $Y[j]$ is called a* singleton.

Let $\mathsf{sing}_X[l, r)$ (resp., $\mathsf{sing}_Y[l, r)$) denote the number of singletons in $X[l..r-1]$ (resp., $Y[l..r-1]$). Let $\mathsf{sing}[l, r) := \mathsf{sing}_X[l, r) + \mathsf{sing}_Y[l, r)$.

▶ **Lemma 27.** *Let $\mathcal{P}$ be a stable partition. For $I < I'$, let $J = \mathsf{pred}_{\mathcal{P}}(I), J' = \mathsf{pred}_{\mathcal{P}}(I')$. Then*

(a) $\mathsf{ed}(X[I..I'-1], Y[J..J'-1]) \leq \mathsf{sing}_X[I, I') + \mathsf{sing}_Y[J, J') \leq \mathsf{sing}[J, I')$*;*

(b) $|(I' - J') - (I - J)| = |(I' - I) - (J' - J)| \leq \mathsf{sing}_X[I, I') + \mathsf{sing}_Y[J, J') \leq \mathsf{sing}[J, I')$*.*

4.2 Proof of Proposition 18

Before proving Proposition 18, we need the following lemma, which shows a CGK random walk goes from a stable state to a distant stable state with low cost.

▶ **Lemma 28** (From stable to stable). *Consider a CGK random walk starting from a stable state $(I_0, J_0), I_0 > J_0$. Let D be a distance bound satisfying $D \geq I_0 - J_0$.*

Consider the first time $T > 0$ that either $I_T - J_T > D$, or the following three conditions hold simultaneously: $J_T \geq I_0$, and $I_T \geq 2I_0 - J_0$, and (I_T, J_T) is a stable state. Let P be the number of progress steps before time T and let $S = \mathsf{sing}[J_0, I_T)$. Then

$$\mathbb{E}\left[P - 2000 \cdot \left(S \cdot D + S^2\right)\right] \leq 0.$$

The process of Lemma 28 consists of a "catch-up phase" (i.e., from a stable state to a distant non-stable state) and then a "stabilization phase" (i.e., from a non-stable state to a nearby stable state). We describe the latter one as Lemma 29.

▶ **Lemma 29** (From non-stable to stable). *Consider a CGK random walk starting from a non-stable state $(\tilde{I}_0, \tilde{J}_0), \tilde{I}_0 > \tilde{J}_0$. Let $\mathcal{P}$ be a stable partition and let b' be such that $\tilde{L}_0 < \tilde{J}_0 < \tilde{R}_0$ where $\tilde{L}_0 = \mathsf{pred}_{\mathcal{P}}^{(b')}(\tilde{I}_0), \tilde{R}_0 = \mathsf{pred}_{\mathcal{P}}^{(b'-1)}(\tilde{I}_0)$. Let D be a distance bound satisfying $D \geq \tilde{I}_0 - \tilde{J}_0$. Consider the first time T' that either $(\tilde{I}_{T'}, \tilde{J}_{T'})$ is a stable state or $\tilde{I}_{T'} - \tilde{J}_{T'} > D$. Let P' be the number of progress steps before time T' and let $S' = \mathsf{sing}[\tilde{L}_0, \tilde{I}_{T'})$. Then*

$$\mathbb{E}\left[P' - 50 \cdot \left((\tilde{R}_0 - \tilde{J}_0)(\tilde{J}_0 - \tilde{L}_0) + S' \cdot D + {S'}^2\right)\right] \leq 0.$$

Given previous lemmas to control progress steps, we are now ready to prove Proposition 18.

Proof of Proposition 18. Since $X[1]$ and $Y[L]$ are matched to dummy characters after we extend X, Y, there are $K := \mathsf{sing}[-\infty, +\infty) = 2 \cdot (L - M - 1)$ singletons in total. Let $d := \hat{I} - \hat{J}$ be the initial distance between the two pointers and let $D := 2 \cdot d$. For a state $(I, J), I \geq J$,

- if $I = J \leq L$ or $I - J > D$, then we say it is a *failure state*;
- if it is not a failure state and $I > L$, then we say it is a *success state.*

We stop the CGK random walk when it reaches a success state or a failure state. The former case implies that the random walk misses (L, L). So it suffices to prove that we stop at a success state with probability at least 0.5.

Phases in the CGK random walk. Let $I_0 = \hat{I}, J_0 = \hat{J}$ and $t_0 = 0$. Let $t_1 \geq 0$ be the first time that either (I_{t_1}, J_{t_1}) is a stable state or $I_{t_1} - J_{t_1} > D$.

For every $i \geq 2$, if $(I_{t_{i-1}}, J_{t_{i-1}})$ is neither a success state nor a failure state, we know $J_{t_{i-1}} < I_{t_{i-1}} \leq L$ and $I_{t_{i-1}} - J_{t_{i-1}} \leq D$. Then we recursively define $t_i > t_{i-1}$ to be the first time that either $I_{t_i} - J_{t_i} > D$, or the following three conditions hold simultaneously: $I_{t_i} \geq 2I_{t_{i-1}} - J_{t_{i-1}}$, and $J_{t_i} \geq I_{t_{i-1}}$, and (I_{t_i}, J_{t_i}) is a stable state.

Assume we stop at (I_{t_m}, J_{t_m}), which is either a success state or a failure state. Let P_i be the number of progress steps made during the time interval $[t_i, t_{i+1})$. Then $P := \sum_{i=0}^{m-1} P_i$ is the total number of progress steps before we stop.

Bounds on $\mathbb{E}[P_0]$. Let $\mathcal{P}$ be an arbitrary stable partition and let b be such that $\mathsf{pred}_{\mathcal{P}}^{(b)}(I_0) \leq J_0 < \mathsf{pred}_{\mathcal{P}}^{(b-1)}(I_0)$. Let $L_0 = \mathsf{pred}_{\mathcal{P}}^{(b)}(I_0), R_0 = \mathsf{pred}_{\mathcal{P}}^{(b-1)}(I_0)$. Since $X[1]$ is matched to $Y[0]$, we know $\mathsf{pred}_{\mathcal{P}}(1) = 0$. Hence applying Lemma 27 with $I' = R_0, I = 1$, we have

$$(R_0 - L_0) - (1 - 0) \leq \mathsf{sing}[0, R_0) \leq \mathsf{sing}[0, I_0).$$

Therefore, let $S_0 = \mathsf{sing}[0, I_{t_1})$ and we have

$$(J_0 - L_0)(R_0 - J_0) \leq \left\lfloor \frac{R_0 - L_0}{2} \right\rfloor \cdot \left\lceil \frac{R_0 - L_0}{2} \right\rceil \leq (\mathsf{sing}[0, I_0))^2 \leq S_0^2.$$

Thus by Lemma 29, we have $\mathbb{E}\left[P_0 - 50 \cdot \left(2 \cdot S_0^2 + S_0 \cdot D\right)\right] \leq 0$.

Bounds on $\mathbb{E}[P_i], 1 \leq i \leq m - 1$. Let $S_i := \mathsf{sing}[J_{t_i}, I_{t_{i+1}})$. By Lemma 28, we have $\mathbb{E}\left[P_i - 2000 \cdot \left(S_i \cdot D + S_i^2\right)\right] \leq 0$.

Final bounds. Note that $\sum_{0\le i<m} S_i \le 2\cdot \mathsf{sing}[0, I_{t_m}) \le 2\cdot K$. This is because $J_{t_{i+1}} \ge I_{t_i}$ for all $i \ge 1$, implying each singleton is counted at most twice. Hence

$$\mathbb{E}[P] = \mathbb{E}\left[\sum_{i=0}^{m-1} P_i\right] \le \mathbb{E}\left[2000\sum_{i=0}^{m-1}(S_iD + S_i^2)\right] \le 2000\cdot\mathbb{E}\left[D\sum_{i=0}^{m-1} S_i + \left(\sum_{i=0}^{m-1} S_i\right)^2\right]$$
$$\le 8000\cdot(K\cdot d + K^2).$$

For $1 \le j < +\infty$, let r_j be the deviation brought by the j-th progress step.[10] Then r_j are i.i.d. random variables with $\mathbf{Pr}[r_j = 0] = 1/2, \mathbf{Pr}[r_j = +1] = \mathbf{Pr}[r_j = -1] = 1/4$. Hence by Cauchy-Schwarz inequality, we have

$$\mathbb{E}\left[\left|\sum_{j=1}^{P} r_j\right|\right] = \mathbb{E}\left[\left|\sum_{j=1}^{+\infty} r_j\cdot 1_{\{j\le P\}}\right|\right] \le \sqrt{\mathbb{E}\left[\left(\sum_{j=1}^{+\infty} r_j\cdot 1_{\{j\le P\}}\right)^2\right]} = \sqrt{\mathbb{E}\left[\frac{P}{2}\right]}$$
$$\le \sqrt{4000\cdot(K\cdot d + K^2)}.$$

Observe that in the end we have $I_{t_m} - J_{t_m} = d + \sum_{j=1}^{P} r_j$. By setting $\rho = C_4\cdot(L-M)$ for some large enough constant C_4, we have $d \ge \rho \ge C_4\cdot K/2$ and

$$d \ge 4\cdot\sqrt{4000\cdot(K\cdot d + K^2)} \ge 4\cdot\mathbb{E}\left[\left|\sum_{j=1}^{P} r_j\right|\right].$$

Then by Markov's inequality, with probability at least 0.5 we have $|I_{t_m} - J_{t_m} - d| \le d/2$, which indicates (I_{t_m}, J_{t_m}) is not a failure state. Hence we stop at some success state with probability at least 0.5. ◀

5 Discussion

Building upon [7], we present an improved sketching algorithm for edit distance with sketch size $\tilde{O}(k^3)$. Although the algorithm itself is essentially the same as in [7], the analysis is more involved. We conclude the paper with a few remarks on further problems.

- **Lower bounds.** We conjecture the lower bound for this problem (i.e., $\mathscr{Q}_{n,k,\delta}$) is $\tilde{\Omega}(k^2)$, since $\Theta(k^2)$ is the distortion of the CGK random walk embedding [13]. However, to the best of our knowledge, there is no lower bound beyond $\tilde{\Omega}(k)$. (Since we do not find any paper formally stating the lower bounds, we present them in Appendix B of full version.)
- **Edit distance.** It is natural to wonder if current framework can be pushed further. For example, is it possible that we only run $\tau = O(1)$ rounds of CGK random walks and there will be an optimal matching going through all edges that are common to these walks? Unfortunately this is not true, and we can show $\tau = \Omega(\sqrt{k})$ with the following example:

$$x = Ac_1c_2\cdots c_{k-1}Bc_1c_2\cdots c_{k-1}\underbrace{d\cdots d}_{2k}Ac_1c_2\cdots c_{k-1},$$
$$y = Bc_1c_2\cdots c_{k-1}\underbrace{d\cdots d}_{2k}Ac_1c_2\cdots c_{k-1}Bc_1c_2\cdots c_{k-1}.$$

[10] Though we will only use $r_1, \ldots, r_P$, we define it in this way to make the next Cauchy-Schwarz inequality easier to understand.

Then with probability $1 - \Theta(1/\sqrt{k})$, a CGK random walk walks through (k, k). Note that $\mathsf{ed}(x, y) \le 2 \cdot k$ by deleting $x[1..k]$ and inserting $y[4k + 1..5k]$. However any edit sequence leaving (k, k) matched will have at least $(2 \cdot k + 1)$ edits, where the one more edit comes from substituting $x[1]$ with $y[1]$. Moreover, this example may generalize to the binary alphabet by replacing each symbol with a short random binary string.

- **Ulam distance.** The Ulam distance is the edit distance on two permutations, i.e., $x \in [n]^n$ (resp., $y \in [n]^n$) and $x_i \ne x_j$ (resp., $y_i \ne y_j$) for distinct i, j. Our algorithm (as well as the algorithm in [7]) works for Ulam distance with an improved bound $\tilde{O}(k^{2.5})$. This comes from the following observation: there is no matched edge in the stable zone, hence the length of stable zone is at most k, which means we can set $\rho = O(\sqrt{L})$ in Proposition 18. It would be interesting to improve the algorithm for Ulam distance.
- **Only the distance.** Though our algorithm computes edit distance as well as an optimal edit sequence, it is reasonable to relax the problem by simply asking for the distance or even a constant approximation of the distance. However, we are not aware of any result achieving better sketch size in this setting.

References

1 Alexandr Andoni and Robert Krauthgamer. The smoothed complexity of edit distance. *ACM Trans. Algorithms*, 8(4):44:1–44:25, 2012. doi:10.1145/2344422.2344434.

2 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 377–386, 2010. doi:10.1109/FOCS.2010.43.

3 Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it's a constant factor. *CoRR*, abs/2005.07678, 2020. To appear in FOCS 2020. arXiv:2005.07678.

4 Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. *SIAM J. Comput.*, 41(6):1635–1648, 2012. doi:10.1137/090767182.

5 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.*, 47(3):1087–1097, 2018. doi:10.1137/15M1053128.

6 Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 550–559, 2004. doi:10.1109/FOCS.2004.14.

7 Djamal Belazzougui and Qin Zhang. Edit distance: Sketching, streaming, and document exchange. In *Proceedings of the 57th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 51–60. IEEE Computer Society, 2016. Full version at arXiv:1607.04200. doi:10.1109/FOCS.2016.15.

8 Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, Mohammad Taghi Hajiaghayi, and Saeed Seddighin. Approximating edit distance in truly subquadratic time: Quantum and MapReduce. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1170–1189, 2018. doi:10.1137/1.9781611975031.76.

9 Mahdi Boroujeni, Masoud Seddighin, and Saeed Seddighin. Improved algorithms for edit distance and LCS: beyond worst case. In *Proceedings of the 31st ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 1601–1620. SIAM, 2020. doi:10.1137/1.9781611975994.99.

10 Joshua Brakensiek, Moses Charikar, and Aviad Rubinstein. A simple sublinear algorithm for gap edit distance. *CoRR*, abs/2007.14368, 2020. arXiv:2007.14368.

11 Joshua Brakensiek and Aviad Rubinstein. Constant-factor approximation of near-linear edit distance in near-linear time. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 685–698, 2020. doi:10.1145/3357713.3384282.

12 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In *Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 979–990. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00096.

13 Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for embedding and computing edit distance in the low distance regime. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 712–725. ACM, 2016. doi:10.1145/2897518.2897577.

14 Moses Charikar and Robert Krauthgamer. Embedding the ulam metric into ℓ_1. *Theory Comput.*, 2(11):207–224, 2006. doi:10.4086/toc.2006.v002a011.

15 Kuan Cheng, Zhengzhong Jin, Xin Li, and Ke Wu. Deterministic document exchange protocols, and almost optimal binary codes for edit errors. In *Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 200–211, 2018. doi:10.1109/FOCS.2018.00028.

16 Kuan Cheng and Xin Li. Efficient document exchange and error correcting codes with asymmetric information. *CoRR*, abs/2007.00870, 2020. To appear in SODA 2021. arXiv:2007.00870.

17 Elazar Goldenberg, Robert Krauthgamer, and Barna Saha. Sublinear algorithms for gap edit distance. In *Proceedings of the 60th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1101–1120, 2019. doi:10.1109/FOCS.2019.00070.

18 Bernhard Haeupler. Optimal document exchange and new codes for insertions and deletions. In *Proceedings of the 60th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 334–347, 2019. doi:10.1109/FOCS.2019.00029.

19 Subhash Khot and Assaf Naor. Nonembeddability theorems via Fourier analysis. *Mathematische Annalen*, 334:821–852, 2006.

20 Tomasz Kociumaka and Barna Saha. Sublinear-time algorithms for computing & embedding gap edit distance. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, 2020.

21 Michal Koucký and Michael E. Saks. Constant factor approximations to edit distance on far input pairs in nearly linear time. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 699–712. ACM, 2020. doi:10.1145/3357713.3384307.

22 Robert Krauthgamer and Yuval Rabani. Improved lower bounds for embeddings into l_1. *SIAM J. Comput.*, 38(6):2487–2498, 2009. doi:10.1137/060660126.

23 William J. Masek and Mike Paterson. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.*, 20(1):18–31, 1980. doi:10.1016/0022-0000(80)90002-1.

24 Noam Nisan. Pseudorandom generators for space-bounded computation. *Comb.*, 12(4):449–461, 1992. doi:10.1007/BF01305237.

25 Alon Orlitsky. Interactive communication: Balanced distributions, correlated files, and average-case complexity. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 228–238, 1991. doi:10.1109/SFCS.1991.185373.

26 Rafail Ostrovsky and Yuval Rabani. Low distortion embeddings for edit distance. *J. ACM*, 54(5):23, 2007. doi:10.1145/1284320.1284322.

Locality Sensitive Hashing for Efficient Similar Polygon Retrieval

Haim Kaplan ✉
School of Computer Science, Tel Aviv University, Israel

Jay Tenenbaum ✉
School of Computer Science, Tel Aviv University, Israel

Abstract

Locality Sensitive Hashing (LSH) is an effective method of indexing a set of items to support efficient nearest neighbors queries in high-dimensional spaces. The basic idea of LSH is that similar items should produce hash collisions with higher probability than dissimilar items.

We study LSH for (not necessarily convex) polygons, and use it to give efficient data structures for similar shape retrieval. Arkin et al. [2] represent polygons by their “turning function” - a function which follows the angle between the polygon’s tangent and the x-axis while traversing the perimeter of the polygon. They define the distance between polygons to be variations of the L_p (for $p = 1, 2$) distance between their turning functions. This metric is invariant under translation, rotation and scaling (and the selection of the initial point on the perimeter) and therefore models well the intuitive notion of shape resemblance.

We develop and analyze LSH near neighbor data structures for several variations of the L_p distance for functions (for $p = 1, 2$). By applying our schemes to the turning functions of a collection of polygons we obtain efficient near neighbor LSH-based structures for polygons. To tune our structures to turning functions of polygons, we prove some new properties of these turning functions that may be of independent interest.

As part of our analysis, we address the following problem which is of independent interest. Find the vertical translation of a function f that is closest in L_1 distance to a function g. We prove tight bounds on the approximation guarantee obtained by the translation which is equal to the difference between the averages of g and f.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Computational geometry; Information systems → Information retrieval

Keywords and phrases Locality sensitive hashing, polygons, turning function, L_p distance, nearest neighbors, similarity search

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.46

Related Version *Full Version*: `https://arxiv.org/abs/2101.04339`

Funding This work was supported by ISF grant no. 1595/19, GIF grant no. 1367/2017 and the Blavatnik Foundation.

1 Introduction

This paper focuses on similarity search between polygons, where we aim to efficiently retrieve polygons with a shape resembling the query polygon.

Large image databases are used in many multimedia applications in fields such as computer vision, pattern matching, content-based image retrieval, medical diagnosis and geographical information systems. Retrieving images by their content in an efficient and effective manner has therefore become an important task, which is of rising interest in recent years.

When designing content-based image retrieval systems for large databases, the following properties are typically desired:

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 46; pp. 46:1–46:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- **Efficiency:** Since the database is very large, iterating over all objects is not feasible, so an efficient indexing data structure is necessary.
- **Human perception:** The retrieved objects should be perceptually similar to the query.
- **Invariance to transformations:** The retrieval probability of an object should be invariant to translating, scaling, and rotating the object. Moreover, since shapes are typically defined by a time signal describing their boundary, we desire invariance also to the initial point of the boundary parametrization.

There are two general methods to define how much two images are similar (or distant): intensity-based (color and texture) and geometry-based (shape). The latter method is arguably more intuitive [17] but more difficult since capturing the shape is a more complex task than representing color and texture features. Shape matching has been approached in several other ways, including tree pruning [18], the generalized Hough transform [5], geometric hashing [16] and Fourier descriptors [20]. For an extensive survey on shape matching metrics see Veltkamp and Hagedoorn [19].

A noteworthy distance function between shapes is that of Arkin et al. [2], which represents a curve using a cumulative angle function. Applied to polygons, the *turning function* (as used by Arkin et al. [2]) t_P of a polygon P returns the cumulative angle between the polygon's counterclockwise tangent at the point and the x-axis, as a function of the fraction x of the perimeter (scaled to be of length 1) that we have traversed in a counterclockwise fashion. The turning function is a step function that changes at the vertices of the polygon, and either increases with left turns, or decreases with right turns (see Figure 2). Clearly, this function is invariant under translation and scale of the polygon.

To find similar polygons based on their turning functions, we define the distance $L_p(P, Q)$ between polygons P and Q to be the L_p distance between their turning functions $t_P(x)$ and $t_Q(x)$. That is

$$L_p(P,Q) = \left(\int_0^1 |t_P(x) - t_Q(x)|^p \right)^{1/p}.$$

The turning function $t_P(x)$ depends on the rotation of P, and the (starting) point of P where we start accumulating the angle. If the polygon is rotated by an angle α, then the turning function $t_P(x)$ becomes $t_P(x) + \alpha$. Therefore, we define the (rotation invariant) distance $D_p^{\updownarrow}(P, Q)$ between polygons P and Q to be the $D_p^{\updownarrow}$ distance between their turning functions t_P and t_Q, which is defined as follows

$$D_p^{\updownarrow}(P,Q) \overset{def}{=} D_p^{\updownarrow}(t_P, t_Q) \overset{def}{=} \min_{\alpha \in \mathbb{R}} L_p(t_P + \alpha, t_Q) = \min_{\alpha \in \mathbb{R}} \sqrt[p]{\int_0^1 |t_P(x) + \alpha - t_Q(x)|^p \, dx}.$$

If the starting point of P is clockwise shifted along the boundary by t, the turning function $t_P(x)$ becomes $t_P(x + t)$. Thus, we define the distance $D_p(P, Q)$ between polygons P and Q to be the D_p distance between their turning functions t_P and t_Q which is defined as follows

$$D_p(P,Q) \overset{def}{=} D_p(t_P, t_Q) \overset{def}{=} \min_{\alpha \in \mathbb{R}, t \in [0,1]} \left(\int_0^1 |t_P(x+t) + \alpha - t_Q(x)|^p \right)^{1/p}.$$

The distance $D_p(f, g)$ between two functions f and g extends f to the domain $[0, 2]$ by defining $t_P(x + 1) = t_P(x) + 2\pi$. The distance metric D_p is invariant under translation, rotation, scaling and the selection of the starting point. A comprehensive presentation of these distances, as well as a proof that they indeed satisfy the metric axioms appears in [2].

We develop efficient nearest neighbor data structures for functions under these distances and then specialize them to functions which are turning functions of polygons.

Since a major application of polygon similarity is content-based image retrieval from large databases (see Arkin et al. [2]), the efficiency of the retrieval is a critical metric. Traditionally, efficient retrieval schemes used tree-based indexing mechanisms, which are known to work well for prevalent distances (such as the Euclidean distance) and in low dimensions. Unfortunately such methods do not scale well to higher dimensions and do not support more general and computationally intensive metrics. To cope with this phenomenon (known as the "curse of dimensionality"), Indyk and Motwani [15, 14] introduced Locality Sensitive Hashing (LSH), a framework based on hash functions for which the probability of hash collision is higher for near points than for far points.

Using such hash functions, one can determine near neighbors by hashing the query point and retrieving the data points stored in its bucket. Typically, we concatenate hash functions to reduce false positives, and use several hash functions to reduce false negatives. This gives rise to a data structure which satisfies the following property: for any query point q, if there exists a neighbor of distance at most r to q in the database, it retrieves (with constant probability) a neighbor of distance at most cr to q for some constant $c > 1$. This data structure is parameterized by the parameter $\rho = \frac{\log(p_1)}{\log(p_2)} < 1$, where p_1 is the minimal collision probability for any two points of distance at most r, and p_2 is the maximal collision probability for any two points of distance at least cr. The data structure can be built in time and space $O(n^{1+\rho})$, and its query time is $O(n^\rho \log_{1/p_2}(n))$ where n is the size of the data set.[1]

The trivial retrieval algorithm based on the turning function distance of Arkin et al. [2], is to directly compute the distance $D_2(P, Q)$ (or $D_1(P, Q)$) between the query Q and all the polygons P in the database. This solution is invariant to transformations but not efficient (i.e., linear in the size of the database).

In this paper, we rely on the turning function distance of Arkin et al. [2] for $p = 1, 2$, and create the first retrieval algorithm with respect to the turning function distance which is sub-linear in the size of the dataset. To do so, we design and analyze LSH retrieval structures for function distance, and feed the turning functions of the polygons to them. Our results give rise to a shape-based content retrieval (a near neighbor polygon) scheme which is efficient, invariant to transformations, and returns perceptually similar results.

Our contribution

We develop simple but powerful (r, cr)-LSH near neighbor data structures for efficient similar polygon retrieval, and give a theoretical analysis of their performance. We give the first structure (to the best of our knowledge) for approximate similar polygon retrieval which is provably invariant to shape rotation, translation and scale, and with a query time which is sub-linear in the number of data polygons. In contrast to many other structures for similar shape retrieval which often use heuristics, all our results are backed with theoretical proofs, using properties of the turning function distance and the theory of LSH.

[1] To ease on the reader, in this paper we suppress the term $1/p_1$ in the structure efficiency, and the time it takes to compute a hash and distances between two polygons/functions. For example for polygons with at most m vertices (which we call m-gons), all our hash computations take $O(m)$ time, and using Arkin et al. [2] we may compute distances in $O(m^2 \log(m))$ time.

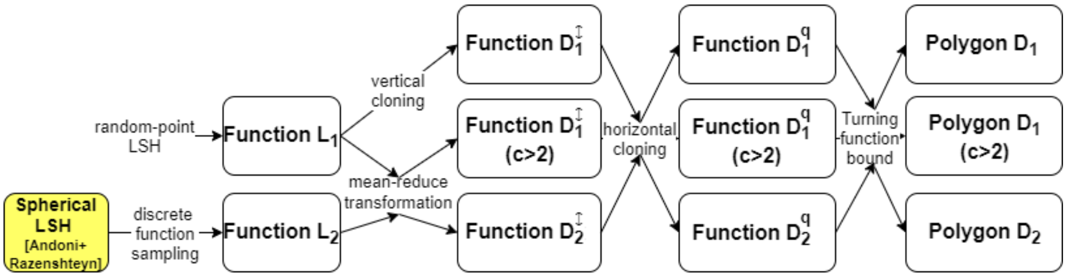


Figure 1 Our structures: each box is an (r, cr)-LSH near neighbor data structure, and the arrow $A \to B$ with label t signifies that we use the method t over the structure A to get a structure for B.

To give our (r, cr)-LSH near neighbor data structures for polygons, we build such structures for step functions with distances which are derived from the L_p distance for $p = 1, 2$, and apply them to turning functions of polygons.[2] Here $r > 0$ and $c > 1$ are the LSH parameters as defined above, and n is the number of objects in the data structure. The (r, cr)-LSH data structures which we present exist for any $r > 0$ and $c > 1$ (except when c is explicitly constrained). For an interval I, we say that a function $f : I \to \mathbb{R}$ is a k-step function, if I can be divided into k sub-intervals, such that over each sub-interval f is constant. All the following results for functions are for k-step functions with ranges bounded in $[a, b]$ for some $a < b$ where for simplicity of presentation, we fix $a = 0$ and $b = 1$.[3,4] The results we present below are slightly simplified versions than those that appear in the body of the paper. For an overview of our structures see Figure 1.

Near neighbors data structures for functions

1. For the L_1 distance over functions, we design a simple but powerful LSH hash family. This hash selects a uniform point p from the rectangle $[0, 1] \times [0, 1]$, and maps each function to 1, 0 or -1 based on its vertical relation (above, on or below) with p. This yields an (r, cr)-LSH structure for L_1 which requires sub-quadratic preprocessing time and space of $O(n^{1+\rho})$, and sub-linear query time of $O(n^\rho \log n)$, where $\rho = \log(1 - r)/\log(1 - cr) \leq \frac{1}{c}$. For the L_2 distance over functions, we observe that sampling each function at evenly spaced points reduces the L_2 distance to Euclidean distance. We use the data structure of Andoni and Razenshteyn [1] for the Euclidean distance to give an (r, cr)-LSH for the L_2 distance, which requires sub-quadratic preprocessing time of $O(n^{1+\rho} + n_{r,c} \cdot n)$, sub-quadratic space of $O(n_{r,c} \cdot n^{1+\rho})$ and sub-linear query time of $O(n_{r,c} \cdot n^\rho)$, where $\rho = \frac{1}{2c-1}$ and $n_{r,c} = \frac{2k}{(\sqrt{c}-1)r^2}$ is the dimension of the sampled vectors. We also give an alternative asymmetric LSH hash family for the L_2 distance inspired by our hash family for the L_1 distance, and create an LSH structure based on it.

2. For the $D_2^{\updownarrow}$ distance, we leverage a result of Arkin et al. [2], to show that the mean-reduce transformation, defined to be $\hat{\phi}(x) = \phi(x) - \int_0^1 \phi(s)ds$, reduces $D_2^{\updownarrow}$ distances to L_2 distances with no approximation loss. That is, for every f and g, $D_2^{\updownarrow}(f, g) = L_2(\hat{f}, \hat{g})$, so we get an

[2] Our structures for step functions can be extended to support also functions which are concatenations of at most $k \in \mathbb{N}$ functions which are M-Lipschitz for some $M > 0$. Also, we can give similar structures for variations of the function D_1 and D_2 distances where we extend the functions from the domain $[0, 1]$ to the domain $[0, 2]$, not by $f(x) = f(x - 1) + 2\pi$, but by $f(x) = f(x - 1) + q$ for any constant $q \in \mathbb{R}$.

[3] For general values of these parameters, the dependency of the data structure's run-time and memory is roughly linear or squared in $b - a$.

[4] Since $a = 0$ and $b = 1$, the distance between any two functions is at most 1, so we focus on $r < 1$.

(r, cr)-LSH structure for the $D_2^{\updownarrow}$ distance which uses our previous L_2 structure, and with identical performance. For the $D_1^{\updownarrow}$ distance, we approximately reduce $D_1^{\updownarrow}$ distances to L_1 distances using the same mean-reduction. We give a simple proof that this reduction gives a 2-approximation, and improve it to a tight approximation bound showing that for any two step functions $f, g : [0,1] \to [0,1]$, $L_1(\hat{f}, \hat{g}) \leq \left(2 - D_1^{\updownarrow}(f,g)\right) \cdot D_1^{\updownarrow}(f,g)$. This proof (see full version), which is of independent interest, characterizes the approximation ratio by considering the function $f - g$, dividing its domain into 3 parts and averaging over each part, thereby considering a single function with 3 step heights. This approximation scheme yields an (r, cr)-LSH structure for any $c > 2 - r$, which is substantially smaller than 2 (approaching 1) for large values of r.

We also give an alternative structure *step-shift-LSH* that supports any $c > 1$, but has a slightly diminished performance. This structure leans on the observation of Arkin et al. [2], that the optimal vertical shift aligns a step of f with a step of g. It therefore replaces each data step function by a set of vertical shifts of it, each aligning a different step value to $y = 0$, and constructs an L_1 data structure containing all these shifted functions. It then replaces a query with its set of shifts as above, and performs a query in the internal L_1 structure with each of these shifts.

3. For the D_1 and D_2 distances, we leverage another result of Arkin et al. [2], that the optimal horizontal shift horizontally aligns a discontinuity point of f with a discontinuity point of g. Similarly to *step-shift-LSH*, we give a structure for D_1 (or D_2) by keeping an internal structure for $D_1^{\updownarrow}$ (or $D_2^{\updownarrow}$) which holds a set of horizontal shifts of each data functions, each aligns a different discontinuity point in to $x = 0$. It then replaces a query with its set of shifts as above, and performs a query in the internal structure with each of these shifts.

Near neighbors data structures for polygons

We design LSH structures for the polygonal D_1 and D_2 distances, by applying the D_1 and D_2 structures to the turning functions of the polygons. We assume that all the data and query polygons have at most m vertices (are m-gons), where m is a constant known at preprocessing time. It is clear that the turning functions are $(m+1)$-step functions, but the range of the turning functions is not immediate (note that performance inversely relates to the range size).

First, we show that turning functions of m-gons are bounded in the interval $I = [-(\lfloor m/2 \rfloor - 1)\pi, (\lfloor m/2 \rfloor + 3)\pi]$ of size $\lambda_m := (2 \cdot \lfloor m/2 \rfloor + 2)\pi$. We show that this bound is tight in the sense that there are m-gons whose turning functions get arbitrarily close to these upper and lower bounds.

Second, we define the *span* of a function $\xi : [0,1] \to \mathbb{R}$ to be $\mathit{span}(\xi) = \max_{x \in [0,1]}(\xi(x)) - \min_{x \in [0,1]}(\xi(x))$, and show that for m-gons, the span is at most $\lambda_m/2 = (\lfloor m/2 \rfloor + 1)\pi$, and that this bound is tight - there are m-gons whose turning functions have arbitrarily close spans to $\lambda_m/2$. Since the D_1 and D_2 distances are invariant to vertical shifts, we perform an a priori vertical shift to each turning function such that its minimal value becomes 0, effectively morphing the range to $[0, \lambda_m/2]$, which is half the original range size. This yields the following structures:

For the D_1 distance, for any $c > 2$ we give an (r, cr)-LSH structure storing n polygons with at most m vertices which requires $O((nm)^{1+\rho})$ preprocessing time and space which are sub-quadratic in n, and $O(m^{1+\rho} n^{\rho} \log(nm))$ query time which is sub-linear in n, where ρ is roughly $2/c$. Also for D_1, for any $c > 1$ we get an (r, cr)-LSH structure which requires sub-quadratic preprocessing time and space of $O((nm^2)^{1+\rho})$, and sub-linear query time of $O(m^{2+2\rho} n^{\rho} \log(nm))$, where ρ is roughly $1/c$.

For the D_2 distance, we give an (r, cr)-LSH structure which requires sub-quadratic preprocessing time of $\tilde{O}(n^{1+\rho})$, sub-quadratic space of $\tilde{O}(n^{1+\rho})$, and sub-linear query time of $\tilde{O}(n^{\rho})$, where $\rho = \frac{1}{2\sqrt{c}-1}$.[5]

Other similar works

Babenko et al. [4] suggest a practical method for similar image retrieval, by embedding images to a Euclidean space using Convolutional Neural Networks (CNNs), and retrieving similar images to a given query based on their embedding's euclidean distance to the query embedding. This approach has been the most effective practical approach for similar image retrieval in recent years.

Gudmundsson and Pagh [13] consider a metric in which there is a constant grid of points, and shapes are represented by the subset of grid points which are contained in them. The distance between polygons is then defined to be the *Jaccard distance* between the corresponding subsets of grid points. Their solution lacks invariance to scale, translation and rotation, however our work is invariant to those, and enables retrieving polygons which have a similar shape, rather than only spatially similar ones.

Other metrics over shapes have been considered. Cakmakov et al. [7] defined a metric based on snake-like moving of the curves. Bartolini et al. [6] proposed a new distance function between shapes, which is based on the Discrete Fourier Transform and the Dynamic Time Warping distance. Chavez et al. [9] give an efficient polygon retrieval technique based on Fourier descriptors. Their distance works for exact matches, but is a weak proxy for visual similarity, since it relates to the distances between corresponding vertices of the polygons.

There has been a particular effort to develop efficient structures for the discrete Fréchet distance and the dynamic time warping distance for polygonal curves in $\mathbb{R}^d$. Such works include Driemel et al. [10] who gave LSH structures for these metrics via snapping the curve points to a grid, Ceccarello et al. [8] who gave a practical and efficient algorithm for the r-range search for the discrete Fréchet distance, Filtser et al. [11] who built a deterministic approximate near neighbor data structure for these metrics using a subsample of the data, and Astefanoaei et al. [3] who created a suite of efficient sketches for trajectory data. Grauman and Darrell [12] performed efficient contour-based shape retrieval (which is sensitive (not invariant) to translations, rotations and scaling) using an embedding of Earth Mover's Distance into L_1 space and LSH.

2 Preliminaries

We first formally define LSH, then discuss the turning function representation of Arkin et al. [2], and then define the distance functions between polygons and functions which rise from this representation.

2.1 Locality sensitive hashing

We use the following standard definition of a *Locality Sensitive Hash Family (LSH)* with respect to a given distance function $d : Z \times Z \to \mathbb{R}_{\geq 0}$.

[5] The $\tilde{O}$ notation hides multiplicative constants which are small powers (e.g., 5) of m, $\frac{1}{r}$ and $\frac{1}{\sqrt[4]{c}-1}$.

▶ **Definition 1** (Locality Sensitive Hashing (LSH)). *Let $r > 0$, $c > 1$ and $p_1 > p_2$. A family H of functions $h : Z \to \Gamma$ is an (r, cr, p_1, p_2)-LSH for a distance function $d : Z \times Z \to \mathbb{R}_{\geq 0}$ if for any $x, y \in Z$,*

1. *If $d(x, y) \leq r$ then $\Pr_{h \in H}[h(x) = h(y)] \geq p_1$, and*
2. *If $d(x, y) \geq cr$ then $\Pr_{h \in H}[h(x) = h(y)] \leq p_2$.*

Note that in the definition above, and in all the following definitions, the hash family H is always sampled uniformly.

We say that a hash family is an (r, cr)*-LSH* for a distance function d if there exist $p_1 > p_2$ such that it is an (r, cr, p_1, p_2)-LSH. A hash family is a *universal LSH* for a distance function d if for all $r > 0$ and $c > 1$ it is an (r, cr)-LSH.

From an (r, cr, p_1, p_2)-LSH family, we can derive, via the general theory developed in [15, 14], an (r, cr)*-LSH data structure*, for finding approximate near neighbors with respect to r. That is a data structure that finds (with constant probability) a neighbor of distance at most cr to a query q if there is a neighbor of distance at most r to q. This data structure uses $O(n^{1+\rho})$ space (in addition to the data points), and $O(n^{\rho} \log_{1/p_2}(n))$ hash computations per query, where $\rho = \frac{\log(1/p_1)}{\log(1/p_2)} = \frac{\log(p_1)}{\log(p_2)}$.

2.2 Representation of polygons

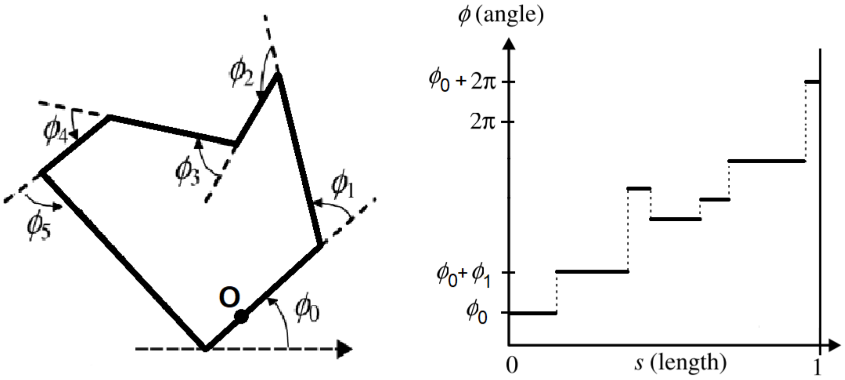


Figure 2 Left: a polygon P with 6 vertices. Right: the turning function t_P of P, with 7 steps.

Let P be a simple polygon scaled such that its perimeter is one. Following the work of Arkin et al. [2], we represent P via a *turning function* $t_P(s) : [0, 1] \to \mathbb{R}$, that specifies the angle of the counterclockwise tangent to P with the x-axis, for each point q on the boundary of P. A point q on the boundary of P is identified by its counterclockwise distance (along the boundary which is of length 1 by our scaling) from some fixed reference point O. It follows that $t_P(0)$ is the angle α that the tangent at O creates with the x-axis, and $t_P(s)$ follows the cumulative turning, and increases with left turns and decreases with right turns. Although t_P may become large or small, since P is a simple closed polygon we must have that $t_P(1) = t_P(0) + 2\pi$ if O is not a vertex of P, and $t_P(1) - t_P(0) \in [\pi, 3\pi]$ otherwise. Figure 2 illustrates the polygon turning function.

Note that since the angle of an edge with the x-axis is constant and angles change at the vertices of P, then the function is constant over the edges of P and has discontinuity points over the vertices. Thus, the turning function is in fact a step function.

In this paper, we often use the term m-gon – a polygon with **at most** m vertices.

2.3 Distance functions

Consider two polygons P and Q, and their associated turning functions $t_P(s)$ and $t_Q(s)$ accordingly. Define the *aligned* L_p *distance* (often abbreviated to L_p *distance*) between P and Q denoted by $L_p(P,Q)$, to be the L_p distance between $t_P(s)$ and $t_Q(s)$ in $[0,1]$: $L_p(P,Q) = \sqrt[p]{\int_0^1 |t_P(x) - t_Q(x)|^p \, dx}$.

Note that even though the L_p distance between polygons is invariant under scale and translation of the polygon, it depends on the rotation of the polygon and the choice of the reference points on the boundaries of P and Q.

Since rotation of the polygon results in a vertical shift of the function t_P, we define the *vertical shift-invariant* L_p distance between two functions f and g to be $D_p^{\updownarrow}(f,g) = \min_{\alpha\in\mathbb{R}} L_p(f+\alpha, g) = \min_{\alpha\in\mathbb{R}} \sqrt[p]{\int_0^1 |f(x) + \alpha - g(x)|^p \, dx}$. Accordingly, we define the *rotation-invariant* L_p distance between two polygons P and Q to be the vertical shift-invariant L_p distance between the turning functions t_P and t_Q of P and Q respectively: $D_p^{\updownarrow}(P,Q) = D_p^{\updownarrow}(t_P, t_Q) = \min_{\alpha\in\mathbb{R}} \sqrt[p]{\int_0^1 |t_P(x) + \alpha - t_Q(x)|^p \, dx}$.

To tweak the distance $D_p^{\updownarrow}$ such that it will be invariant to changes of the reference points, we need the following definition. We define the 2π*-extension* $f^{2\pi} : [0,2] \to \mathbb{R}$ of a function $f : [0,1] \to \mathbb{R}$ to the domain $[0,2]$, to be $f^{2\pi} = \begin{cases} f(x), & \text{for } x \in [0,1] \\ f(x-1) + 2\pi, & \text{for } x \in (1,2] \end{cases}$.

A turning function t_P is naturally 2π-extended to the domain $[0,2]$ by circling around P one more time. We define the u-slide of a function $g : [0,2] \to \mathbb{R}$, $slide_u^{\leftrightarrow}(g) : [0,1] \to \mathbb{R}$, for a value $u \in [0,1]$ to be $(slide_u^{\leftrightarrow}(g))(x) = g(x+u)$. These definitions are illustrated in Figure 3. Note that shifting the reference point by a counterclockwise distance of u around the perimeter of a polygon P changes the turning function from t_P to $slide_u^{\leftrightarrow}(t_P^{2\pi})$.

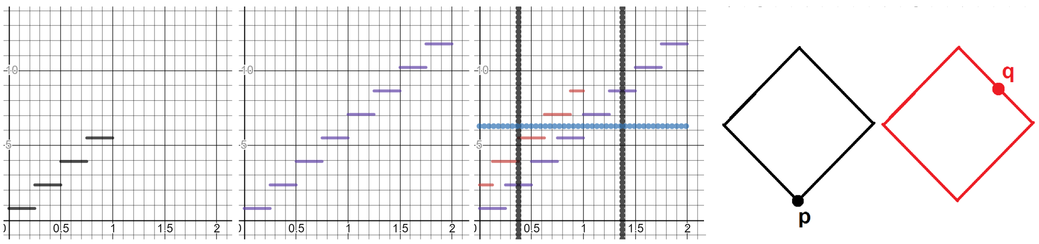


Figure 3 Left: The turning function t_P of the square with reference point p. Center: the 2π-extension $t_P^{2\pi}$ of t_P. Right: The turning function of the square with the reference point q in red (this is in fact the function $t_P^{2\pi}$ cropped to between the black vertical lines, i.e., to $[0.375, 1.375]$).

We therefore define the (vertical and horizontal) *shift-invariant* L_p *distance* between two functions $f, g : [0,1] \to \mathbb{R}$ to be: $D_p(f,g) = \min_{u\in[0,1]} D_p^{\updownarrow}(slide_u^{\leftrightarrow}(f^{2\pi}), g) = \min_{\alpha\in\mathbb{R},\ u\in[0,1]} \sqrt[p]{\int_0^1 |f^{2\pi}(x+u) + \alpha - g(x)|^p \, dx}$, and define the (rotation and reference point invariant) L_p distance between two polygons P and Q to be $D_p(P,Q) = D_p(t_P, t_Q)$. Arkin et al. [2] proved that $D_p(f,g)$ is a metric for any $p > 0$.

3 L_1-based distances

In this section, we give LSH structures for the L_1 distance, the $D_1^{\updownarrow}$ distance and then the D_1 distance. Note that the D_1 distance reduces to the $D_1^{\updownarrow}$ distance, which by using the *mean-reduction* transformation presented in Section 3.2, reduces to the L_1 distance.

3.1 Structure for L_1

In this section we present *random-point-LSH*, a simple hash family for functions $f : [0,1] \to [a,b]$ with respect to the L_1 distance. *Random-point-LSH* is the hash family $H_1(a,b) = \{h_{(x,y)} \mid (x,y) \in [0,1] \times [a,b]\}$, where the points (x,y) are uniformly selected from the rectangle $[0,1] \times [a,b]$. Each $h_{(x,y)}$ receives a function $f : [0,1] \to [a,b]$, and returns 1 if f is vertically above the point (x,y), returns -1 if f is vertically below (x,y), and 0 otherwise.

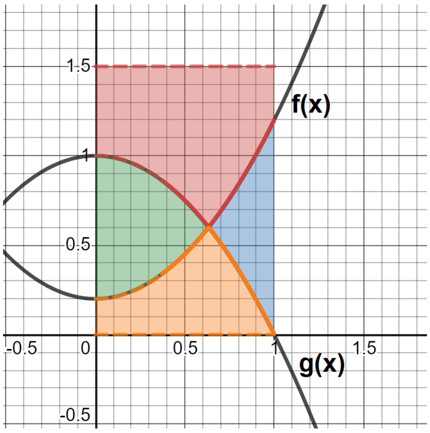


Figure 4 Illustration of the hash of two functions f and g w.r.t. $h_{(x,y)}$ for $a = 0$ and $b = 1.5$. For (x,y) in the green area $h_{(x,y)}(f) = -1 \neq 1 = h_{(x,y)}(g)$, in the blue area $h_{(x,y)}(f) = 1 \neq -1 = h_{(x,y)}(g)$, in the red area $h_{(x,y)}(f) = h_{(x,y)}(g) = -1$, and in the orange area $h_{(x,y)}(f) = h_{(x,y)}(g) = 1$.

The intuition behind *random-point-LSH* is that any two functions $f, g : [0,1] \to [a,b]$ collide precisely over hash functions $h_{(x,y)}$ for which the point (x,y) is outside the area bounded between the graphs of f and g. This fact is illustrated in the following Figure 4. Thus, this hash incurs a collision probability of $1 - \frac{L_1(f,g)}{b-a} = 1 - \frac{L_1(f,g)}{b-a}$, which is a decreasing function with respect to $L_1(f,g)$. This intuition leads to the following results.

▶ **Theorem 2.** *For any two functions $f, g : [0,1] \to [a,b]$, we have that $P_{h \sim H_1(a,b)}(h(f) = h(g)) = 1 - \frac{L_1(f,g)}{b-a}$.*

Proof. Fix $x \in [0,1]$, and denote by $U(S)$ the uniform distribution over a set S. We have that

$$\begin{aligned} P_{y \sim U([a,b])}(h_{(x,y)}(f) = h_{(x,y)}(g)) &= 1 - P_{y \sim U([a,b])}(h_{(x,y)}(f) \neq h_{(x,y)}(g)) \\ &= 1 - \frac{|f(x) - g(x)|}{b-a}, \end{aligned}$$

where the last equality follows since $h_{(x,y)}(f) \neq h_{(x,y)}(g)$ precisely for the y values between $f(x)$ and $g(x)$. Therefore, by the law of total probability,

$$\begin{aligned} P_{h \sim H_1(a,b)}(h(f) = h(g)) &= P_{(x,y) \sim U([0,1] \times [a,b])}(h_{(x,y)}(f) = h_{(x,y)}(g)) \\ &= \int_0^1 P_{y \sim U([a,b])}(h_{(x,y)}(f) = h_{(x,y)}(g))dx \\ &= \int_0^1 \left(1 - \frac{|f(x) - g(x)|}{b-a}\right) dx = 1 - \frac{L_1(f,g)}{b-a}. \end{aligned}$$ ◀

▶ **Corollary 3.** *For any $r > 0$ and $c > 1$, one can construct an $(r, cr)-$LSH structure for the L_1 distance for n functions with ranges bounded in $[a, b]$. This structure requires $O(n^{1+\rho})$ space and preprocessing time, and has $O(n^\rho \log(n))$ query time, where $\rho = \frac{\log\left(1-\frac{r}{b-a}\right)}{\log\left(1-\frac{cr}{b-a}\right)} \approx \frac{1}{c}$ for $r \ll b - a$.*

Proof. Fix $r > 0$ and $c > 1$. By the general result of Indyk and Motwani [15], it suffices to show that $H_1(a, b)$ is an $(r, cr, 1 - \frac{r}{b-a}, 1 - \frac{cr}{b-a})$-LSH for the L_1 distance.

Indeed, by Theorem 2, $P_{h\sim H_1(a,b)}(h(f) = h(g)) = 1 - \frac{L_1(f,g)}{b-a}$, so we get that

- If $L_1(f, g) \leq r$, then $P_{h\sim H_1(a,b)}(h(f) = h(g)) = 1 - \frac{L_1(f,g)}{b-a} \geq 1 - \frac{r}{b-a}$.
- If $L_1(f, g) \geq cr$, then $P_{h\sim H_1(a,b)}(h(f) = h(g)) = 1 - \frac{L_1(f,g)}{b-a} \leq 1 - \frac{cr}{b-a}$. ◀

3.2 Structure for $D_1^{\updownarrow}$

In this section we present *mean-reduce-LSH*, an LSH family for the vertical translation-invariant L_1 distance, $D_1^{\updownarrow}$. Observe that finding an LSH family for $D_1^{\updownarrow}$ is inherently more difficult than for L_1, since even evaluating $D_1^{\updownarrow}(f, g)$ for a query function g and an input function f requires minimizing $L_1(f + \alpha, g)$ over the variable α, and the optimal value of α depends on both f and g.

Our structure requires the following definitions. We define $\bar{\phi} = \int_0^1 \phi(x)dx$ to be the mean of a function ϕ over the domain $[0, 1]$, and define the *mean-reduction* of ϕ, denoted by $\hat{\phi} : [0, 1] \to [a - b, b - a]$, to be the vertical shift of ϕ with zero integral over $[0, 1]$, i.e., $\hat{\phi}(x) = \phi(x) - \bar{\phi}(x)$. These definitions are illustrated in Figure 5. Our solution relies on the crucial observation that for the pair of functions $f, g : [0, 1] \to [a, b]$, the value of α which minimizes $L_1(f + \alpha, g)$ is "well approximated" by $\bar{g} - \bar{f}$. That is the distance $L_1(f + (\bar{g} - \bar{f}), g) = L_1(f - \bar{f}, g - \bar{g}) = L_1(\hat{f}, \hat{g})$ approximates $D_1^{\updownarrow}(f, g)$. This suggests that if we replace any data or query function f with $\hat{f}$, then the $D_1^{\updownarrow}$ distances are approximately the L_1 distances of the shifted versions $\hat{f}$, for which we can use the hash H_1 from Section 3.1.

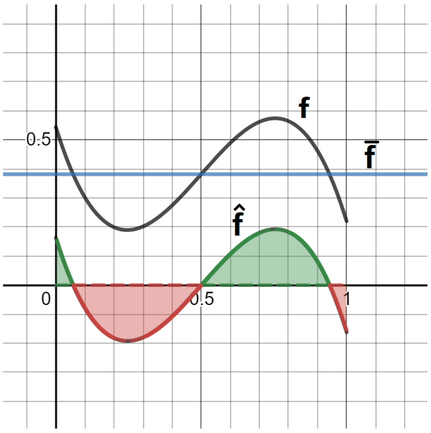


Figure 5 A function f (black), its mean $\bar{f}$(blue), and its mean-reduction $\hat{f}$ (below). Notice that the red and green areas are equal.

Indeed, we use the hash family H_1 from Section 3.1, and define *mean-reduce-LSH* for functions with images contained in $[a, b]$ to be the family $H_1^{\updownarrow}(a, b) = \{f \to h \circ \hat{f} \mid h \in H_1(a - b, b - a)\}$. Each hash of $H_1^{\updownarrow}(a, b)$ is defined by a function $h \in H_1(a - b, b - a)$, and given a function f, it applies h on its mean-reduction $\hat{f}$.

The following theorem gives a tight bound for the L_1 distance between mean-reduced functions in terms of their original vertical translation-invariant L_1 distance $D_1^{\updownarrow}$. The proof of this tight bound as well as a simpler 2-approximation appear in the full version of the paper. Our elegant but more complicated proof of the tight bound characterizes and bounds the approximation ratio using properties of $f - g$, and demonstrates its tightness by giving the pair of step functions f, g which meet the bound.

We conclude this result in the following theorem.

▶ **Theorem 4.** *Let $f, g : [0,1] \to [a,b]$ be step functions and let $r \in (0, b-a]$ be their vertical shift-invariant L_1 distance $r = D_1^{\updownarrow}(f,g)$. Then $r \le L_1(\hat{f},\hat{g}) \le \left(2 - \frac{r}{b-a}\right) \cdot r$. This bound is tight, i.e, there exist two functions f_0, g_0 as above for which $L_1(\hat{f_0},\hat{g_0}) = \left(2 - \frac{r}{b-a}\right) \cdot r$.*

We use Theorem 4 to prove that *mean-reduce-LSH* is an LSH family (Theorem 5). We then use Theorem 5 and the general result of Indyk and Motwani [15] to get Corollary 6.

▶ **Theorem 5.** *For any $r \in (0, b-a)$ and $c > 2 - \frac{r}{b-a}$, $H_1^{\updownarrow}(a,b)$ is an $\left(r, cr, 1 - \left(2 - \frac{r}{b-a}\right) \cdot \frac{r}{2(b-a)}, 1 - c \cdot \frac{r}{2(b-a)}\right)$-LSH family for the $D_1^{\updownarrow}$ distance.*

▶ **Corollary 6.** *For any $r > 0$ and $c > 2 - \frac{r}{b-a}$, one can construct an $(r, cr)-$LSH structure for the $D_1^{\updownarrow}$ distance for n functions with ranges bounded in $[a,b]$. This structure requires $O(n^{1+\rho})$ extra space and preprocessing time, and $O(n^{\rho}\log(n))$ query time, where $\tilde{r} = r/(2(b-a))$ and $\rho = \log\left(1 - (2 - 2\tilde{r}) \cdot \tilde{r}\right) / \log\left(1 - c\tilde{r}\right)$ for small $\tilde{r}$.*

Step-shift-LSH

We present *step-shift-LSH*, a structure for the $D_1^{\updownarrow}$ distance which works for any $c > 1$ (unlike *mean-reduce-LSH*), but has a slightly worse performance, which depends on an upper bound k on the number of steps in of the data and query functions. This structure uses an internal structure for the L_1 distance, and leverages the observation of Arkin et al. [2] that the optimal vertical shift α to align two step functions f and g, is such that $f + \alpha$ has a step which partially overlaps a step of g, i.e., there is some segment $S \subseteq [0,1]$ over which $f + \alpha = g$.

Therefore, we overcome the uncertainty of the optimal α by a priori cloning each function by the number of steps it has, and vertically shifting each clone differently to align each step to be at $y = 0$.[6] For a query function g, we clone it similarly to align each step to $y = 0$, and use each clone as a separate query for the L_1 structure. This process effectively gives a chance to align each step of the query g with each step of each data step function f.

▶ **Corollary 7.** *For any $a < b$, $r > 0$ and $c > 1$, there exists an (r, cr)-LSH structure for the $D_1^{\updownarrow}$ distance for n functions, each of which is a k-step function with range bounded in $[a,b]$. This structure requires $O((nk)^{1+\rho})$ extra space and preprocessing time, and $O(k^{1+\rho}n^{\rho}\log(nk))$ query time, where $\rho = \log\left(1 - \frac{r}{2(b-a)}\right) / \log\left(1 - \frac{cr}{2(b-a)}\right) \approx \frac{1}{c}$ for $r \ll b-a$.*

3.3 Structure for D_1

In this section, we present *slide-clone-LSH*, a data structure for the distance function D_1 defined over step functions $f : [0,1] \to [a,b]$. To do so, we use an $(r', c'r')$-LSH data structure (for appropriate values of r' and c') for the distance function $D_1^{\updownarrow}$ which will hold slided functions with ranges contained in $[a, b + 2\pi]$.

[6] This idea of cloning appears once again (but in a horizontal version), and in more detail, in Section 3.3 for the D_1 distance.

Recall that the D_1 distance between a data function f and a query function g is defined to be the minimal $D_1^{\updownarrow}$ distance between a function in the set $\left\{slide_u^{\leftrightarrow}(f^{2\pi}) \mid u \in [0,1]\right\}$ and the function g, and we obviously do not know u a priori and cannot build a structure for each possible $u \in [0,1]$. Fortunately, in the proof of Theorem 6 from Arkin et al. [2], they show that for any pair of step functions f and g, the optimal slide u is such that a discontinuity of f is aligned with a discontinuity of g. They show that this is true also for the D_2 distance.

Therefore, we can overcome the uncertainty of the optimal u by a priori cloning each function by the number of discontinuity points it has, and sliding each clone differently to align its discontinuity point to be at $x = 0$. For a query function g, we clone it similarly to align each discontinuity point to $x = 0$, use each clone as a separate query. The above process effectively gives a chance to align each discontinuity point of the query function g with each discontinuity point of each data step function f.

Slide-clone-LSH works as follows.

Preprocessing phase

We are given the parameters $r > 0$, $c > 1$, $a < b$ and a set of step functions F, where each function is defined over the domain $[0,1]$ and has a range bounded in $[a,b]$. Additionally, we are given an upper bound k on the number of steps a data or query step function may have. First, we replace each function $f \in F$ with the set of (at most $k+1$) u slides of it's 2π-extension for each discontinuity point u, i.e., $slide_u^{\leftrightarrow}(f^{2\pi})$ for each discontinuity point $u \in [0,1]$. For each such clone we remember its original unslided function. Next, we store the at most $(k+1) \cdot |F|$ resulted functions in an $(r', c'r')$-LSH data structure for the $D_1^{\updownarrow}$ distance for functions with ranges bounded in $[a, b+2\pi]$, tuned with the parameters $r' = r$ and $c' = c$.

Query phase

Let g be a query function. We query the $D_1^{\updownarrow}$ structure constructed in the preprocessing phase with each of the slided queries $slide_u^{\leftrightarrow}(g^{2\pi})$ for each discontinuity point $u \in [0,1]$. If one of the queries returns a data function f, we return its original unslided function, and otherwise return nothing.

In Theorem 8, we prove that *slide-clone-LSH* is an (r, cr)-data structure for D_1.

▶ **Theorem 8.** *Slide-clone-LSH is an (r, cr)-LSH structure for the D_1 distance.*

▶ **Corollary 9.** *For any $a < b$, $r > 0$, $\omega = b + 2\pi - a$ and $c > 2 - \frac{r}{\omega}$, there exists an (r, cr)-LSH structure for the D_1 distance for n functions, each of which is a k-step function with range bounded in $[a,b]$. This structure requires $O((nk)^{1+\rho})$ extra space and preprocessing time, and $O(k^{1+\rho}n^{\rho}\log(nk))$ query time, where $\tilde{r} = r/(2\omega)$ and $\rho = \log\left(1 - (2 - 2\tilde{r}) \cdot \tilde{r}\right) / \log\left(1 - c\tilde{r}\right) \approx \frac{2}{c}$ for small $\tilde{r}$.*[7]

▶ **Corollary 10.** *For any $a < b$, $r > 0$ and $c > 1$, there exists an (r, cr)-LSH structure for the D_1 distance for n functions, each of which is a k-step function with range bounded in $[a,b]$. This structure requires $O((nk^2)^{1+\rho})$ extra space and preprocessing time, and $O(k^{2+2\rho}n^{\rho}\log(nk))$ query time, where $\rho = \log\left(1 - \frac{r}{2(b+2\pi-a)}\right) / \log\left(1 - \frac{cr}{2(b+2\pi-a)}\right) \approx \frac{1}{c}$ for $r \ll 2(b + 2\pi - a)$.*

[7] Given a bound s on the span of the functions, we can a priori vertically shift all the functions such that their minimum is 0, effectively making the range size smaller (within $[0,s]$) and improving the performance of the structure (see the full version).

4 L_2-based distances

This section, which appears in detail in the full version of the paper, gives LSH structures for the L_2 distance, the $D_2^{\updownarrow}$ distance and then the D_2 distance.

First, we present *discrete-sample-LSH*, a simple LSH structure for functions $f : [0,1] \to [a,b]$ with respect to the L_2 distance. The intuition behind *discrete-sample-LSH* is that the L_2 distance between the step functions $f, g : [0,1] \to [a,b]$ can be approximated via a sample of f and g at the evenly spaced set of points $\{i/n\}_{i=0}^{n}$. Specifically, by replacing each function f by the vector $vec_n(f) = \left(\frac{1}{\sqrt{n}} f\left(\frac{0}{n}\right), \frac{1}{\sqrt{n}} f\left(\frac{1}{n}\right), \ldots, \frac{1}{\sqrt{n}} f\left(\frac{n-1}{n}\right)\right)$, one can show that for a large enough value of $n \in \mathbb{N}$, $L_2(f,g)$ can be approximated by $L_2\left(vec_n(f) - vec_n(g)\right)$. We prove that for any two k-step functions $f, g : [0,1] \to [a,b]$, and for any $r > 0$ and $c > 1$: **(1)** if $L_2(f,g) \le r$ then $L_2\left(vec_{n_{r,c}}(f), vec_{n_{r,c}}(g)\right) \le c^{1/4} r$, and **(2)** if $L_2(f,g) > cr$ then $L_2\left(vec_{n_{r,c}}(f), vec_{n_{r,c}}(g)\right) > c^{3/4} r$ for a sufficiently large $n_{r,c}$ (see full version for the exact value). Note that the bounds $A = c^{1/4} r$ and $B = c^{3/4} r$ are selected for simplicity, and other trade-offs are possible. The proof of this claim relies on the observation that $(f-g)^2$ is also a step function, and that $L_2\left(vec_{n_{r,c}}(f), vec_{n_{r,c}}(g)\right)^2$ is actually the left Riemann sum of $(f-g)^2$, so as $n \to \infty$, it must approach $\int_0^1 (f(x) - g(x))^2 dx = (L_2(f,g))^2$. *Discrete-sample-LSH* replaces data and query functions f with the vector samples $vec_{n_{r,c}}(f)$, and holds an $(c^{1/4} r, c^{3/4} r)$-LSH structure for the $n_{r,c}$-dimensional Euclidean distance (e.g., the *Spherical-LSH* based structure of Andoni and Razenshteyn [1]). The resulting structure has the parameter $\rho = \frac{1}{2c-1}$.

In the full version of the paper, we present an alternative structure tailored for the L_2 distance for general (not necessarily k-step) integrable functions $f : [0,1] \to [a,b]$, based on a simple and efficiently computable asymmetric hash family which uses *random-point-LSH* as a building block. We note that this structure's ρ values are larger than those of *discrete-sample-LSH* for small values of r.

Next, we give *vertical-alignment-LSH*– a structure for $D_2^{\updownarrow}$. Recall that the mean-reduction (Section 3.2) of a function f is defined to be $\hat{f}(x) = f(x) - \int_0^1 f(t)dt$. We show that the *mean-reduction* has no approximation loss when used for reducing $D_2^{\updownarrow}$ distances to L_2 distances, i.e., it holds that $D_2^{\updownarrow}(f,g) = L_2\left(\hat{f}, \hat{g}\right)$ for any f, g. Thus, to give an (r, cr)-LSH structure for $D_2^{\updownarrow}$, *vertical-alignment-LSH* simply holds a (r, cr)-LSH structure for L_2, and translates data and query functions f for $D_2^{\updownarrow}$ to data and query functions $\hat{f}$ for L_2.

Finally, we employ the same cloning and sliding method as in Section 3.3, to obtain an (r, cr)-LSH structure for D_2 using a structure for $D_2^{\updownarrow}$.

5 Polygon distance

In this section (which appears in detail in the full version of the paper) we consider polygons, and give efficient structures to find similar polygons to an input polygon. All the results of this section depend on a fixed value $m \in \mathbb{N}$, which is an upper bound on the number of vertices in all the polygons which the structure supports (both data and query polygons). Recall that the distance functions between two polygons P and Q which we consider, are defined to be variations of the L_p distance between the turning functions t_P and t_Q of the polygons, for $p = 1, 2$. To construct efficient structures for similar polygon retrieval, we apply the structures from previous sections to the turning functions of the polygons.

To apply these structures and analyze their performance, it is necessary to bound the range of the turning functions, and represent them as k-step functions. Since the turning functions are $(m+1)$-step functions, it therefore remains to compute bounds for the range of the turning function t_P.

A coarse bound of $[-(m+1)\pi,(m+3)\pi]$ can be derived by noticing that the initial value of the turning function is in $[0,2\pi]$, that any two consecutive steps in the turning function differ by an angle less than π, and that the turning function has at most $m+1$ steps.
We give an improved and tight bound for the range of the turning function, which relies on the fact that turning functions may wind up and accumulate large angles, but they must almost completely unwind towards the end of the polygon traversal, such that $t_P(1) \in [t_P(0)+\pi, t_P(0)+3\pi]$. Our result is as follows.

▶ **Theorem 11** (Simplified). *Let P be a polygon with m vertices. Then for the turning function t_P, $\forall x \in [0,1], -(\lfloor m/2 \rfloor - 1)\pi \leq t_P(x) \leq (\lfloor m/2 \rfloor + 3)\pi$, and this bound is tight.*

We denote the lower and upper bounds on the range by $a_m = -(\lfloor m/2 \rfloor - 1)\pi$ and $b_m = (\lfloor m/2 \rfloor + 3)\pi$ respectively, and define λ_m to be the size of this range, $\lambda_m = (2 \cdot \lfloor m/2 \rfloor + 2)\pi$. Having the results above, we get LSH structures for the different corresponding polygonal distances which support polygons with at most m vertices, by simply replacing each data and query polygon by its turning function.

Regarding the distances $D_1^{\updownarrow}$ and D_1, we can improve the bound above using the crucial observation that even though the range of the turning function may be of size near $m\pi$, its span can actually only be of size approximately $\frac{m}{2} \cdot \pi$ (Theorem 12), where we define the span of a function ϕ over the domain $[0,1]$, to be $span(\phi) = \max_{x\in[0,1]}(\phi(x)) - \min_{x\in[0,1]}(\phi(x))$.

A simplified version of this result is as follows.

▶ **Theorem 12** (Simplified). *Let Q be a polygon with m vertices. Then for the turning function t_Q, it holds that $span(t_Q) \leq (\lfloor m/2 \rfloor + 1)\pi = \lambda_m/2$. Moreover, for any $\varepsilon > 0$ there exists such a polygon with span at least $(\lfloor m/2 \rfloor + 1)\pi - \varepsilon$.*

Since the $D_1^{\updownarrow}$ distance is invariant to vertical shifts, we can improve the overall performance of our $D_1^{\updownarrow}$ LSH structure by simply mapping each data and query polygon $P \in S$ to its vertically shifted turning function $x \to t_P(x) - \min_{z\in[0,1]} t_P(z)$ (such that its minimal value becomes 0). This shift morphs the ranges of the set of functions F to be contained in $[0, \max_{f\in F}(span(f))]$. By Theorem 12, we can therefore use the adjusted bounds of $a = 0$ and $b = \lambda_m/2$ (each function $f \in S_0$ is obviously non-negative, but also bounded above by $\lambda_m/2$ by Theorem 12), and effectively halve the size of the range from $\lambda_m = b_m - a_m$ to $\lambda_m/2$.

To summarize our results for polygons, we use the $\tilde{O}$ notation to hide multiplicative constants which are small powers (e.g., 5) of m, $\frac{1}{r}$, and $\frac{1}{\sqrt{c}-1}$:
For the D_1 distance, for any $c > 2$ we give an (r, cr)-LSH structure which for $r \ll \frac{2\lambda_m}{c}$ roughly requires $\tilde{O}(n^{1+\rho})$ preprocessing time and space, and $\tilde{O}(n^{1+\rho}\log n)$ query time, where ρ is roughly $\frac{2}{c}$. Also for D_1, for any $c > 1$ we get an (r, cr)-LSH structure which for $r \ll \lambda_m$ roughly requires $O((nm^2)^{1+\rho})$ preprocessing time and space, and $O(m^{2+2\rho}n^{\rho}\log(nm))$ query time, where ρ is roughly $1/c$.

For the D_2 distance, we give an (r, cr)-LSH structure which requires $\tilde{O}(n^{1+\rho})$ preprocessing time, $\tilde{O}(n^{1+\rho})$ space, and $\tilde{O}(n^{\rho})$ query time, where $\rho = \frac{1}{2\sqrt{c}-1}$.

6 Conclusions and directions for future work

We present several novel LSH structures for searching nearest neighbors of functions with respect to the L_1 and the L_2 distances, and variations of these distances which are invariant to horizontal and vertical shifts. This enables us to devise efficient similar polygon retrieval structures, by applying our nearest neighbor data structures for functions, to the turning functions of the polygons. For efficiently doing this, we establish interesting bounds on the range and span of the turning functions of m-gons.

As part of our analysis, we proved that for any two functions $f, g : [0,1] \to [a,b]$ such that $D_1^{\updownarrow}(f,g) = r$, it holds that $L_1(\hat{f}, \hat{g}) \leq \left(2 - \frac{r}{b-a}\right) \cdot r$. This tight approximation guarantee may be of independent interest. An interesting line for further research is to find near neighbor structures with tighter guarantees for simple and frequently occurring families of polygons such as rectangles, etc.

All the reductions we describe have some performance loss, which is reflected in the required space, preprocessing and query time. Finding optimal reduction parameters (e.g., an optimal value of ξ in Section 3.3 for polygons) and finding more efficient reductions is another interesting line for further research. Finding an approximation scheme for the horizontal distance (similarly to the $\left(2 - \frac{r}{b-a}\right)$-approximation for the $D_1^{\updownarrow}$ distance which appears in Section 3.2) is another intriguing open question.

References

1 Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *STOC*, pages 793–801. ACM, 2015.

2 Esther M Arkin, L Paul Chew, Daniel P Huttenlocher, Klara Kedem, and Joseph S Mitchell. An efficiently computable metric for comparing polygonal shapes. Technical report, Cornell University, 1991.

3 Maria Astefanoaei, Paul Cesaretti, Panagiota Katsikouli, Mayank Goswami, and Rik Sarkar. Multi-resolution sketches and locality sensitive hashing for fast trajectory processing. In *SIGSPATIAL*, pages 279–288. ACM, 2018.

4 Artem Babenko, Anton Slesarev, Alexandr Chigorin, and Victor Lempitsky. Neural codes for image retrieval. In *ECCV*, pages 584–599. Springer, 2014.

5 Dana H Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122, 1981.

6 Ilaria Bartolini, Paolo Ciaccia, and Marco Patella. Using the time warping distance for fourier-based shape retrieval. Technical report, IEIIT-BO-03-02, 2002.

7 Dusan Cakmakov and Emilija Celakoska. Estimation of curve similarity using turning functions. *International Journal of Applied Mathematics*, 15:403–416, 2004.

8 Matteo Ceccarello, Anne Driemel, and Francesco Silvestri. Fresh: Fréchet similarity with hashing. In *WADS*, pages 254–268. Springer, 2019.

9 Edgar Chávez, Ana C Chávez Cáliz, and Jorge L López-López. Affine invariants of generalized polygons and matching under affine transformations. *Computational Geometry*, 58:60–69, 2016.

10 Anne Driemel and Francesco Silvestri. Locality-sensitive hashing of curves. In *SOCG*, pages 37:1–37:16, 2017.

11 Arnold Filtser, Omrit Filtser, and Matthew J Katz. Approximate nearest neighbor for curves—simple, efficient, and deterministic. *arXiv preprint*, 2019. `arXiv:1902.07562`.

12 Kristen Grauman and Trevor Darrell. Fast contour matching using approximate earth mover's distance. In *CVPR*, pages I–220. IEEE, 2004.

13 Joachim Gudmundsson and Rasmus Pagh. Range-efficient consistent sampling and locality-sensitive hashing for polygons. In *ISAAC*, 2017.

14 Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of computing*, 8(1):321–350, 2012.

15 Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, pages 604–613. ACM, 1998.

16 Yehezkel Lamdan and Haim J Wolfson. Geometric hashing: A general and efficient model-based recognition scheme. In *ICCV*, page 238–249, 1988.

17 Lambert Schomaker, Edward de Leau, and Louis Vuurpijl. Using pen-based outlines for object-based annotation and image-based queries. In *AVI*, pages 585–592. Springer, 1999.

18 Shinji Umeyama. Parameterized point pattern matching and its application to recognition of object families. *TPAMI*, 15(2):136–144, 1993.

19 Remco C Veltkamp and Michiel Hagedoorn. State of the art in shape matching. In *Principles of visual information retrieval*, pages 87–119. Springer, 2001.

20 Charles T Zahn and Ralph Z Roskies. Fourier descriptors for plane closed curves. *TOC*, 100(3):269–281, 1972.

Binary Matrix Completion Under Diameter Constraints

Tomohiro Koana
Algorithmics and Computational Complexity, Faculty IV, Technische Universität Berlin, Germany

Vincent Froese
Algorithmics and Computational Complexity, Faculty IV, Technische Universität Berlin, Germany

Rolf Niedermeier
Algorithmics and Computational Complexity, Faculty IV, Technische Universität Berlin, Germany

Abstract

We thoroughly study a novel but basic combinatorial matrix completion problem: Given a binary incomplete matrix, fill in the missing entries so that the resulting matrix has a specified maximum diameter (that is, upper-bounding the maximum Hamming distance between any two rows of the completed matrix) as well as a specified minimum Hamming distance between any two of the matrix rows. This scenario is closely related to consensus string problems as well as to recently studied clustering problems on incomplete data.

We obtain an almost complete picture concerning the complexity landscape (P vs NP) regarding the diameter constraints and regarding the number of missing entries per row of the incomplete matrix. We develop polynomial-time algorithms for maximum diameter three, which are based on Deza's theorem [Discret. Math. 1973, J. Comb. Theory, Ser. B 1974] from extremal set theory. In this way, we also provide one of the rare links between sunflower techniques and stringology. On the negative side, we prove NP-hardness for diameter at least four. For the number of missing entries per row, we show polynomial-time solvability when there is only one missing entry and NP-hardness when there can be at least two missing entries. In general, our algorithms heavily rely on Deza's theorem and the correspondingly identified sunflower structures pave the way towards solutions based on computing graph factors and solving 2-SAT instances.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Discrete mathematics

Keywords and phrases sunflowers, binary matrices, Hamming distance, stringology, consensus problems, complexity dichotomy, combinatorial algorithms, graph factors, 2-Sat, Hamming radius

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.47

Related Version *Full Version*: `https://arxiv.org/abs/2002.05068`

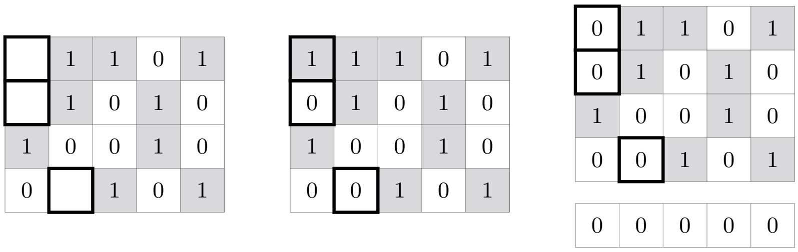
Funding *Tomohiro Koana*: Partially supported by the DFG project FPTinP (NI 369/16).

Acknowledgements We are grateful to Christian Komusiewicz for helpful feedback on an earlier version of this work and to Stefan Szeider for pointing us to the work on clustering incomplete data [9]. We also thank Curtis Bright for mentioning the connection to the Hadamard matrix completion problem.

1 Introduction

In combinatorial matrix completion problems, given an incomplete matrix over a fixed alphabet with some missing entries, the goal is to fill in the missing entries such that the resulting "completed matrix" (over the same alphabet) fulfills a desired property. Performing a parameterized complexity analysis, Ganian et al. [14, 13] and Eiben et al. [9] recently contributed to this growing field by studying various desirable properties. More specifically, Ganian et al. [14] studied the two properties of minimizing the rank or of minimizing the

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 47; pp. 47:1–47:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



$$\begin{array}{ccccc} \boxed{} & 1 & 1 & 0 & 1 \\ \boxed{} & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & \boxed{} & 1 & 0 & 1 \end{array} \qquad \begin{array}{ccccc} \boxed{1} & 1 & 1 & 0 & 1 \\ \boxed{0} & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & \boxed{0} & 1 & 0 & 1 \end{array} \qquad \begin{array}{ccccc} \boxed{0} & 1 & 1 & 0 & 1 \\ \boxed{0} & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & \boxed{0} & 1 & 0 & 1 \\ \\ 0 & 0 & 0 & 0 & 0 \end{array}$$

Figure 1 An illustration of matrix completion problems with the input matrix (left). Missing entries (and their completions) are framed by thick lines. The middle matrix is a completion of diameter four and the right matrix is a completion of radius three with the center vector below. Note that missing entries in the same column might be filled with different values to meet the diameter constraint, whereas this is never necessary for the radius constraint.

number of distinct rows of the completed matrix. Ganian et al. [13] analyzed the complexity of completing an incomplete matrix so that it fulfills certain constraints and can be partitioned into subspaces of small rank. Eiben et al. [9] investigated clustering problems where one wants to partition the rows of the completed matrix into a given number of clusters of small radius or of small diameter. Finally, Koana et al. [20] studied two cases of completing the matrix into one which has small (local) radius. The latter two papers [9, 20] rely on Hamming distance as a distance measure; in general, all considered matrix completion problems are NP-hard and thus the above papers [9, 14, 13, 20] mostly focused on parameterized complexity studies. In this work, we focus on a desirable property closely related to small radius, namely diameter bounds. Doing so, we further focus on the case of binary alphabet. For a matrix $\mathbf{T} \in \{0,1\}^{n \times \ell}$, let $\gamma(\mathbf{T}) := \min_{i \neq i' \in [n]} d(\mathbf{T}[i], \mathbf{T}[i'])$ and $\delta(\mathbf{T}) := \max_{i \neq i' \in [n]} d(\mathbf{T}[i], \mathbf{T}[i'])$, where d denotes the Hamming distance and $\mathbf{T}[i]$ denotes the i-th row of $\mathbf{T}$. We use the special symbol $\square$ to represent a missing entry. Specifically, we study the following problem.

Diameter Matrix Completion (DMC)
Input: An incomplete matrix $\mathbf{S} \in \{0,1,\square\}^{n \times \ell}$ and $\alpha \leq \beta \in \mathbb{N}$.
Question: Is there a completion $\mathbf{T} \in \{0,1\}^{n \times \ell}$ of $\mathbf{S}$ with $\alpha \leq \gamma(\mathbf{T})$ and $\delta(\mathbf{T}) \leq \beta$?

Before motivating the study of DMC, we refer to the example in Figure 1 that also illustrates significant differences between radius minimization [20] and diameter minimization (the latter referring to $\delta(\mathbf{T}) \leq \beta$ above).

Compare DMC with Constraint Radius Matrix Completion as studied by Koana et al. [20]:

Constraint Radius Matrix Completion (ConRMC)
Input: An incomplete matrix $\mathbf{S} \in \{0,1,\square\}^{n \times \ell}$ and $r \in \mathbb{N}^n$.
Question: Is there a completion $\mathbf{T} \in \{0,1\}^{n \times \ell}$ of $\mathbf{S}$ and a row vector $v \in \{0,1\}^\ell$ such that $d(v, \mathbf{T}[i]) \leq r[i]$ for all $i \in [n]$?

An important difference between DMC and ConRMC is that in DMC we basically have to compare all rows against each other, but in ConRMC we have to compare one "center row" against all others. Indeed, this makes these two similarly defined problems quite different in many computational complexity aspects.

Now, let us consider potential application scenarios where DMC may be relevant. It is a natural combinatorial matrix problems which may appear in the following contexts:

- In coding theory, one may want to "design" (by filling in the missing entries) codewords that are pairwise neither too close (parameter α in DMC) nor too far (parameter β in DMC) from each other. One prime example is the completion into a Hadamard matrix [18]. This is a special case of DMC with $n = \ell$ and $\alpha = \beta = n/2$.
- In computational biology, one may want to minimize the maximum distance of sequences in order to determine their degree of relatedness (thus minimizing β); missing entries refer to missing data points.[1]
- In data science, each row may represent an entity with its attributes, and solving the DMC problem may fulfill some constraints with respect to the pairwise (dis)similarity of the completed entities.
- In stringology, DMC seems to constitute a new and natural problem, closely related to several intensively studied consensus problems (many of which are NP-hard for binary alphabets) [1, 4, 5, 6, 16, 17, 21, 23].

Somewhat surprisingly, although simple to define and well-motivated, in the literature there seems to be no systematic study of DMC and its computational complexity. The two closest studies are the work of Eiben et al. [9] and Koana et al. [20]. Eiben et al. [9] focus on clustering while we focus on only finding one cluster (that is, the whole resulting matrix with small diameter). Another crucial difference from the work of Eiben et al. [9] is that we also model the aspect of achieving a minimum pairwise distance (not only a maximum diameter); actually, one may say that we essentially combine their "dispersion" and diameter clustering problems (for the special case of a single cluster). In this sense the problems are incomparable.

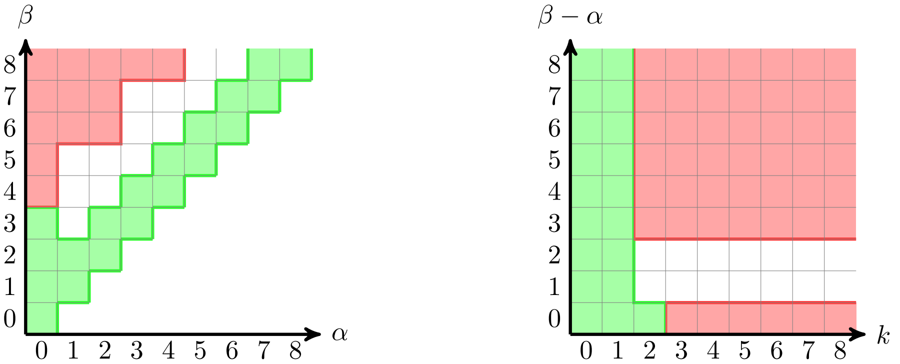
We perform a more fine-grained complexity study in terms of diameter bounds α, β and the maximum number k of missing entries in any row. Note that in bioinformatics applications matrix rows may represent sequences with few corrupted data points, thus resulting in small values for k. In fact, computational complexity with respect to this kind of parameters has been studied in the context of computational biology [1, 5, 17]. We identify polynomial-time cases as well as NP-hard cases, taking significant steps towards a computational complexity dichotomy (polynomial-time solvable versus NP-hard), leaving fairly few cases open. While the focus of the previous works [9, 20] is on parameterized complexity studies, in this work we settle more basic algorithmic questions on the DMC problem, relying on several combinatorial insights, including results from (extremal) combinatorics (most prominently, Deza's theorem [8]). Indeed, we believe that exploiting sunflowers based on Deza's theorem in combination with corresponding use of algorithms for 2-SAT and graph factors is our most interesting technical contribution. In this context, we also observe the phenomenon that the running time bounds that we can prove for odd values of α (the "lower bound for dissimilarity") are significantly better than the ones for even values of α – indeed, for even values of α the running time exponentially depends on α while it is independent of α for odd values of α. We survey our results in Figure 2 which also depicts remaining open cases.

2 Preliminaries

For $m \leq n \in \mathbb{N}$, let $[m, n] := \{m, \ldots, n\}$ and let $[n] := [1, n]$.

For a matrix $\mathbf{T} \in \{0, 1\}^{n \times \ell}$, we denote by $\mathbf{T}[i, j]$ the entry in the i-th row and j-th column ($i \in [n]$ and $j \in [\ell]$) of $\mathbf{T}$. We use $\mathbf{T}[i, :]$ (or $\mathbf{T}[i]$ in short) to denote the *row vector* $(\mathbf{T}[i, 1], \ldots, \mathbf{T}[i, \ell])$ and $\mathbf{T}[:, j]$ to denote the *column vector* $(\mathbf{T}[1, j], \ldots, \mathbf{T}[n, j])^T$. For subsets

[1] Here, it would be particularly natural to also study the case of non-binary alphabets; however, most of our positive results probably only hold for binary alphabet.



(a) Complexity of DMC with respect to combinations of constant values for α and β.

(b) Complexity of DMC with respect to combinations of the maximum number k of missing entries in any row and $\beta - \alpha$.

Figure 2 Overview of our results. Green denotes polynomial-time solvability and red denotes NP-hardness. White cells indicate open cases.

$I \subseteq [n]$ and $J \subseteq [\ell]$, we write $\mathbf{T}[I,J]$ to denote the submatrix containing only the rows in I and the columns in J. We abbreviate $\mathbf{T}[I,[\ell]]$ and $\mathbf{T}[[n],J]$ as $\mathbf{T}[I,:]$ (or $\mathbf{T}[I]$ for short) and $\mathbf{T}[:,J]$, respectively. We use the special character $\square$ for a *missing* entry. A matrix $\mathbf{S} \in \{0,1,\square\}^{n\times\ell}$ is called *incomplete* if it contains a missing entry, and it is called *complete* otherwise. We say that $\mathbf{T} \in \{0,1\}^{n\times\ell}$ is a *completion* of $\mathbf{S} \in \{0,1,\square\}^{n\times\ell}$ if either $\mathbf{S}[i,j] = \square$ or $\mathbf{S}[i,j] = \mathbf{T}[i,j]$ holds for all $i \in [n]$ and $j \in [\ell]$.

Let $u, w \in \{0,1,\square\}^\ell$ be row vectors. Let $D(u,w) := \{j \in [\ell] \mid u[j] \neq w[j] \wedge u[j] \neq \square \wedge w[j] \neq \square\}$ be the set of column indices where u and v disagree (not considering positions with missing entries). The *Hamming distance* between u and w is $d(u,w) := |D(u,w)|$. Note that the Hamming distance obeys the triangle inequality $d(u,w) \leq d(u,v) + d(v,w)$ for a complete vector $v \in \{0,1\}^\ell$. For a subset $J \subseteq [\ell]$, we also define $d_J(u,w) := d(u[J],w[J])$. Let $u', v', w' \in \{0,1\}^\ell$ be complete row vectors. Then, it holds that $d(u',w') = |D(u',v') \triangle D(v',w')| = |D(u',v')| + |D(v',w')| - 2|D(u',v') \cap D(v',w')|$. The binary operation $u \oplus v$ replaces the missing entries of u with the corresponding entries in v for $v \in \{0,1\}^\ell$. We sometimes use string notation to represent row vectors, such as 001 for $(0,0,1)$.

3 Constant Diameter Bounds α and β

In this section we consider the special case (α,β)-DMC of DMC, where $\alpha \leq \beta$ are some fixed constants. We prove the results depicted in Figure 2a. To start with, we show the following simple linear-time special case which will subsequently be used several times.

▶ **Lemma 1.** DMC *can be solved in linear time for a constant number ℓ of columns.*

Proof. If $\alpha > 0$ and $n > 2^\ell$, then there is no completion $\mathbf{T}$ of $\mathbf{S}$ with $\gamma(\mathbf{T}) \geq \alpha > 0$. Thus, we can assume that the input matrix comprises of at most $n\ell \leq 2^\ell \cdot \ell$ (that is, constantly many) entries for the case $\alpha > 0$. Suppose that $\alpha = 0$. Consider a set $\mathcal{V} \subseteq \{0,1\}^\ell$ in which the pairwise Hamming distances are at most β. We simply check whether each row vector in the input matrix can be completed to some row vector in $\mathcal{V}$ in $O(n \cdot 2^\ell) = O(n)$ time. Since there are at most 2^{2^ℓ} choices for $\mathcal{V}$, this procedure can be done in linear time. ◀

3.1 Polynomial time for $\alpha = 0$ and $\beta \leq 3$

As an entry point, we show that $(0,1)$-DMC is easily solvable. To this end, we call a column vector *dirty* if it contains both 0 and 1. Clearly, for $\alpha = 0$, we can ignore columns that are not dirty since they can always be completed without increasing the Hamming distances between rows. Hence, throughout this subsection, we assume that the input matrix contains only dirty columns. Now, any $(0,1)$-DMC instance is a **Yes**-instance if and only if there is at most one dirty column in the input matrix:

▶ **Lemma 2.** *A matrix* $\mathbf{S} \in \{0,1,\square\}^{n \times \ell}$ *admits a completion* $\mathbf{T} \in \{0,1\}^{n \times \ell}$ *with* $\delta(\mathbf{T}) \leq 1$ *if and only if* $\mathbf{S}$ *contains at most one dirty column.*

Proof. Suppose that $\mathbf{S}$ contains two dirty columns $\mathbf{S}[:, j_0]$ and $\mathbf{S}[:, j_1]$ for $j_0 \neq j_1 \in [\ell]$. We claim that $\delta(\mathbf{T}) \geq 2$ holds for any completion $\mathbf{T}$ of $\mathbf{S}$. Let $i \in [n]$. Then, there exist $i_0, i_1 \in [n]$ with $\mathbf{T}[i, j_0] \neq \mathbf{T}[i_0, j_0]$ and $\mathbf{T}[i, j_1] \neq \mathbf{T}[i_1, j_1]$. If $\delta(\mathbf{T}) \leq 1$, then we obtain $\mathbf{T}[i_0, j_1] = \mathbf{T}[i, j_1]$ and $\mathbf{T}[i_1, j_0] = \mathbf{T}[i, j_0]$. Now we have $d(\mathbf{T}[i_0], \mathbf{T}[i_1]) \geq 2$ because $\mathbf{T}[i_0, j_0] \neq \mathbf{T}[i_1, j_0]$ and $\mathbf{T}[i_0, j_1] \neq \mathbf{T}[i_1, j_1]$. The reverse direction follows easily. ◀

Lemma 2 implies that one can solve $(0,1)$-DMC in linear time. In the following, we extend this to a linear-time algorithm for $(0,2)$-DMC (Theorem 12) and a polynomial-time algorithm for $(0,3)$-DMC (Theorem 13).

For these algorithms, we make use of a concept from extremal set theory, known as Δ-systems [19]. We therefore consider matrices as certain set systems.

▶ **Definition 3.** *For a matrix* $\mathbf{T} \in \{0,1\}^{n \times \ell}$*, let* $\mathcal{T}$ *denote the set system* $\{D(\mathbf{T}[i], \mathbf{T}[n]) \mid i \in [n-1]\}$*. Moreover, for* $x \in \mathbb{N}$*, let* $\mathcal{T}_x$ *denote the set system* $\{D(\mathbf{T}[i], \mathbf{T}[n]) \mid i \in [n-1], d(\mathbf{T}[i], \mathbf{T}[n]) = x\}$*.*

The set system $\mathcal{T}$ contains the subsets (without duplicates) of column indices corresponding to the columns where the row vectors $\mathbf{T}[1], \ldots, \mathbf{T}[n-1]$ differ from $\mathbf{T}[n]$. For given $\mathbf{T}[n]$, all the rows of $\mathbf{T}$ can be determined from $\mathcal{T}$, as we have binary alphabet.

The concept of Δ-systems has previously been used to obtain efficient algorithms [9, 10, 11]. They are defined as follows:

▶ **Definition 4** (Weak Δ-system)**.** *A set family* $\mathcal{F} = \{S_1, \ldots, S_m\}$ *is a* weak Δ-system *if there exists an integer* $\lambda \in \mathbb{N}$ *such that* $|S_i \cap S_j| = \lambda$ *for any pair of distinct sets* $S_i, S_j \in \mathcal{F}$*. The integer* λ *is called the* intersection size *of* $\mathcal{F}$*.*

▶ **Definition 5** (Strong Δ-system, Sunflower)**.** *A set family* $\mathcal{F} = \{S_1, \ldots, S_m\}$ *is a* strong Δ-system *(or* sunflower*) if there exists a subset* $C \subsetneq S_1 \cup \cdots \cup S_m$ *such that* $S_i \cap S_j = C$ *for any pair of distinct sets* $S_i, S_j \in \mathcal{F}$*. We call the set* C *the* core *and the sets* $P_i = S_i \setminus C$ *the* petals *of* $\mathcal{F}$*.*

Clearly, every strong Δ-system is a weak Δ-system.

Our algorithms employ the combinatorial property that under certain conditions the set system $\mathcal{T}$ of a matrix $\mathbf{T}$ with bounded diameter forms a strong Δ-system (which can be algorithmically exploited). We say that a family $\mathcal{F}$ of sets is h-uniform if $|S| = h$ holds for each $S \in \mathcal{F}$. Deza [8] showed that an h-uniform weak Δ-system is a strong Δ-system if its cardinality is sufficiently large (more precisely, if $|\mathcal{F}| \geq h^2 - h + 2$). Moreover, Deza [7] also proved a stronger lower bound for uniform weak Δ-systems in which the intersection size is exactly half of the cardinality of each set.

▶ **Lemma 6** ([7, Théorème 1.1])**.** *Let* $\mathcal{F}$ *be a* (2μ)*-uniform weak* Δ*-system with intersection size* μ*. If* $|\mathcal{F}| \geq \mu^2 + \mu + 2$*, then* $\mathcal{F}$ *is a strong* Δ*-system.*

We extend this result to the case in which the set size is odd in the full version.

▶ **Lemma 7.** *Let $\mathcal{F}$ be a $(2\mu+1)$-uniform weak Δ-system.*

(i) *If the intersection size of $\mathcal{F}$ is $\mu+1$ and $|\mathcal{F}| \geq \mu^2+\mu+3$, then $\mathcal{F}$ is a strong Δ-system.*

(ii) *If the intersection size of $\mathcal{F}$ is μ and $|\mathcal{F}| \geq (\mu+1)^2+\mu+3$, then $\mathcal{F}$ is a strong Δ-system.*

In order to obtain a linear-time algorithm for $(0,2)$-DMC, we will prove that for $\mathbf{T} \in \{0,1\}^{n\times\ell}$ with $\delta(\mathbf{T}) \leq 2$ and sufficiently large ℓ, the set system $\mathcal{T}$ is a sunflower. This yields a linear-time algorithm via a reduction to a linear-time solvable special case of ConRMC. We start with a simple observation on matrices of diameter two, which will be helpful in the subsequent proofs.

▶ **Observation 8.** *Let $\mathbf{T} \in \{0,1\}^{n\times\ell}$ be a matrix with $\delta(\mathbf{T}) \leq 2$. For each $T_1 \in \mathcal{T}_1$ and $T_2, T_2' \in \mathcal{T}_2$, it holds that $T_1 \subseteq T_2$ and that $|T_2 \cap T_2'| \geq 1$ (otherwise there exists a pair of rows with Hamming distance three).*

The next lemma states that $|\mathcal{T}_2|$ restricts the number of columns.

▶ **Lemma 9.** *Let $\mathbf{T} \in \{0,1\}^{n\times\ell}$ be a matrix consisting of only dirty columns with $\delta(\mathbf{T}) \leq 2$. If $\mathcal{T}_2 \neq \emptyset$, then $\ell \leq |\mathcal{T}_2|+1$.*

Proof. First, observe that $\ell = |\bigcup_{T_1\in\mathcal{T}_1} T_1 \cup \bigcup_{T_2\in\mathcal{T}_2} T_2|$ because each column of $\mathbf{T}$ is dirty. Thus, it follows from Observation 8 that $\ell = |\bigcup_{T_2\in\mathcal{T}_2} T_2|$. We prove the lemma by induction on $|\mathcal{T}_2|$. Clearly, we have at most two columns if $|\mathcal{T}_2| = 1$. Suppose that $|\mathcal{T}_2| \geq 2$. For $T_2 \in \mathcal{T}_2$, we claim that

$$\ell = \left|\bigcup_{T_2'\in\mathcal{T}_2} T_2'\right| = \left|\bigcup_{T_2'\in\mathcal{T}_2\setminus\{T_2\}} T_2'\right| + \left|T_2 \setminus \bigcup_{T_2'\in\mathcal{T}_2\setminus\{T_2\}} T_2'\right| \leq |\mathcal{T}_2|+1.$$

The induction hypothesis gives us that $|\bigcup_{T_2'\in\mathcal{T}_2\setminus\{T_2\}} T_2'| \leq |\mathcal{T}_2|$. For the other term, observe that $|T_2 \setminus \bigcup_{T_2'\in\mathcal{T}_2\setminus\{T_2\}} T_2'| \leq |T_2 \setminus T_2''| = |T_2| - |T_2 \cap T_2''|$ for $T_2'' \in \mathcal{T}_2 \setminus \{T_2\}$. Hence, it follows from Observation 8 that the second term is at most 1. ◀

Next, we show that a matrix with diameter at most two has radius at most one as long as it has at least five columns. Thus, we can solve DMC by solving ConRMC with radius one, which can be done in linear time via a reduction to 2-SAT [20]. We use the following lemma concerning certain intersections of a set with elements of a sunflower.

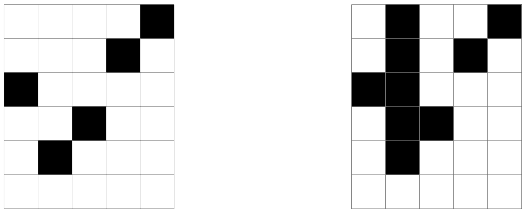
▶ **Lemma 10** ([11, Lemma 8]). *Let $\lambda \in \mathbb{N}$, let $\mathcal{F}$ be a sunflower with core C, and let X be a set such that $|X \cap S| \geq \lambda$ for all $S \in \mathcal{F}$. If $|\mathcal{F}| > |X|$, then $\lambda \leq |C|$ and $|X \cap C| \geq \lambda$.*

▶ **Lemma 11.** *Let $\mathbf{T} \in \{0,1\}^{n\times\ell}$ be a matrix with $\delta(\mathbf{T}) \leq 2$. If $\ell \geq 5$, then there exists a vector $v \in \{0,1\}^\ell$ such that $d(v, \mathbf{T}[i]) \leq 1$ for all $i \in [n]$.*

Proof. If $\mathcal{T}_2 = \emptyset$, then we are immediately done by definition, because $d(\mathbf{T}[n], \mathbf{T}[i]) \leq 1$ for all $i \in [n]$ (see Figure 3a for an illustration). Since $\ell \geq 5$, Lemma 9 implies $|\mathcal{T}_2| \geq 4$.

It follows from Observation 8 that $\mathcal{T}_2$ is a 2-uniform weak Δ-system with intersection size one (see Figure 3b). Thus, $\mathcal{T}_2$ is a sunflower by Lemma 6. Let $\{j_{\text{core}}\}$ denote the core of $\mathcal{T}_2$. Note that $|T_1 \cap T_2| \geq 1$ holds for each $T_1 \in \mathcal{T}_1$ and $T_2 \in \mathcal{T}_2$ by Observation 8. Now we can infer from Lemma 10 (let $X = T_1$, $\lambda = 1$, and $\mathcal{F} = \mathcal{T}_2$) that $\mathcal{T} \subseteq \{T_1\}$, where $T_1 = \{j_{\text{core}}\}$.

Hence, it holds that $d(v, \mathbf{T}[i]) \leq 1$ for all $i \in [n]$, where $v \in \{0,1\}^\ell$ is a row vector such that $v[j_{\text{core}}] = 1 - \mathbf{T}[n, j_{\text{core}}]$ and $v[j] = \mathbf{T}[n,j]$ for each $j \in [\ell] \setminus \{j_{\text{core}}\}$. ◀



(a) The case $\mathcal{T}_2 = \emptyset$. **(b)** The case $|\mathcal{T}_2| \geq 4$.

Figure 3 Illustration of Lemma 11 with $n = 6$. A black cell denotes a value different from row $\mathbf{T}[6]$. In (b) the set system $\mathcal{T}_2$ forms a sunflower with core $\{2\}$. In both cases the radius is one.

▶ **Theorem 12.** $(0, 2)$-DMC *can be solved in* $O(n\ell)$ *time.*

Proof. Let $\mathbf{S} \in \{0, 1, \square\}^{n\times\ell}$ be an input matrix of $(0, 2)$-DMC. If $\ell \leq 4$, then we use the linear-time algorithm of Lemma 1. Henceforth, we assume that $\ell \geq 5$.

We claim that $\mathbf{S}$ is a **Yes**-instance if and only if the ConRMC instance $I = (\mathbf{S}, 1^n)$ is a **Yes**-instance.

($\Rightarrow$) Let $\mathbf{T}$ be a completion of $\mathbf{S}$ with $\delta(\mathbf{T}) \leq 2$. Since $\ell \geq 5$, there exists a vector v such that $d(v, \mathbf{T}[i]) \leq 1$ for all $i \in [n]$ by Lemma 11. It follows that I is a **Yes**-instance.

($\Leftarrow$) Let v be a solution of I. Let $\mathbf{T}$ be the matrix such that for each $i \in [n]$, $\mathbf{T}[i] = \mathbf{S}[i] \oplus v$ (recall that $u \oplus v$ denotes the vector obtained from u by replacing all missing entries of u with the entries of v in the corresponding positions). Then, we have $d(v, \mathbf{T}[i]) \leq 1$ for each $i \in [n]$. By the triangle inequality, we obtain $d(\mathbf{T}[i], \mathbf{T}[i']) \leq d(v, \mathbf{T}[i]) + d(v, \mathbf{T}[i']) \leq 2$ for each $i, i' \in [n]$.

Since ConRMC can be solved in linear time when $\max_{i\in[n]} r[i] = 1$ [20, Theorem 1], it follows that $(0, 2)$-DMC can be solved in linear time. ◀

In the full version, we show polynomial-time solvability of $(0, 3)$-DMC. The overall idea is, albeit technically more involved, similar to $(0, 2)$-DMC. We first show that the set family $\mathcal{T}$ of a matrix $\mathbf{T}$ with $\delta(\mathbf{T}) = 3$ contains a sunflower by Lemma 7. We then show that such a matrix has a certain structure which again allows us to reduce the problem to the linear-time solvable special case of ConRMC with radius one.

▶ **Theorem 13.** $(0, 3)$-DMC *can be solved in* $O(n\ell^4)$ *time.*

Our algorithms work via reductions to ConRMC. Although ConRMC imposes an upper bound on the diameter implicitly by the triangle inequality, it is seemingly difficult to enforce any lower bounds (that is, $\alpha > 0$). In the next subsection, we will see polynomial-time algorithms for $\alpha > 0$, based on reductions to the graph factorization problem.

3.2 Polynomial time for $\beta = \alpha + 1$

We now give polynomial-time algorithms for (α, β)-DMC with constant $\alpha > 0$ given that $\beta \leq \alpha + 1$. As in Section 3.1, our algorithms exploit combinatorial structures revealed by Deza's theorem (Lemmas 6 and 7). Recall that $\mathcal{T}$ denotes a set system obtained from a complete matrix $\mathbf{T}$ (Definition 3). We show that $\mathcal{T}$ essentially is a sunflower when $\gamma(\mathbf{T}) \geq \alpha$ and $\delta(\mathbf{T}) \leq \alpha + 1$. For the completion into such a sunflower, it suffices to solve the following matrix completion problem, which we call Sunflower Matrix Completion.

Sunflower Matrix Completion (SMC)

Input: An incomplete matrix $\mathbf{S} \in \{0,1,\square\}^{n\times\ell}$ and $s, m \in \mathbb{N}$.

Question: Is there a completion $\mathbf{T} \in \{0,1\}^{n\times\ell}$ of $\mathbf{S}$ such that $D(\mathbf{T}[1],\mathbf{T}[n]),\dots,D(\mathbf{T}[n-1],\mathbf{T}[n])$ are pairwise disjoint sets each of size at most s and $\sum_{i\in[n-1]}|D(\mathbf{T}[i],\mathbf{T}[n])| \geq m$.

Intuitively speaking, the problem asks for a completion into a sunflower with empty core and bounded petal sizes. All algorithms presented in this subsection are via reductions to SMC. First, we show that SMC is indeed polynomial-time solvable. We prove this using a well-known polynomial-time algorithm for the graph problem (g,f)-Factor [12].

(g,f)-Factor

Input: A graph $G=(V,E)$, functions $f,g\colon V\to\mathbb{N}$, and $m'\in\mathbb{N}$.

Question: Does G contain a subgraph $G'=(V,E')$ such that $|E'|\geq m'$ and $g(v)\leq \deg_{G'}(v)\leq f(v)$ for all $v\in V$?

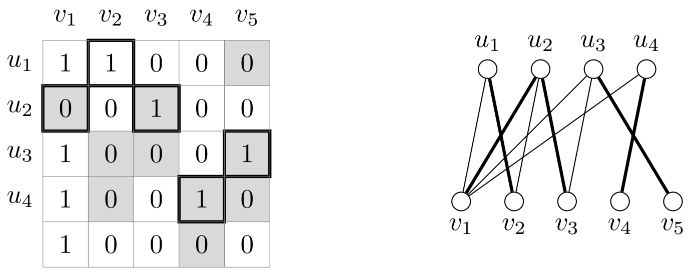
▶ **Lemma 14.** *For constant $s>0$,* SMC *can be solved in $O(n\ell\sqrt{n+\ell})$ time.*

Proof. Let $(\mathbf{S},s,m)$ be an SMC instance. Let a_j^x be the number of occurrences of $x\in\{0,1\}$ in $\mathbf{S}[:,j]$ for each $j\in[\ell]$. We can assume that $a_j^0\geq a_j^1$ for each $j\in[\ell]$ (otherwise swap the occurrences of 0's and 1's in the column). If $a_j^0\geq 2$ and $\mathbf{S}[n,j]=1$ for some $j\in[\ell]$, then we can return **No** since there will be two intersecting sets. Also, if $a_j^1\geq 2$, then we return **No**.

We construct an instance of (g,f)-Factor as follows. We introduce a vertex u_i for each $i\in[n-1]$ and a vertex v_j for each $j\in[\ell]$. The resulting graph G will be a bipartite graph with one vertex subset $\{u_1,\dots,u_{n-1}\}$ representing rows and the other $\{v_1,\dots,v_\ell\}$ representing columns. Essentially, we add an edge between u_i and v_j if the column $\mathbf{S}[:,j]$ can be completed such that the i-th entry differs from all other entries on $\mathbf{S}[:,j]$ (see Figure 4 for an illustration). Intuitively, such an edge encodes the information that column index j can be contained in a petal of the sought sunflower. Formally, there is an edge $\{u_i,v_j\}$ if and only if there is a completion $t_j\in\{0,1\}^n$ of $\mathbf{S}[:,j]$ in which $t_j[i]=1-t_j[n]$ and $t_j[h]=t_j[n]$ for all $h\in[n-1]\setminus\{i\}$. We set $g(u_i):=0$ and $f(u_i):=s$ for each $i\in[n-1]$, $g(v_j):=a_j^1$ and $f(v_j):=1$ for each $j\in[\ell]$, and $m':=m$. This construction can be done in $O(n\ell)$ time. To see this, note that the existence of an edge $\{u_i,v_j\}$ only depends on a_j^0, a_j^1, and $\mathbf{S}[i,j]$.

- If $a_j^0\leq 1$ and $a_j^1=0$, then add the edge $\{u_i,v_j\}$. The corresponding completion t_j can be seen as follows:
  - If $\mathbf{S}[h,j]=\square$ for all $h\in[n-1]$, then let $t_j[i]:=1$ and let $t_j[h]:=0$ for all $h\in[n]\setminus\{i\}$.
  - If $\mathbf{S}'[h,j]=0$ for some $h\in[n-1]$, then $\mathbf{S}'[h',j]=\square$ for all $h'\in[n]\setminus\{h\}$. If $h\neq i$, then let $t_j[i]:=1$ and let $t_j[h]:=0$ for all $h\in[n]\setminus\{i\}$. Otherwise, let $t_j[h]:=1$ for all $h\in[n]\setminus\{i\}$.
- If $a_j^0=1$ and $a_j^1=1$, then add the edge $\{u_i,v_j\}$ if $\mathbf{S}[i,j]\neq\square$.
- If $a_j^0\geq 2$ and $a_j^1=0$, then add the edge $\{u_i,v_j\}$ if $\mathbf{S}[i,j]=\square$.
- If $a_j^0\geq 2$ and $a_j^1=1$, then add the edge $\{u_i,v_j\}$ if $\mathbf{S}[i,j]=1$ (because $\mathbf{S}[n,j]$ must be completed with 0).

The correctness of the reduction easily follows from the definition of an edge: If $\mathbf{T}$ is a solution for $(\mathbf{S},s,m)$, then the corresponding subgraph of G contains the edge $\{u_i,v_j\}$ for each $i\in[n-1]$ and each $j\in D(\mathbf{T}[i],\mathbf{T}[n])$. Conversely, a completion of $\mathbf{S}$ is obtained from a subgraph G' by taking for each edge $\{u_i,v_j\}$ the corresponding completion t_j as the j-th column. Note that no vertex v_j can have two incident edges since $f(v_j)=1$. Moreover, if v_j has no incident edges, then this implies that $g(v_j)=a_j^1=0$. Hence, we can complete all missing entries in column j by 0.



	v_1	v_2	v_3	v_4	v_5
u_1	1	1	0	0	0
u_2	0	0	1	0	0
u_3	1	0	0	0	1
u_4	1	0	0	1	0
	1	0	0	0	0

Figure 4 A completion of a 5×5 incomplete matrix (left). The known entries are highlighted in gray. A bipartite graph as constructed in the reduction (right). Note that the entries framed by thick lines (which differ from all others in the same column) correspond to the subgraph represented by the thick lines.

Regarding the running time, note that the constructed graph G has at most $n\ell$ edges and $\sum_{i\in[n-1]} f(u_i) \in O(n)$ and $\sum_{j\in[\ell]} f(v_j) \in O(\ell)$. Since (g,f)-Factor can be solved in $O(|E|\sqrt{f(V)})$ time [12] for $f(V) = \sum_{v\in V} f(v)$, SMC can be solved in $O(n\ell\sqrt{n+\ell})$ time. ◀

Using Lemma 14, we first show that (α,α)-DMC can be solved in polynomial time.

▶ **Theorem 15.** (α,α)-DMC *can be solved in* $O(n\ell\sqrt{n+\ell})$ *time.*

Proof. We first show that (α,α)-DMC can easily be solved if α is odd. Consider row vectors $u,v,w \in \{0,1\}^\ell$ and let $U := D(u,v)$ and $W := D(v,w)$. Then, $d(u,v)+d(v,w)+d(w,u) = |U|+|W|+(|U|+|W|-2|U\cap W|) = 2(|U|+|W|-|U\cap W|)$ and hence $d(u,v)+d(v,w)+d(w,u)$ is even. Thus, we can immediately answer **No** if $n \geq 3$. It is also easy to see that DMC can be solved in linear time if $n \leq 2$.

We henceforth assume that α is even. Eiben et al. [9, Theorem 34] provided a linear-time algorithm for $(0,\alpha)$-DMC with constant n (and arbitrary α) using reductions to integer linear programming (ILP). It is straightforward to adapt their ILP formulation to show that (α,α)-DMC can also be solved in linear time for constant n (basically, we just need the additional constraint that each pairwise distance is at least α). So we can assume that $n \geq (\alpha/2)^2 + (\alpha/2) + 3$ (otherwise (α,α)-DMC can be solved in linear time). We claim that there is a completion $\mathbf{T}$ of $\mathbf{S}$ with $\gamma(\mathbf{T}) = \delta(\mathbf{T}) = \alpha$ if and only if the SMC instance $(\mathbf{S}',\alpha/2,\alpha n/2)$ is a **Yes**-instance for the matrix $\mathbf{S}' \in \{0,1,\square\}^{(n+1)\times\ell}$ obtained from $\mathbf{S}$ with an additional row vector $\square^\ell$.

($\Rightarrow$) Let $\mathbf{T}$ be a completion of $\mathbf{S}$ with $\gamma(\mathbf{T}) = \delta(\mathbf{T}) = \alpha$. Then, $\mathcal{T}$ is a weak Δ-system with intersection size $\alpha/2$. Since $|\mathcal{T}| \geq (\alpha/2)^2 + (\alpha/2) + 2$, Lemma 6 tells us that $\mathcal{T}$ is a sunflower. Let C be the core of $\mathcal{T}$. Consider the completion $\mathbf{T}'$ of $\mathbf{S}'$ such that

- $\mathbf{T}'[[n],:] = \mathbf{T}$,
- $\mathbf{T}'[n+1,j] = 1 - \mathbf{T}[n,j]$ for each $j \in C$, and
- $\mathbf{T}'[n+1,j] = \mathbf{T}[n,j]$ for each $j \in [\ell] \setminus C$.

Note that $D(\mathbf{T}'[i],\mathbf{T}'[n+1]) = D(\mathbf{T}'[i],\mathbf{T}'[n]) \setminus C$ for each $i \in [n-1]$. Note also that $D(\mathbf{T}'[n],\mathbf{T}'[n+1]) = C$. Hence, $D(\mathbf{T}'[1],\mathbf{T}'[n+1]),\dots,D(\mathbf{T}'[n],\mathbf{T}'[n+1])$ are pairwise disjoint sets of size $\alpha/2$.

($\Leftarrow$) Let $\mathbf{T}'$ be a completion of $\mathbf{S}'$ such that $D(\mathbf{T}'[1],\mathbf{T}'[n+1]),\dots,D(\mathbf{T}'[n],\mathbf{T}'[n+1])$ are pairwise disjoint sets of size $\alpha/2$. For the completion $\mathbf{T} = \mathbf{T}'[[n],:]$ of $\mathbf{S}$, it holds that $d(\mathbf{T}[i],\mathbf{T}[i']) = |D(\mathbf{T}'[i],\mathbf{T}'[n+1]) \triangle D(\mathbf{T}'[i'],\mathbf{T}'[n+1])| = |D(\mathbf{T}'[i],\mathbf{T}'[n+1])| + |D(\mathbf{T}'[i'],\mathbf{T}'[n+1])| = \alpha$ for each $i,i' \in [n]$. ◀

Now we proceed to develop polynomial-time algorithms for the case $\alpha + 1 = \beta$. We will make use of the following observation made by Froese et al. [11, Proof of Theorem 9].

▶ **Observation 16.** *Let* $\mathbf{T} \in \{0,1\}^{n\times\ell}$ *with* $\gamma(\mathbf{T}) \geq \alpha$ *and* $\delta(\mathbf{T}) \leq \beta = \alpha + 1$. *For* $T_\alpha \neq T'_\alpha \in \mathcal{T}_\alpha$ *and* $T_\beta \neq T'_\beta \in \mathcal{T}_\beta$, *it holds that* $|T_\alpha \cap T'_\alpha| = \lfloor \alpha/2 \rfloor$, $|T_\alpha \cap T_\beta| = \lceil \alpha/2 \rceil = \lfloor \beta/2 \rfloor$, *and* $|T_\beta \cap T'_\beta| = \lceil \beta/2 \rceil$.

Surprisingly, odd α seems to allow for significantly more efficient algorithms than even α.

▶ **Theorem 17.** (α, β)-DMC *with* $\beta = \alpha + 1$ *can be solved in*

(i) $O(n\ell\sqrt{n+\ell})$ *time for odd* α, *and*

(ii) $(n\ell)^{O(\alpha^3)}$ *time for even* α.

Proof. (i) We can assume that $n \geq \beta^2/2 + \beta + 7$ holds since otherwise the problem is linear-time solvable via a reduction to ILP as in the proof of Theorem 15. Suppose that $\mathbf{S}$ admits a completion $\mathbf{T}$ with $\gamma(\mathbf{T}) \geq \alpha$ and $\delta(\mathbf{T}) \leq \beta$. Since $\mathcal{T} = \mathcal{T}_\alpha \cup \mathcal{T}_\beta$ and $|\mathcal{T}| \geq \beta^2/2 + \beta + 6$, it follows that $\max\{|\mathcal{T}_\alpha|, |\mathcal{T}_\beta|\} \geq c := (\beta/2)^2 + (\beta/2) + 3$. We consider two cases depending on the size of $\mathcal{T}_\alpha$ and $\mathcal{T}_\beta$.

1. Suppose that $|\mathcal{T}_\alpha| \geq c$. Since $\mathcal{T}_\alpha$ is a weak Δ-system with intersection size $(\alpha-1)/2$, $\mathcal{T}_\alpha$ is a sunflower with a core of size $(\alpha-1)/2$ and petals of size $(\alpha+1)/2$ by Lemma 7 (ii). We claim that $\mathcal{T}_\beta = \emptyset$. Suppose not and let $T_\beta \in \mathcal{T}_\beta$. Consequently, we obtain $|T_\alpha \cap T_\beta| = (\alpha+1)/2$ for all $T_\alpha \in \mathcal{T}_\alpha$ by Observation 16, which contradicts Lemma 10.
2. Suppose that $|\mathcal{T}_\beta| \geq c$. Again, $\mathcal{T}_\beta$ is a sunflower whose core C has size $\beta/2$ by Lemma 6. By Observation 16 and Lemma 10, $T_\alpha \supseteq C$ holds for each $T_\alpha \in \mathcal{T}_\alpha$. Now suppose that there exist $T_\alpha \neq T'_\alpha \in \mathcal{T}_\alpha$. Since $C \subseteq T_\alpha$ and $C \subseteq T'_\alpha$, it follows that $|T_\alpha \cap T'_\alpha| \geq \beta/2$, thereby contradicting Observation 16. Hence, we have $|\mathcal{T}_\alpha| \leq 1$.

We construct an instance I of SMC covering both cases above, as in Theorem 15. We use the matrix $\mathbf{S}'$ obtained from $\mathbf{S}$ by appending a row vector $\square^\ell$, and we set $s := \beta/2$ and $m := ns - 1$. Basically, we allow at most one "petal" to have size $s - 1$. We return **Yes** if and only if I is a **Yes**-instance. The correctness can be shown analogously to the proof of Theorem 15.

(ii) Suppose that there is a completion $\mathbf{T}$ of $\mathbf{S}$ with $\gamma(\mathbf{T}) \geq \alpha$ and $\delta(\mathbf{T}) \leq \beta$. Again, we can assume that $n > 2c$ for $c := (\beta/2)^2 + (\beta/2) + 4$, and consider a case distinction regarding the size of $\mathcal{T}_\alpha$ and $\mathcal{T}_\beta$.

1. Suppose that $|\mathcal{T}_\alpha| \geq c$ and $|\mathcal{T}_\beta| \geq c$. It follows from Observation 16 and Lemmas 6 and 7 that $\mathcal{T}_\alpha$ and $\mathcal{T}_\beta$ are sunflowers. Let C_α and C_β be the cores of $\mathcal{T}_\alpha$ and $\mathcal{T}_\beta$, respectively. Note that $|C_\alpha| = \alpha/2$ and $|C_\beta| = \alpha/2 + 1$, and hence $C_\alpha \subsetneq C_\beta$ holds by Observation 16 and Lemma 10. Let $j \in [\ell]$ be such that $C_\alpha \cup \{j\} = C_\beta$ and let $\mathbf{T}' := \mathbf{T}[:, [\ell] \setminus \{j\}]$. Then, the set family $\mathcal{T}'$ is a sunflower with a core of size $\alpha/2$ and petals of size $\alpha/2$. Hence, there exists $j \in [\ell]$ such that the (α, α)-DMC instance $\mathbf{S}[:, [\ell] \setminus \{j\}]$ is a **Yes**-instance. On the other hand, if there is a completion $\mathbf{T}'$ of $\mathbf{S}[:, [\ell] \setminus \{j\}]$ with $\gamma(\mathbf{T}') = \delta(\mathbf{T}') = \alpha$, then $\gamma(\mathbf{T}) \geq \alpha$ and $\delta(\mathbf{T}) \leq \alpha + 1$ hold for any completion $\mathbf{T}$ of $\mathbf{S}$ with $\mathbf{T}[:, [\ell] \setminus \{j\}] = \mathbf{T}'$.
2. Suppose that $|\mathcal{T}_\alpha| \geq c$ and $|\mathcal{T}_\beta| < c$. The same argument as above shows that $T_\alpha \cap T_\beta = C$ holds for each $T_\alpha \in \mathcal{T}_\alpha$ and $T_\beta \in \mathcal{T}_\beta$, where C is the size-$\alpha/2$ core of sunflower $\mathcal{T}_\alpha$. Let $I_\beta = \{i \in [n-1] \mid d(\mathbf{T}[i], \mathbf{T}[n]) = \beta\}$ be the row indices that induce the sets in $\mathcal{T}_\beta$ and let $J_\beta = \bigcup_{T_\beta \in \mathcal{T}_\beta} T_\beta$. Consider $\mathbf{T}' = \mathbf{S}[[n] \setminus I_\beta, [\ell] \setminus (C \cup J_\beta)]$ and note that the family $\mathcal{T}'$ consists of pairwise disjoint sets, each of size $\alpha/2$. We use this observation to obtain a reduction to SMC. The idea is to test all possible choices for $\mathbf{T}'$, that is, we simply try out all possibilities to choose the following sets:
   - $C \subseteq [\ell]$ of size exactly $\alpha/2$.

- $I_\beta \subseteq [n-1]$ of size at most c.
- $J_\beta \subseteq [\ell] \setminus C$ of size at most $\beta \cdot c$ such that $d_{[\ell]\setminus(C\cup J_\beta)}(\mathbf{S}[i_\beta], \mathbf{S}[n]) = 0$ for all $i_\beta \in I_\beta$.

For each possible choice, we check whether it allows for a valid completion. Formally, it is necessary that the following exist:

- A completion t_C of $\mathbf{S}[n, C]$ such that $\mathbf{S}[i, j] \neq t_C[j]$ for all $i \in [n-1]$ and $j \in C$.
- A completion t_{J_β} of $\mathbf{S}[n, J_\beta]$ such that $d(t_{J_\beta}, \mathbf{S}[i, J_\beta]) = 0$ for all $i \in [n-1] \setminus I_\beta$.
- A completion t_{i_β} of $\mathbf{S}[i_\beta, J_\beta]$ for each $i_\beta \in I_\beta$ such that $d(t_{i_\beta}, t_{J_\beta}) = \alpha/2 + 1$ for each $i_\beta \in I_\beta$ and $d(t_{i_\beta}, t_{i'_\beta}) = \alpha$ for each $i_\beta \neq i'_\beta \in I_\beta$.

The existence of the above completions can be checked in $O(n)$ time. We then construct an SMC instance $(\mathbf{S}', \alpha/2, (n - |I_\beta| - 1) \cdot \alpha/2)$, where $\mathbf{S}'$ is an incomplete matrix with $n' = n - |I_\beta|$ rows and $\ell - |C| - |J_\beta|$ columns defined as follows:

- $\mathbf{S}'[[n'-1]] = \mathbf{S}[[n-1] \setminus I_\beta, [\ell] \setminus (C \cup J_\beta)]$.
- $\mathbf{S}'[n', j] = \square$ for each $j \in [\ell] \setminus (C \cup J_\beta)$ such that $\mathbf{S}[i_\beta, j] = \square$ for all $i_\beta \in I_\beta \cup \{n\}$.
- $\mathbf{S}'[n', j] = \mathbf{S}[i_\beta, j]$ for each $j \in [\ell]\setminus(C\cup J_\beta)$ such that $\mathbf{S}[i_\beta, j] \neq \square$ for some $i_\beta \in I_\beta \cup \{n\}$.

Overall, we solve at most $(n\ell)^{O(\alpha^3)}$ SMC instances and hence it requires $(n\ell)^{O(\alpha^3)}$ time.

3. Suppose that $|\mathcal{T}_\alpha| < c$ and $|\mathcal{T}_\beta| \geq c$. Let $i \in [n-1]$ be such that $d(\mathbf{T}[i], \mathbf{T}[n]) = \beta$. Then, $d(\mathbf{T}[i], \mathbf{T}[i']) = \alpha$ holds for each $i' \in [n-1] \setminus \{i\}$ with $d(\mathbf{T}[i'], \mathbf{T}[n]) = \beta$. Since there are at least $c - 1 = (\beta/2)^2 + (\beta/2) + 3$ such row indices, it follows that this case is essentially equivalent to the previous case (by considering row i as the last row). ◀

A natural question is whether one can extend our approach above to the case $\beta = \alpha + 2$ (particularly $\alpha = 1$ and $\beta = 3$). The problem is that the petals of the sunflowers $\mathcal{T}_2$ and $\mathcal{T}_3$ may have nonempty intersections. Thus, reducing to SMC to obtain a polynomial-time algorithm is probably hopeless.

3.3 NP-hardness

Hermelin and Rozenberg [17, Theorem 5] proved that ConRMC (under the name Closest String with Wildcards) is NP-hard even if $r[i] = 2$ for all $i \in [n]$. Using this result, we prove the following (the proof is in the full version).

▶ **Theorem 18.** (α, β)-DMC *is NP-hard if* $\beta \geq 2\lceil \alpha/2 \rceil + 4$.

It remains open whether NP-hardness also holds for $(\alpha, \alpha + 3)$-DMC with $\alpha \geq 1$ (recall that $(0, 3)$-DMC is polynomial-time solvable). In Section 4, however, we show NP-hardness for $\beta = \alpha + 3$ when α and β are part of the input.

4 Bounded number k of missing entries per row

In this section, we consider DMC with α and β being part of the input, hence not necessarily being constants. We consider the maximum number k of missing entries in any row as a parameter (DMC is clearly trivial for $k = 0$). We obtain two polynomial-time algorithms and two NP-hardness results. Our polynomial-time algorithms are based on reductions to 2-SAT (see the full version for the proof).

▶ **Theorem 19.** DMC *can be solved in* $O(n^2\ell)$ *time*

(i) *for* $k = 1$, *and*

(ii) *for* $k = 2$ *and* $\alpha = \beta$.

$$\mathbf{T} = \left[\begin{array}{c|c|c|c} 001 & 111 & 001 & 000000000 \\ 111 & 111 & 001 & 000000000 \\ \hline 111 & 111 & 010 & 111111111 \\ 111 & 010 & 111 & 111111111 \end{array}\right]$$

Figure 5 An illustration of the reduction from Orthogonal Vectors, where $\mathcal{U} = \{010, 110\}$ and $\mathcal{V} = \{110, 101\}$.

To complement this result, we show that the quadratic dependence on n in the running time of Theorem 19 is inevitable under Orthogonal Vectors Conjecture (OVC), which states that Orthogonal Vectors cannot be solved in $O(n^{2-\epsilon} \cdot \ell^c)$ time for any $\epsilon, c > 0$ (it is known that Strong Exponential Time Hypothesis implies OVC [24]).

Orthogonal Vectors
Input: Sets $\mathcal{U}, \mathcal{V}$ of row vectors in $\{0,1\}^\ell$ with $|\mathcal{U}| = |\mathcal{V}| = n$.
Question: Are there row vectors $u \in \mathcal{U}$ and $v \in \mathcal{V}$ such that $u[j] \cdot v[j] = 0$ holds for all $j \in [\ell]$?

▶ **Theorem 20.** DMC *cannot be solved in* $O(n^{2-\epsilon} \cdot \ell^c)$ *time for any* $c, \epsilon > 0$*, unless* OVC *breaks.*

Proof. We reduce from Orthogonal Vectors. Let $u_1, \ldots, u_n, v_1, \ldots, v_n \in \{0,1\}^\ell$ be row vectors. Consider the matrix $\mathbf{T} \in \{0,1\}^{2n \times 6\ell}$ where

$$\mathbf{T}[i, [3j-2, 3j]] = \begin{cases} 001 & \text{if } u_i[j] = 0, \\ 111 & \text{if } u_i[j] = 1, \end{cases} \qquad \mathbf{T}[i, [3\ell + 3j - 2, 3\ell + 3j]] = 000,$$

$$\mathbf{T}[n+i, [3j-2, 3j]] = \begin{cases} 010 & \text{if } v_i[j] = 0, \\ 111 & \text{if } v_i[j] = 1, \end{cases} \qquad \mathbf{T}[n+i, [3\ell + 3j - 2, 3\ell + 3j]] = 111,$$

for each $i \in [n]$ and $j \in [\ell]$ (see Figure 5 for an illustration). We show that $\delta(\mathbf{T}) = 5\ell$ if and only if there are $i, i' \in [n]$ such that u_i and $v_{i'}$ are orthogonal. By construction, we have

$$d(\mathbf{T}[i, [3j-2, 3j]], \mathbf{T}[n+i', [3j-2, 3j]]) = \begin{cases} 2 & \text{if } u_i[j] = 0 \text{ or } v_i[j] = 0, \\ 0 & \text{otherwise.} \end{cases}$$

for any $i, i' \in [n]$ and $j \in [\ell]$. Thus, it holds for any orthogonal vectors u_i and v'_i that $d(\mathbf{T}[i], \mathbf{T}[n+i']) = 5\ell$. Conversely, suppose that there exist $i < i' \in [2n]$ such that $d(\mathbf{T}[i], \mathbf{T}[i']) = 5\ell$. It is easy to see that $i \in [n]$, $i' \in [n+1, 2n]$ (otherwise $d(\mathbf{T}[i], \mathbf{T}[i']) \leq 3\ell$). Then, the vectors u_i and $v_{i'-n}$ are orthogonal. ◀

Finally, we prove the following two NP-hardness results (the proofs are in the full version).

▶ **Theorem 21.** DMC *is NP-hard*

(i) *for* $k = 2$ *and* $\alpha + 3 \leq \beta$*, and*
(ii) *for* $k = 3$ *and* $\alpha = \beta$*.*

The proof for Theorem 21 is based on rather technical reductions from (3, B2)-SAT [2] and Cubic Monotone 1-in-3 SAT [22]. The challenge here is to ensure the bounds on the Hamming distances between all row pairs. To overcome this challenge, we adjust pairwise row distances by making heavy use of the matrix, in which one pair of rows has distance exactly two greater than any other:

▶ **Lemma 22.** *For each $n \geq 3$ and $i < i' \in [n]$, one can construct in $n^{O(1)}$ time, a matrix $\mathbf{B}^n_{i,i'} \in \{0,1\}^{n \times \ell}$ with n rows and $\ell := (\binom{n}{2} - 1)(2n-1)$ columns such that for all $h \neq h' \in [n]$,*

$$d(\mathbf{B}^n_{i,i'}[h], \mathbf{B}^n_{i,i'}[h']) = \begin{cases} \gamma(\mathbf{B}^n_{i,i'}) + 2 & \text{if } (h,h') = (i,i'), \\ \gamma(\mathbf{B}^n_{i,i'}) & \text{otherwise.} \end{cases}$$

The problem of deciding whether an incomplete matrix $\mathbf{S} \in \{1,-1,\square\}^{n \times n}$ can be completed into a Hadamard matrix as seen in Johnson [18], is equivalent to DMC with $n = \ell$ and $\alpha = \beta = n/2$. We conjecture that one can adapt the proof of Theorem 21 (ii) to show the NP-hardness of this problem. We also conjecture that DMC with $k = 3$ is actually NP-hard for every value of $\beta - \alpha$. Similar reductions might work here as well. By contrast, we believe the case $k = 2$ and $\beta - \alpha = 1$ to be polynomial-time solvable, again by reducing to 2-SAT.

5 Conclusion

Together with the recent work of Eiben et al. [9], we are seemingly among the first in the context of stringology that make extensive use of Deza's theorem and sunflowers. While Eiben et al. [9] achieved classification results in terms of parameterized (in)tractability, we conducted a detailed complexity analysis in terms of polynomial-time solvable versus NP-hard cases. Figure 2 provides a visual overview on our results for Diameter Matrix Completion (DMC), also spotting concrete open questions.

Going beyond open questions directly arising from Figure 2, we remark that it is known that the clustering variant of DMC can be solved in polynomial time when the number of clusters is two and the matrix is complete [15]. Hence, it is natural to ask whether our tractability results can be extended to this matrix completion clustering problem as well. Furthermore, we proved that there are polynomial-time algorithms solving DMC when $\beta \leq 3$ and $\alpha = 0$ (Theorems 12 and 13). This leads to the question whether these algorithms can be extended to matrices with arbitrary alphabet size. Next, we are curious whether the phenomenon we observed in Theorem 17 concerning the exponential dependence of the running time for $(\alpha, \alpha+1)$-DMC when α is even but independence of α when it is odd can be further substantiated or whether one can get rid of the "α-dependence" in the even case. In terms of standard parameterized complexity analysis, we wonder whether DMC is fixed-parameter tractable with respect to $\beta + k$ (in our NP-hardness proof for the case $\beta = 4$ (Theorem 18) we have $k \in \theta(\ell)$). Finally, performing a multivariate fine-grained complexity analysis in the same spirit as in recent work for Longest Common Subsequence [3] would be another natural next step.

References

1 Vineet Bafna, Sorin Istrail, Giuseppe Lancia, and Romeo Rizzi. Polynomial and APX-hard cases of the individual haplotyping problem. *Theoretical Computer Science*, 335(1):109–125, 2005.

2 Piotr Berman, Marek Karpinski, and Alex D. Scott. Approximation hardness of short symmetric instances of MAX-3SAT. *Electronic Colloquium on Computational Complexity (ECCC)*, 049, 2003.

3 Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *29th Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA '18)*, pages 1216–1235, 2018.

4 Laurent Bulteau, Vincent Froese, and Rolf Niedermeier. Tight hardness results for consensus problems on circular strings and time series. *SIAM Journal on Discrete Mathematics*, 34(3):1854–1883, 2020.

5 Laurent Bulteau, Falk Hüffner, Christian Komusiewicz, and Rolf Niedermeier. Multivariate algorithmics for NP-hard string problems. *Bulletin of the EATCS*, 114, 2014.

6 Laurent Bulteau and Markus L. Schmid. Consensus strings with small maximum distance and small distance sum. *Algorithmica*, 82(5):1378–1409, 2020.

7 Michel Deza. Une propriété extrémale des plans projectifs finis dans une classe de codes équidistants. *Discrete Mathematics*, 6(4):343–352, 1973.

8 Michel Deza. Solution d'un problème de Erdős-Lovász. *Journal of Combinatorial Theory, Series B*, 16(4):166–167, 1974.

9 Eduard Eiben, Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider. On clustering incomplete data. *CoRR*, abs/1911.01465, 2019. arXiv:1911.01465.

10 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

11 Vincent Froese, René van Bevern, Rolf Niedermeier, and Manuel Sorge. Exploiting hidden structure in selecting dimensions that distinguish vectors. *Journal of Computer and System Sciences*, 82(3):521–535, 2016.

12 Harold N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *15th Annual ACM Symposium on Theory of Computing, (STOC '83)*, pages 448–456, 1983.

13 Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider. On the parameterized complexity of clustering incomplete data into subspaces of small rank. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, (AAAI '20)*, pages 3906–3913, 2020.

14 Robert Ganian, Iyad A. Kanj, Sebastian Ordyniak, and Stefan Szeider. Parameterized algorithms for the matrix completion problem. In *35th International Conference on Machine Learning, (ICML '18)*, volume 80 of *Proceedings of Machine Learning Research*, pages 1642–1651. PMLR, 2018.

15 Leszek Gąsieniec, Jesper Jansson, and Andrzej Lingas. Approximation algorithms for Hamming clustering problems. *Journal of Discrete Algorithms*, 2(2):289–301, 2004.

16 Jens Gramm, Rolf Niedermeier, and Peter Rossmanith. Fixed-parameter algorithms for Closest String and related problems. *Algorithmica*, 37(1):25–42, 2003.

17 Danny Hermelin and Liat Rozenberg. Parameterized complexity analysis for the closest string with wildcards problem. *Theoretical Computer Science*, 600:11–18, 2015.

18 Charles R Johnson. Matrix completion problems: a survey. In *Matrix Theory and Applications*, volume 40 of *Proceedings of Symposia in Applied Mathematics*, pages 171–198. American Mathematical Society, 1990.

19 Stasys Jukna. *Extremal Combinatorics: With Applications in Computer Science*. Springer Science & Business Media, 2011.

20 Tomohiro Koana, Vincent Froese, and Rolf Niedermeier. Parameterized algorithms for matrix completion with radius constraints. In *31st Annual Symposium on Combinatorial Pattern Matching, (CPM '20)*, pages 20:1–20:14, 2020.

21 Ross Lippert, Russell Schwartz, Giuseppe Lancia, and Sorin Istrail. Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem. *Briefings in Bioinformatics*, 3(1):23–31, 2002.

22 Cristopher Moore and J. M. Robson. Hard tiling problems with simple tiles. *Discrete & Computational Geometry*, 26(4):573–590, 2001.

23 Markus L. Schmid. Finding consensus strings with small length difference between input and solution strings. *ACM Transactions on Computation Theory*, 9(3):13:1–13:18, 2017.

24 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005.

Absorbing Patterns in *BST*-Like Expression-Trees

Florent Koechlin ✉
Laboratoire d'Informatique Gaspard-Monge, Université Gustave Eiffel, Marne-la-Vallée, France

Pablo Rotondo ✉
Laboratoire d'Informatique Gaspard-Monge, Université Gustave Eiffel, Marne-la-Vallée, France

Abstract

In this article we study the effect of simple semantic reductions on random BST-like expression-trees. Such random unary-binary expression-trees are often used in benchmarks for model-checking tools. We consider the reduction induced by an absorbing pattern for some given operator $\circledast$, which we apply bottom-up, producing an equivalent (and smaller) tree-expression. Our main result concerns the expected size of a random tree, of given input size $n \to \infty$, after reduction. We show that there are two different thresholds, leading to a total of five regimes, ranging from no significant reduction at all, to almost complete reduction. These regimes are completely characterized according to the probability of the absorbing operator. Our results prove that random BST-like trees have to be considered with care, and that they offer a richer range of behaviours than uniform random trees.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics

Keywords and phrases BST trees, absorbing pattern, reduction, analytic combinatorics

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.48

Funding *Pablo Rotondo*: partially supported by the Projet RIN Alenor (Regional Project from French Normandy).

Acknowledgements The authors would like to thank Arnaud Carayol and Cyril Nicaud for their very useful advice and remarks in the writing of this paper.

1 Introduction

There are two main ways to evaluate the performances of an algorithm or of its implementation: a theoretical one studying its complexity, and a more practical one, using benchmarks. On the theoretical side, it is classical to study the worst-case performances, which only gives a partial view of the practical usability of an algorithm. For instance, bubblesort and quicksort have the same worst-case complexity but perform very differently in practice to the extent that quicksort is actually implemented in some standard libraries. Average complexity tries to remedy this problem by considering a more appropriate probabilistic model on the inputs. The problem with this approach is that one must find a distribution simple enough to be mathematically tractable whilst being complex enough to model accurately the *real life distribution*. In the literature, this latter aspect is often relegated to a second place by theorists, as even simple algorithms may be hard to analyze for the uniform distribution.

Practical approaches consist in executing the tools and measuring directly their performances. These benchmarks are performed on real-world test cases, when possible. These are often complemented with randomly generated ones, which is an easy way of generating test cases of arbitrary sizes. Note that in this setting one is free to choose more mathematically complex distributions for the inputs, provided that they are fast to generate, and reasonably close to real-life examples.

In this paper we concentrate on algorithms manipulating tree-like expressions. Tree-like expressions are ubiquitous in computer science, for describing regular languages, boolean formulas, LTL formulas, … In Figure 1 we give several examples of such expression trees.

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 48; pp. 48:1–48:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Note that this representation is purely syntactical and might be redundant; several trees can have the same semantics, i.e., represent the same object. For example, the logical formula in Figure 1 is equivalent to x_2.

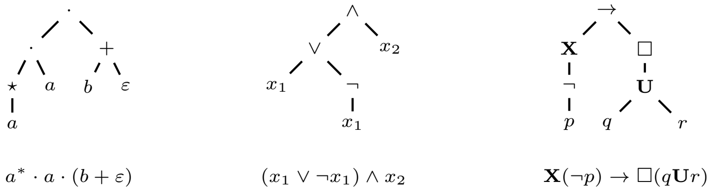

$a^* \cdot a \cdot (b + \varepsilon)$
$(x_1 \vee \neg x_1) \wedge x_2$
$\mathbf{X}(\neg p) \rightarrow \Box(q\mathbf{U}r)$

Figure 1 Three examples of expression trees. From left to right: a regular expression, a logical formula and an LTL formula.

In the absence of information on the real-life distribution for the algorithms, the uniform distribution is commonly considered: it appears a natural choice as in some sense it maximizes the coverage of possible inputs. Recently it was shown in [11, 12] that the uniform distribution is not relevant in benchmarks for tools manipulating tree-expressions. The authors proved that even if this distribution offers a good coverage of the tree-expressions, their coverage is poor when it comes to the objects represented by these tree-expressions, in the presence of simple simplifications. More precisely, the authors consider tree expressions with an operator $\circledast$ which admits a particular fixed tree $\mathcal{P}$, called the *absorbing pattern*, such that $\mathcal{P} \circledast t$, $t \circledast \mathcal{P}$ and $\mathcal{P}$ represent the same object. This situation occurs in most natural examples. For instance, `False` is absorbing for $\wedge$, $(a + b)^\star$ is absorbing for $+$ in regular expressions, 0 is absorbing for $\times$ in arithmetic expressions, etc. In the presence of an absorbing pattern, one can reduce a tree-expression in a bottom-fashion in order to remove all occurrences of a tree-expression of the form $\mathcal{P} \circledast t$ or $t \circledast \mathcal{P}$. This reduction can be performed in linear time and preserves the object represented by the tree-expression. Surprisingly, the authors proved that if we draw uniformly at random a tree-expression of size n the expected size of its reduced tree-expression is bounded by a constant. Hence using the uniform distribution in a benchmark might only be testing the ability of the tool to perform simple reductions on expressions.

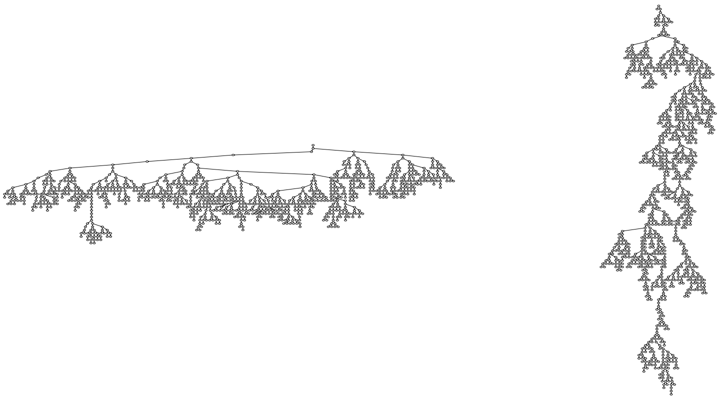
Looking at the random generators used in most benchmarks for model-checking tools dealing with tree-expressions, we see that the distribution produced is not the uniform one. In the case of random LTL formulas, Algorithm 1 presents the algorithm used by the tool `LTL-to-Büchi translator testbench` (`lbtt`) of `TCS` [17] (see also [4] and `Spot` [6] for other examples). They are all based on well-known distributions in combinatorics called *BST-like* distributions [7] as they are strongly related to random binary search trees.

The procedure used to generate such random expression trees of size n, where n is the number of nodes, is the following one:

(0) assign a probability distribution for the operators, and another one for the leaves;
(1) if $n = 1$ then draw a random leaf;
(2) if $n = 2$, then draw an operator of arity 1 following their probabilities, and a random leaf;
(3) if n is greater than 2, draw a random operator for the root. If the operator is unary, proceed to build a subtree of size $n - 1$. If the operator is binary, draw first uniformly the sizes for the left and right subtrees (so that they add to $n - 1$), and recursively generate these subtrees.

Algorithm 1 The pseudo-code used in `lbtt` [17, p.46] to draw a random LTL formula.

```
function RandomFormula($n$):
if $n$ = $1$ then
    $p$ := random symbol in $AP \cup \{\top, \bot\}$;
    return $p$;
else if $n = 2$ then
    $op$ := random operator in $\{\neg, \mathbf{X}, \Box, \Diamond\}$;
    $f$ := RandomFormula(1);
    return $op\ f$;
else
    $op$ := random operator in $\{\neg, \mathbf{X}, \Box, \Diamond, \wedge, \vee, \rightarrow, \leftrightarrow, \mathbf{U}, \mathbf{R}\}$;
    if $op$ in $\{\neg, \mathbf{X}, \Box, \Diamond\}$ then
        $f$ := RandomFormula($n - 1$);
        return $op\ f$;
    else
        $x$ := uniform integer in $[1, n - 2]$;
        $f_1$ := RandomFormula($x$);
        $f_2$ := RandomFormula($n - x - 1$);
        return ($f_1\ op\ f_2$);
```



(a) A *BST-like* tree **(b)** A *uniform* tree

Figure 2 The expected height of a random BST tree (on the left) of size n is $\Theta(\log(n))$ whereas for a uniform tree (on the right) of size n, it is $\Theta(\sqrt{n})$.

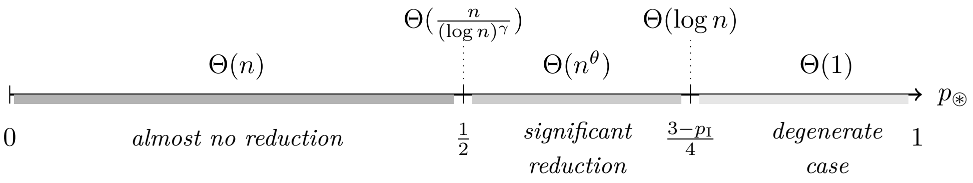
The resulting distribution over the trees with n nodes is not uniform. In fact, the shape of a typical tree drawn from the BST-like distribution differs greatly from that of a tree drawn from the uniform distribution [5, 8]. It can be seen by comparing Figure 2a and Figure 2b. This difference is also apparent when it comes to the average behaviour of algorithms. It was shown in [14] that the Glushkov automaton (a.k.a., the position automaton) of a regular expression under a BST-like distribution has an average of $\Theta(n^2)$ transitions, in stark contrast to the case of the uniform distribution, where it was previously showed [13] that the average is $\Theta(n)$. Observe that for the uniform distribution, if we reduce the expression (according to the absorbing pattern $(a + b)^*$ for $+$) first, the expected size of the Glushkov automaton is in fact bounded by a constant, as a consequence of [11, 12].

It seems likely that the choice of the BST-like distributions in benchmark is motivated by its greater flexibility to model real-word distributions (i.e. by playing with the operator probabilities) and also because of its very efficient generation procedure.

Seeing the flaw of the uniform distribution for tree-expressions discovered in [11, 12], it is natural to wonder if the BST-like distributions suffer from the same short-comings. This question is the starting point of the work present in this paper. We assume the existence of an absorbing pattern $\mathcal{P}$ for some operator $\circledast$ and study the expected size of the tree-expression after reduction[1].

Our main result paints a complete picture of the possible asymptotic behaviour of the expected size after reduction as the original size n tends to infinity. We show that there are two different thresholds, leading to a total of five regimes, depending on the probability $p_\circledast$ of the absorbing operator and the probability of drawing a unary operator p_I. The main regimes are shown experimentally in Figure 3.

▶ **Theorem.** *Consider a family of expression trees defined from unary and binary operators. Suppose there is a tree pattern $\mathcal{P}$, of size at least[2] 3, that is absorbing for a distinguished operator $\circledast$. We consider the simplification algorithm that consists in inductively changing a $\circledast$-node by $\mathcal{P}$ whenever one of its children can be simplified into $\mathcal{P}$. Then the expected size of a random BST-like tree after simplification has an asymptotic behaviour given by the following cases, depending on the probability $p_\circledast$ of the absorbing operator:*



There are two critical points $p_\circledast = 1/2$ and $p_\circledast = (3 - p_\mathrm{I})/4$, the latter depending on the probability p_I of drawing a unary operator. This gives a total of five regimes spanning the spectrum from almost no reduction $\Theta(n)$ to complete reduction $\Theta(1)$. The exponents γ and θ are given by $\gamma = \frac{2}{1-p_\mathrm{I}}$ and $\theta = 1 - \frac{4p_\circledast - 2}{1-p_\mathrm{I}}$ respectively.

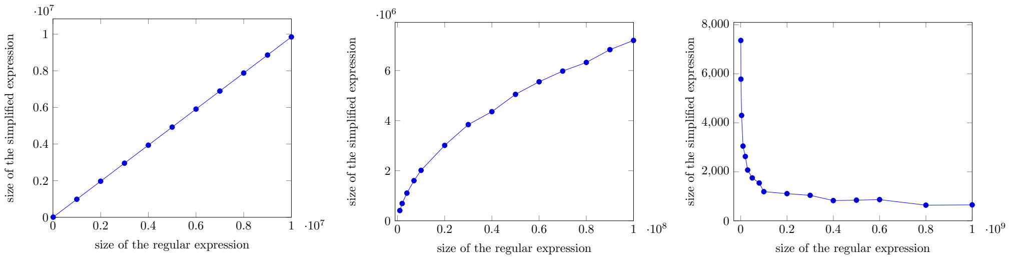


Figure 3 The three main regimes observed experimentally on regular expressions on two letters, with 10 000 samples for each size: (from left to right) linear ($p_+ = p_\star = p_\cdot = 1/3$), sublinear ($p_+ = 19/29$, $p_\star = p_\cdot = 5/29$) and constant ($p_+ = 8/10$, $p_\star = p_\cdot = 1/10$).

[1] Particular cases where these and many more reductions are available have been studied in the literature [2, 3, 10]. However, these consider a very particular example, namely the $\wedge/\vee$-trees with $p_\wedge = p_\vee = \frac{1}{2}$.

[2] This restriction is not a real constraint. For $|\mathcal{P}| \leq 2$ it is easy to build from $\mathcal{P}$ a larger absorbing pattern and our result then applies. See the discussion at the end of Section 2.2.

Methods. To obtain our results we employ techniques from the framework of Analytic Combinatorics [9]. The recursive procedure used to produce a random BST-like tree naturally leads in turn to a recurrence for the probabilities of interest. We encode the probabilities into ordinary generating functions, and the recurrences lead formally to differential equations (of Riccati type) for the expected value of the size after reduction. This first part is purely symbolic. At this point, as is usual in Analytic Combinatorics, we see our generating functions as power series on the complex plane to take advantage of the powerful theory of holomorphic functions. The singularities, i.e., the points where these functions cease to be smooth, are related to the asymptotics of the coefficients. In particular, those that are closer to the origin (i.e., dominant singularities) give the leading terms. This link is formally given by the Transfer Theorem [9, Ch VI.3], which translates asymptotic equivalents for the generating functions near the singularities to asymptotics for their coefficients.

Plan of the article. In Section 2 we give precise definitions for our settings, namely random BST-like expression trees, absorbing patterns and the ensuing simplification. Section 3 gives a general overview of the techniques and gives an outline of the proof of our main Theorem. It explains in particular why we need to consider the probability of fully reducible trees first (those reducing to $\mathcal{P}$), in order to prove our main result. Section 4 then is devoted to these fully reducible trees, and as a side product we prove (see Thm 10) that the probability of being fully reducible tends to 0 for $p_{\circledast} \leq 1/2$ and to a positive constant otherwise. This first threshold is intimately linked with that of the main result. Finally, Section 5 completes the sketched proof of the main theorem.

Most proofs are either sketched or omitted in this extended abstract.

2 Model, definitions and probability of complete reduction

2.1 The BST-like model

Consider three non-empty sets of labels $\mathcal{A}_0$, $\mathcal{A}_1$, and $\mathcal{A}_2$, corresponding respectively to the sets of possible leaves, unary operators and binary operators. For example, to describe the set of regular expressions on the alphabet $\{a, b\}$: $\mathcal{A}_0 = \{\varepsilon, a, b\}$, $\mathcal{A}_1 = \{\star\}$ and $\mathcal{A}_2 = \{\cdot, +\}$.

We define the family of trees on $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ in Definition 1 below. It is important to emphasize that when we say trees we actually mean planar trees throughout the article: the order of the branches does matter, hence $\overset{op}{\underset{T_1\ T_2}{/\backslash}}$ and $\overset{op}{\underset{T_2\ T_1}{/\backslash}}$ are not the same tree.

▶ **Definition 1** (Expression trees). *The family $\mathcal{E}(\mathcal{A})$ of expression trees over $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ is defined inductively:*

- *any leaf $a_0 \in \mathcal{A}_0$ is an expression tree;*
- *if T is an expression tree and $op_1 \in \mathcal{A}_1$ is a unary operator, then $\overset{op_1}{\underset{T}{|}} \in \mathcal{E}(\mathcal{A})$;*
- *if T_1, T_2 are expression trees and $op_2 \in \mathcal{A}_2$ is a binary operator, then $\overset{op_2}{\underset{T_1\ T_2}{/\backslash}} \in \mathcal{E}(\mathcal{A})$.*

The size $|T|$ of an expression tree T is its number of nodes (operators and leaves).

Now that we have defined the family of expression trees, we introduce the BST-like distribution over them. For $n \in \mathbb{N}$, let $\mathcal{E}_n$ denote the set of expression trees of size n.

First we endow the set of leaves $\mathcal{A}_0$ and the set of operators $\mathcal{A}_{\mathsf{ops}} = \mathcal{A}_1 \cup \mathcal{A}_2$ with probabilities, $(p_a)_{\mathcal{A}_0}$ and $(p_{op})_{\mathcal{A}_{\mathsf{ops}}}$. Remark then that $\sum_{a \in \mathcal{A}_0} p_a = 1$ and $\sum_{op \in \mathcal{A}_{\mathsf{ops}}} p_{op} = 1$.

We denote by p_{I} the probability of picking a unary operator, i.e., $p_{\mathrm{I}} = \sum_{op_1 \in \mathcal{A}_1} p_{op_1}$.

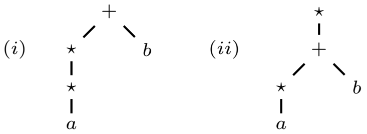


Figure 4 Example of two trees of size $n = 5$, for regular expressions, having different probabilities for any choice of distribution. The probability of the trees in (i) and (ii) are $\frac{1}{3}p_+p_\star p_a p_b$ and $\frac{1}{2}p_+p_\star p_a p_b$ respectively.

▶ **Definition 2** (Random BST-like expression tree). *A random BST-like expression tree of size $n \in \mathbb{N}_{\geq 1}$ is built recursively as follows:*

- *If $n = 1$, we draw a leaf from $\mathcal{A}_0$ according to the probability distribution $(p_a)_{a\in\mathcal{A}_0}$.*
- *If $n = 2$, we draw a unary operator op_1 according to the normalized probabilities $\left(\frac{1}{p_\text{I}}p_{op_1}\right)_{op_1\in\mathcal{A}_1}$, then we draw independently a tree $a_0 \in \mathcal{A}_0$ of size 1 and return $\overset{op_1}{\underset{a_0}{|}}$.*
- *If $n \geq 3$, we pick an operator $\oplus \in \mathcal{A}_\texttt{ops}$ according to the distribution $(p_{op})_{op\in\mathcal{A}_\texttt{ops}}$. $(\star)$ If we obtain a unary operator $\oplus \in \mathcal{A}_1$, we produce recursively and independently a tree T of size $n-1$ and return $\overset{\oplus}{\underset{T}{|}}$. $(\star\star)$ If otherwise we obtain a binary operator $\oplus \in \mathcal{A}_2$, we pick the size k of the left subtree uniformly from $\{1,\dots,n-2\}$ and produce independently two trees T_L and T_R of sizes k and $n-1-k$ respectively. Then we return $\underset{T_L\ \ T_R}{\overset{\oplus}{/\backslash}}$.*

For Definition 2 to make sense, we assume that $p_\text{I} > 0$. Note that otherwise (if $p_\text{I} = 0$) we would produce no trees of size 2, or any even size. This assumption is not really a constraining one, as otherwise we would obtain similar results, but just over the odd sizes.

The procedure of Definition 2 defines a probability distribution over expression trees: the probability $\Pr_n(T)$ of a given expression tree T of size n is the probability of the algorithm in Definition 2 returning T with input n. This distribution is not uniform, as shown in Figure 4.

2.2 Absorbing pattern and reduction

We now define what we mean by an *absorbing pattern* for the family of expression trees $\mathcal{E}(\mathcal{A})$. Fix a binary operator $\circledast \in \mathcal{A}_2$ and an expression tree $\mathcal{P} \in \mathcal{E}(\mathcal{A})$. Informally, the associated simplification $\sigma = \sigma_{\mathcal{P},\circledast}$ is defined by applying bottom up the substitution

$$\underset{T_1\qquad T_2}{\overset{\circledast}{/\quad\backslash}} \rightsquigarrow \mathcal{P}\,,\ \text{whenever } T_i = \mathcal{P} \text{ for some } i \in \{1,\dots,2\}.$$

More precisely, we define recursively the simplification $\sigma = \sigma_{\mathcal{P},\circledast} \colon \mathcal{E}(\mathcal{A}) \to \mathcal{E}(\mathcal{A})$ with absorbing pattern $\mathcal{P}$ for the operator $\circledast$ as follows: if $e \in \mathcal{A}_0$ we set $\sigma(e) = e$ while,

- $\sigma\left(\overset{op_1}{\underset{T}{|}}\right) = \overset{op_1}{\underset{\sigma(T)}{|}}$ for $op_1 \in \mathcal{A}_1$,
- $\sigma\left(\underset{T_1\ \ T_2}{\overset{op_2}{/\backslash}}\right) = \underset{\sigma(T_1)\ \ \sigma(T_2)}{\overset{op_2}{/\backslash}}$ for $op_2 \in \mathcal{A}_2$ with $op_2 \neq \circledast$,
- $\sigma\left(\underset{T_1\ \ T_2}{\overset{\circledast}{/\backslash}}\right) = \mathcal{P}$ if $\sigma(T_i) = \mathcal{P}$ for $i = 1$ or $i = 2$, and $\sigma\left(\underset{T_1\ \ T_2}{\overset{\circledast}{/\backslash}}\right) = \underset{\sigma(T_1)\ \ \sigma(T_2)}{\overset{\circledast}{/\backslash}}$ otherwise.

An expression tree $T \in \mathcal{E}(\mathcal{A})$ is said to be *fully reducible* when $\sigma(T) = \mathcal{P}$.

Henceforth we assume that our family of expression trees $\mathcal{E}(\mathcal{A})$ admits an absorbing $\mathcal{P}$ of size $s := |\mathcal{P}|$ for a fixed binary operation $\circledast \in \mathcal{A}_2$. For technical reasons, we will suppose that $s \geq 3$. This might seem in contradiction to the fact that some leaf can be absorbing (for instance `False` with $\vee$). However this is not much of a restriction since you can always build, from an absorbing pattern $\mathcal{P}$ of size less than 3, a new one of size more than 3 by considering $\mathcal{P}' := \begin{smallmatrix} & \circledast & \\ & / \backslash & \\ \mathcal{P} & & a \end{smallmatrix}$ for a leaf a. This new pattern leads to less reductions in comparison to the former one, so that our results give upperbounds for the expected size after reduction by an absorbing pattern of size less than 3, instead of exact equivalents.

3 Outline of the proof

For our proof we employ methods from the framework of Analytic Combinatorics [9]: we will represent a sequence $(a_n)_{n\geq 0}$ of coefficients by its *ordinary generating function* (OGF for short) $F(z) = \sum a_n z^n$. At first, we treat $F(z)$ as a formal object, and our goal is to obtain an equation characterizing it. Typically, in Analytic Combinatorics, this first step is done by building the studied combinatorial class from set operations, and using a toolbox to translate them into operations between generating functions. In our case it does not apply and we have to extract the equations for the generating functions from the recurrence relations satisfied by the related sequences. This approach is common when the distribution of the studied combinatorial class is not uniform (see for instance [9, 15]). Hence we begin the proof by deriving a recurrence relation satisfied by the expected size (e_n). This relation comes from the recursive nature of the algorithm for constructing a random BST-like tree-expression.

3.1 Recurrence relations

Recurrence for the expected value

We are interested in the probabilities $p_{n,k} := \Pr_n\{T : |\sigma(T)| = k\}$ for a tree of size n to have a reduced size k. More precisely we want to obtain an equation for $e_n := \sum_k k \cdot p_{n,k}$ which is the expected size after reduction for a random tree of fixed size n, according to the BST-like distribution[3]. The following proposition gives the recurrence satisfied by the sequence $(e_n)_n$. It involves the probability that a tree T of size n is fully reducible:

$$\gamma_n = \Pr_n\{\sigma(T) = \mathcal{P}\}\,.$$

We also write $p_{\mathrm{II}} := 1 - p_{\mathrm{I}} - p_{\circledast}$, the probability of drawing a binary operator that is not $\circledast$.

▶ **Proposition 3.** *The sequence (e_n) of expected sizes after reduction satisfies, for all $n > 1$:*

$$e_{n+1} = 1 + (s-1)\gamma_{n+1}\mathbf{1}_{n+1\neq s} + p_{\mathrm{I}} e_n + \frac{2p_{\mathrm{II}}}{n-1}\sum_{j=1}^{n-1} e_j + \frac{2p_{\circledast}}{n-1}\sum_{j=1}^{n-1}(e_j - s\gamma_j)(1-\gamma_{n-j})\,.$$

Proof sketch. We introduce the auxiliary polynomials $F_n(u) = \sum_{k=0}^{n} \Pr_n\{T : |\sigma(T)| = k\}u^k$. These satisfy the recurrence

$$F_{n+1}(u) = \gamma_{n+1}\mathbf{1}_{n+1\neq s}u^s + p_{\mathrm{I}} u F_n(u) + u\frac{p_{\mathrm{II}}}{n-1}\sum_{j=1}^{n-1} F_j(u)F_{n-j}(u)$$
$$+u\frac{p_{\circledast}}{n-1}\sum_{j=1}^{n-1}\left(F_j(u) - \gamma_j u^s\right)\left(F_{n-j}(u) - \gamma_{n-j}u^s\right).$$

[3] From now on, when we write random, we implicitly mean for the BST-like distribution.

Indeed, a tree T of size $n+1$ is either fully reducible (with probability γ_{n+1}) or not. When we pick $\circledast$, the new tree does not reduce to $\mathcal{P}$ only when the subtrees are not fully reducible.

Then e_n is expressed as $e_n = F_n'(1)$. Differentiating the formula and setting $u=1$ we obtain the recurrence for e_n, using the fact that $F_k(1)=1$ for all k. ◀

Recurrence for the probability of full reduction

The recurrence for the expected values (e_n) in Proposition 3 depends strongly on the auxiliary sequence of probabilities $(\gamma_n)_{n\geq 1}$. Clearly, any tree starting by the absorbing operator $\circledast$ and having a fully-reducible child is also fully reducible. Reciprocally, if a tree of size strictly bigger than s is fully reducible, then it has $\circledast$ as a root and at least one fully reducible child. Hence the sequence (γ_n) satisfies a recurrence, which is not linear:

$$\gamma_{n+1} = p_{\circledast} \cdot \frac{1}{n-1} \sum_{k=1}^{n-1} \left(\gamma_k + \gamma_{n-k} - \gamma_k \gamma_{n-k}\right) , \quad \text{for all } n \geq s. \tag{1}$$

Indeed, suppose that k is the size of the left subtree[4], which happens with probability $\frac{1}{n-1}$. Then the probability that one of the children is fully reducible is, by inclusion-exclusion, $\gamma_k + \gamma_{n-k} - \gamma_k\gamma_{n-k}$.

In our study of the sequence $(\gamma_n)_{n\geq 1}$ we will show that it actually converges (Thm. 10). For the time being, we will just remark that if $(\gamma_n)_n$ converges, only certain values are possible for $L = \lim \gamma_n$. For this, let us recall this classical result:

▶ **Lemma 4** (Cèsaro-means). *Consider a sequence $(a_n)_{n\geq 1}$ converging to a real number L. Then we have $\lim_n \frac{1}{n}\sum_{k=1}^n a_k = L$, and $\lim_n \frac{1}{n}\sum_{k=1}^n a_k\, a_{n+1-k} = L^2$.*

From Eq. (1) we see that $L = p_{\circledast} \cdot (2L - L^2)$. Thus the limit, if it exists, can only be 0 or $\gamma_\infty := 2 - 1/p_{\circledast}$. For $p_{\circledast} < 1/2$, we have $\gamma_\infty < 0$ and so $L = 0$. For $p_{\circledast} > 1/2$, Theorem 10 will show that $L = \gamma_\infty$. These limits hint at the possibility of a threshold for e_n at $p_{\circledast} = \frac{1}{2}$.

3.2 Main steps

In order to study the sequence of expected sizes $(e_n)_{n\geq 1}$ it will be necessary to study first the sequence of probabilities $(\gamma_n)_{n\geq 1}$. As announced, we introduce their generating functions:

$$A(z) := \sum_{n=0}^{\infty} \gamma_n z^n , \quad E(z) := \sum_{n=0}^{\infty} e_n z^n .$$

The proof, as is usual in Analytic Combinatorics, proceeds in two steps: a symbolic step and an analytic step. In the symbolic step we obtain appropriate equations for our generating functions, seen as purely formal power series. In our case it will be differential equations, coming from the recurrences. Then in the analytic step, the generating functions are seen as analytic functions of a complex variable. We apply the celebrated *Transfer theorem* (see [9, Ch VI.3]) to obtain the asymptotic equivalents of the sequences. The Transfer Theorem states that, under analytic conditions, an equivalent $E(z) \sim_{z\to 1} \lambda(1-z)^{-\alpha}$ with $\alpha \notin \{0,-1,-2,\ldots\}$, implies $e_n \sim \lambda n^{\alpha-1}/\Gamma(\alpha)$, where Γ is Euler's Gamma-function, the generalized factorial.

[4] We have supposed that there are trees of every possible size, which is equivalent to $p_{\mathrm{I}} > 0$.

Symbolic step

The recurrence (1) for γ_n, as well as the recurrence of e_n in Proposition 3, lead naturally to ordinary differential equations for $A(z)$ and for $E(z)$. As the formal derivative of a series $F(z) = \sum a_n z^n$ is given by $F'(z) = \sum (n+1) a_n z^n$, multiplying Eq. (1) by $(n-1)z^n$ and summing will introduce derivatives. Thus we obtain a differential equation for $A(z)$, under the form of a Riccati equation, and a linear one for $E(z)$, which involves the generating function $A(z)$ as a known quantity:

$$A'(z) = (s-2)\gamma_s z^{s-1} + \left(\frac{2}{z} + 2p_{\circledast}\frac{z}{1-z}\right) A(z) - p_{\circledast} \cdot (A(z))^2 \,, \tag{2}$$

and, for a certain function $F(x, y)$, which can be made explicit

$$E'(z) = F(z, A(z)) + \tfrac{1}{1-p_{\mathrm{I}} z}\left(\tfrac{2}{z} - p_{\mathrm{I}} + 2\,(1-p_{\mathrm{I}})\,\frac{z}{1-z} - 2p_{\circledast} A(z)\right) \cdot E(z)\,.$$

These differential equations constitute our symbolic specifications for the generating functions $A(z)$ and $E(z)$. At this point we switch to their analytic study.

Analytic step

The equation for $E(z)$ is a first order linear ODE, as such it can be solved by the method of variation of constants[5] [1, Th. 6.1] to obtain an explicit solution that involves $A(z)$ as a known quantity. Thus we need first to study $A(z)$ as a complex function, and in particular its domain of analyticity. Since the coefficients of $A(z)$ are probabilities $\gamma_n \in [0, 1]$, the series $A(z)$ defines an analytic function on the unit disk $|z| < 1$. However, for technical reasons we need further information regarding its domain of analyticity in order to apply the Transfer Theorem. Thus in Section 4 we are going into more detail, showing that $z = 1$ is a dominant singularity and that $A(z)$ can be extended analytically to the domain $\Omega = \mathbb{C} \setminus [1, \infty)$. We remark that then the same holds for $E(z)$.

The last hypothesis in order to apply the Transfer Theorem for $E(z)$ is its asymptotic equivalent as $z \to 1$, its dominant singularity. The solution of the ODE for $E(z)$ yields a fundamental approximation

$$E(z) \approx \frac{C}{(1-z)^2} \exp\left(-2p_{\circledast}\int_0^z \frac{A(w)}{1-p_{\mathrm{I}} w} dw\right) \times \left(2 + \int_0^z G(z) \exp\left(2p_{\circledast}\int_0^{\zeta} \frac{A(w)}{1-p_{\mathrm{I}} w} dw\right) d\zeta\right)$$

as $z \to 1$, for a certain constant $C > 0$ and a bounded function $G(z)$.

This means that to study the asymptotic behaviour for $E(z)$ we require quite precise asymptotics regarding $A(z)$ near $z = 1$. In particular, we need to be able to integrate the approximation, and obtain a good approximation after taking the exponential. Thus we need not only an asymptotic equivalent for $A(z)$ as $z \to 1$, but also a remainder term. The integration involving $A(z)$ is dealt with by the Singular Integration Theorem [9, Thm VI.9].

Analysis of $A(z)$ around its dominant singularity. First we turn the Riccati equation (2) into a linear second order ODE that is homogeneous by a classical change of the unknown function $p_{\circledast} A(z) = v'(z)/v(z)$. We analyze the new function $v(z)$ by the Frobenius method [1, pp.181-182] to obtain a local form of $v(z)$ around the singularity $z = 1$. The conclusion

[5] We adapt it for our case. In fact $\frac{2}{z}$ is not defined at $z = 0$, where we give our initial condition precisely.

can be found in Proposition 9, which shows that we have 3 regimes for $A(z)$ depending on whether $p_{\circledast}$ is less, equal, or greater than $1/2$. As a by-product, the Transfer Theorem implies (see Theorem 10) that γ_n tends to 0 for $p_{\circledast} \leq 1/2$ and to the constant $\gamma_\infty > 0$ when $p_{\circledast} > 1/2$. The detailed analysis is explained in Section 4.

Analysis of $E(z)$ around its dominant singularity. We follow the cases of Proposition 9, which already gives the threshold $1/2$. Then there is an extra threshold coming from the term

$$2 + \int_0^z G(z) \exp\left(2p_{\circledast} \int_0^{\zeta} \frac{A(w)}{1 - p_{\mathrm{I}} w} dw\right) d\zeta$$

in the estimate for $E(z)$. This new threshold corresponds exactly to the point where the integral goes from being convergent to divergent as $z \to 1$.

For example, when $p_{\circledast} < 1/2$, Proposition 9 yields that the integral $\int_0^z \frac{A(w)}{1-p_{\mathrm{I}}w} dw$ converges as $z \to 1$. From our approximation for $E(z)$ we see that actually $E(z) \sim \lambda(1-z)^{-2}$ for a certain constant $\lambda > 0$. By the Transfer Theorem this implies that $e_n \sim \lambda \cdot n$ as $n \to \infty$.

4 Fully reducible trees

In this section, we study the probability of being fully reducible $\gamma_n = \Pr_n\{\sigma(T) = \mathcal{P}\}$. This is motivated by the fact that γ_n intervenes in the recurrence for the expected value e_n of the size of a random BST-like tree after reduction, see Section 3. We recall that we have the following recurrence for $(\gamma_n)_{n \geq 1}$: $\gamma_{n+1} = p_{\circledast} \cdot \frac{1}{n-1} \sum_{k=1}^{n-1} (\gamma_k + \gamma_{n-k} - \gamma_k \gamma_{n-k})$ for all $n \geq s$.

4.1 Generating function and its Riccati equation

As announced, we study γ_n by looking at its generating function $A(z) = \sum_{n=0}^{\infty} \gamma_n z^n$. Note that its radius of convergence is at least 1 because the coefficients γ_n belong to $[0, 1]$. The following proposition shows that it is exactly 1.

▶ **Proposition 5.** *The radius of convergence of $A(z)$ is exactly* 1.

Proof. We work by contradiction. Suppose $\sum_k \gamma_k$ was convergent. The inequality $\gamma_k + \gamma_{n-k} - \gamma_k \gamma_{n-k} \geq \gamma_k$, valid for all k, implies $\gamma_n \geq \frac{p_{\circledast}}{n-1} \sum_{k=1}^{n-1} \gamma_k = \Omega(1/n)$ from the recurrence in Eq (1). This is absurd because of the divergence of the Harmonic sums. ◀

We recall the Riccati differential equation (2) satisfied by $A(z)$:

$$A'(z) = (s-2)\gamma_s z^{s-1} + \left(\frac{2}{z} + 2p_{\circledast} \frac{z}{1-z}\right) A(z) - p_{\circledast} \cdot (A(z))^2.$$

Consider now the function[6] $v(z) = \exp\left(p_{\circledast} \int_0^z A(w) dw\right)$, which satisfies $A(z) = 1/p_{\circledast} \cdot v'(z)/v(z)$. This is a classical transformation to turn any Riccati equation into a linear ODE of order two. For our case we obtain

$$v''(z) = p_{\circledast} \cdot (s-2)\gamma_s z^{s-1} v(z) + \left(\frac{2}{z} + 2p_{\circledast} \frac{z}{1-z}\right) v'(z)\,. \tag{3}$$

The function $v(z)$ is analytic on the disk $|z| < 1$ as $A(z)$ is analytic there.

[6] Here $\int_0^z$ means that we integrate on the segment from 0 to z on the complex plane.

The domain of analyticity and the local behaviour of solutions of linear ODE are well understood [1, 16, 18]. We exploit this now to show in Proposition 6 that $A(z)$ is actually analytic on the larger domain $\Omega = \mathbb{C} \setminus [1, \infty)$. Later on (see Prop. 9) we will also use the local form of $v(z)$ to obtain asymptotic equivalents for $A(z)$ around its singularity $z = 1$, which are needed to apply the Transfer Theorem.

▶ **Proposition 6.** *The power series $A(z)$, seen as an analytic function, can be extended analytically to every point of the domain $\Omega = \mathbb{C} \setminus [1, \infty)$. In particular, $z = 1$ is the only singularity on the circle $|z| = 1$.*

Proof sketch. We already know that $v(z)$ is analytic on the disk $|z| < 1$. Then we use [18, Theorem 2.2, p.3] repeatedly and conclude with the uniqueness of analytic continuation. ◀

4.2 Asymptotics for the fully reducible trees

We can now derive the asymptotic behaviour of $v(z)$, where we recall that $v(z)$ satisfies $v''(x) - \left(\frac{2}{x} + 2p_{\circledast}\frac{x}{1-x}\right) v'(x) - p_{\circledast} \cdot (s-2)\gamma_s x^{s-1} v(x) = 0$, a linear equation of order 2, with non-constant coefficients. We analyze the asymptotics of the solutions close to the singularity by using the Frobenius method (see [1, pp.181-182]). For this we introduce some related notation.

▶ **Definition 7.** *Consider the homogeneous linear ODE of order two $y''(x) + d_1(x)y'(x) + d_2(x)y(x) = 0$, where $d_1(x)$ and $d_2(x)$ are meromorphic on a star-shaped domain $\tilde{\Omega}$.*

A point $\zeta \in \tilde{\Omega}$ is said to be a regular singularity *for the ODE, if it is a singularity of either $d_1(x)$ or $d_2(x)$, or maybe both, and the limits $\delta_j := \lim_{z\to\zeta}(z-\zeta)^j d_j(z)$, exist and are finite for $j = 1$ and 2. In this case, we define the* indicial polynomial *$I(\theta)$ at the regular singularity $z = \zeta$ by $I(\theta) = \theta(\theta - 1) + \delta_1\theta + \delta_2$.*

The following theorem explains how the indicial polynomial leads to the asymptotics of the solutions[7]:

▶ **Theorem 8** ([1, Thm 6.14–15])**.** *Consider the homogeneous linear ODE of order two $y''(x)+d_1(x)y'(x)+d_2(x)y(x) = 0$, where $d_1(x)$ and $d_2(x)$ are meromorphic on a star-shaped domain $\tilde{\Omega}$, and ζ a regular singularity for the given ODE.*

- *If the two roots θ_1 and θ_2 of the indicial polynomial associated to ζ do not differ by an integer (including 0 for double roots), then, in a slit neighbourhood of ζ inside $\tilde{\Omega}$, every solution $y(x)$ is of the form $c_1(\zeta - z)^{\theta_1}H_1(\zeta - z) + c_2(\zeta - z)^{\theta_2}H_2(\zeta - z)$, where $c_1, c_2 \in \mathbb{C}$, and $H_1(z), H_2(z)$ are analytic at $z = 0$ and $H_1(0) \neq 0$, $H_2(0) \neq 0$.*
- *If the indicial polynomial has a double root θ_0, then in a slit neighbourhood of ζ inside $\tilde{\Omega}$, every solution $y(x)$ is of the form $(z - \zeta)^{\theta_0}(c_1H_1(z - \zeta) + c_2 \log(z - \zeta)H_2(z - \zeta))$, where $c_1, c_2 \in \mathbb{C}$, and $H_1(z), H_2(z)$ are analytic at $z = 0$ and $H_1(0) \neq 0$, $H_2(0) \neq 0$.*

Using this theorem, we directly derive the local behaviour of $v(z)$ and $v'(z)$ around $z = 1$. Now we are ready to obtain the local expansion for $A(z) = \frac{1}{p_{\circledast}}v'(z)/v(z)$, around the singularity $z = 1$, and we prove the following proposition:

▶ **Proposition 9.** *The ordinary generating function $A(z)$ for $(\gamma_n)_{n\geq1}$ satisfies the following asymptotic expansions as $z \to 1$ over Ω*

[7] The reference uses $|\zeta - z|$ to avoid restricting the domain; here we can use $(\zeta - z)$ because we chose a determination of $\log(1 - z)$.

- *For* $p_\circledast > \frac{1}{2}$, $A(z) = \frac{\gamma_\infty}{1-z} + O((1-z)^{2p_\circledast - 2})$,
- *For* $p_\circledast = \frac{1}{2}$, $A(z) = \frac{2}{1-z}\left(\log\left(\frac{1}{1-z}\right)\right)^{-1} + O\left(\frac{1}{1-z}\left(\log\left(\frac{1}{1-z}\right)\right)^{-2}\right)$
- *For* $p_\circledast < \frac{1}{2}$, $A(z) \sim \frac{D}{(1-z)^{2p_\circledast}}$,

where we recall that $\gamma_\infty := (2p_\circledast - 1)/p_\circledast$ *and* $D > 0$ *is a constant depending on* $p_\circledast$ *and* s.

As a side product of this proposition, we can apply the Transfer Theorem and show that γ_n indeed converges:

▶ **Theorem 10.** *The probability* γ_n *of being fully reducible tends to the constant* $\gamma_\infty := (2p_\circledast - 1)/p_\circledast$ *for* $p_\circledast > \frac{1}{2}$ *and to zero otherwise. More precisely, for* $p_\circledast = \frac{1}{2}$ *we have* $\gamma_n \sim \frac{2}{\log n}$, *while for* $p_\circledast < \frac{1}{2}$, $\gamma_n \sim D \cdot n^{2p_\circledast - 1}/\Gamma(2p_\circledast)$, *where* D *is the constant from Prop. 9.*

▶ Remark 11. A different approach for the case $p_\circledast < 1/2$ yields the value of the constant for the asymptotics, $D = e^{-2p_\circledast} \cdot \left((s-2)\gamma_s \int_0^1 t^{s-3}(1-t)^{2p_\circledast} e^{2p_\circledast t} dt - p_\circledast \int_0^1 (A(t))^2 (1-t)^{2p_\circledast} e^{2p_\circledast t} dt\right)$. Furthermore, the first term in the parenthesis yields a simple upper-bound.

5 Main result: expected values

This section is devoted to the sketch of the proof of the main theorem:

▶ **Theorem 12.** *If the probability* p_{I} *of unary operators is not zero, then the expected size* e_n *of a random BST-like tree-expression of size* n *after the bottom-up reduction using an absorbing pattern for the binary operator* $\circledast$ *satisfies, for some positive constants* $c_1, \ldots, c_4$*:*

- *if* $p_\circledast < 1/2$, *then* $e_n \sim c_4\, n$;
- *if* $p_\circledast = 1/2$, *then* $e_n \sim c_3\, n \log(n)^{-2/(1-p_{\mathrm{I}})}$;
- *if* $p_\circledast > \frac{1}{2}$ *and* $4p_\circledast < 3 - p_{\mathrm{I}}$, *then* $e_n \sim c_2\, n^{1 - \frac{4p_\circledast - 2}{1 - p_{\mathrm{I}}}}$;
- *if* $4p_\circledast = 3 - p_{\mathrm{I}}$, *then* $e_n \sim c_1 \log(n)$;
- *if* $4p_\circledast > 3 - p_{\mathrm{I}}$, *then* $e_n \to e_\infty$, *where* e_∞ *is some positive constant.*

Thus we perform a precise study of the generating function $E(z)$ of the expected size e_n. Solving the differential equation satisfied by $E(z)$, we obtain the following as $z \to 1$

$$E(z) \sim 1 + (1 - p_{\mathrm{I}})^{2/p_{\mathrm{I}} - 1} K(z)^{-1} \left(2 + \int_0^z G(w)K(w)dw\right) \times (1-z)^{-2},$$

where $G(z) \to G(1) > 0$ as $z \to 1$ and $K(z) := \exp\left(2p_\circledast \int_0^z \frac{A(w)}{1 - p_{\mathrm{I}} w} dw\right)$.

Then, to obtain the asymptotic estimates we need for applying the Transfer Theorem to $E(z)$, we have to study $K(z)$ and the integral $\int_0^z G(w)K(w)dw$. We remark that the behaviour of the latter is determined roughly by the behaviour of $\int_0^z K(w)dw$. Indeed, if one integral converges, so does the other, and similarly for the divergence. Moreover, when the integral diverges as $z \to 1$ we also have $\int_0^z G(w)K(w)dw \sim G(1) \int_0^z K(w)dw$.

The asymptotics for $K(z)$ are obtained by the singular integration (see [9, Theorem VI.9]) of the asymptotics of $A(z)$.

▶ **Example 13.** Consider the case $p_\circledast > \frac{1}{2}$. Proposition 9 tells us that $A(z) = \frac{\gamma_\infty}{1-z} + O((1-z)^{2p_\circledast - 2})$. Thus we also have $\frac{A(w)}{1 - p_{\mathrm{I}} w} = \frac{\gamma_\infty/(1-p_{\mathrm{I}})}{1-w} + O((1-w)^{2p_\circledast - 2})$ as $w \to 1$. Singular integration gives $2p_\circledast \int_0^z \frac{A(w)}{1 - p_{\mathrm{I}} w} dw = \frac{2p_\circledast \gamma_\infty}{1 - p_{\mathrm{I}}} \log\left(\frac{1}{1-z}\right) + c_0 + O((1-z)^{2p_\circledast - 1})$ for a certain constant c_0. As the remainder O-term tends to 0, we conclude $K(z) \sim C_K \times (1-z)^{-2p_\circledast \frac{\gamma_\infty}{1 - p_{\mathrm{I}}}}$. We remark that $\frac{2p_\circledast \gamma_\infty}{1 - p_{\mathrm{I}}} = \frac{4p_\circledast - 2}{1 - p_{\mathrm{I}}}$.

Singular integration yields the following estimates for $J(z) := \int_0^z G(w)K(w)dw$:

▶ **Lemma 14.** *The function $J(z)$ satisfies the following asymptotics as $z \to 1$ on Ω:*

- *if $4p_{\circledast} > 3 - p_{\mathrm{I}}$, then $J(z) \sim C_J \times (1-z)^{1-\frac{4p_{\circledast}-2}{1-p_{\mathrm{I}}}}$, with $C_J > 0$*
- *if $4p_{\circledast} = 3 - p_{\mathrm{I}}$ then $J(z) \sim C_J \times \log\left(\frac{1}{1-z}\right)$, with $C_J > 0$*
- *if $4p_{\circledast} < 3 - p_{\mathrm{I}}$ then $J(z) \sim C_J$, with $2 + C_J > 0$*

where C_J is a constant depending on $p_{\mathrm{I}}, p_{\circledast}, s$.

The proof of this lemma proceeds by discussing whether the integral $J(z)$ is convergent or divergent. Notice for example that $\int_0^z K(w)dw$ diverges for $4p_{\circledast} - 2 \geq 1 - p_{\mathrm{I}}$ due to the estimate given at the end of Example 13. This gives the second threshold $p_{\circledast} = (3 - p_{\mathrm{I}})/4$.

This new threshold $p_{\circledast} = (3 - p_{\mathrm{I}})/4$, along with the previous $p_{\circledast} = \frac{1}{2}$ for the behaviour of $A(z)$, determine the 5 cases in the discussion in Theorem 12.

6 Conclusion

In this article we have seen that random BST-like tree-expressions have a rich range of behaviors with respect to the simple reduction linked to an absorbing pattern. This situation contrasts with the case of uniform random tree-expressions [11, 12] where it was previously shown that the expected value of the size after reduction is $O(1)$.

From a theoretical point of view, the existence of two thresholds is interesting in itself, this leads to a variety of different regimes for the simplifications using a simple rule. There are natural extensions of this paper to widen the result:

- Refine the result for small patterns: this will only improve the multiplicative constants of Theorem 12, not change the order of magnitude of the size of the reduced tree.
- Allow for operators of arity more than 2, as BST-like distribution can naturally be extended to handle such operators. This introduces technical difficulties, but our first attempts at addressing this extension indicate similar results (with different thresholds).
- Allow for more involved specifications, using grammar-like rules. This can be used, for instance, to prevent two consecutive stars in a regular expression. Such specifications were studied for the uniform distribution [12], and require dealing with system of equations instead of just one equation.

However, going back to our initial motivation of analyzing the soundness of random benchmarking, the main continuation of this work would be to mix several simplification procedures. The first step would be to allow several absorbing patterns for different operators together (this was done for a very specific distribution on $\wedge/\vee$-formulas in [3]). Going even further, we could focus on the simplification procedure of an existing tool and extensively study it using the techniques we developed in this article, for instance, tools like `Spot` for random LTL-formulas (see Algorithm 1).

To conclude, the message of this paper is that, contrarily to the uniform distribution, a BST-like distribution might be a relevant way to sample interesting random expressions in a practical framework. However, one should be very careful when tuning the parameters, i.e. the probability of the operators, as it may quickly lead to a degenerated case.

References

1 T.M. Apostol. *Calculus. Vol. II: Multi-variable Calculus and Linear Algebra, with Applications to Differential Equations and Probability.* Blaisdell international textbook series. Xerox College Publ., 1969.

2 Nicolas Broutin and Cécile Mailler. And/or trees: A local limit point of view. *Random Struct. Algorithms*, 53(1):15–58, 2018. doi:10.1002/rsa.20758.

3 Brigitte Chauvin, Danièle Gardy, and Cécile Mailler. The growing tree distribution on boolean functions. In *2011 Proceedings of the Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 45–56, 2011. doi:10.1137/1.9781611973013.5.

4 Marco Daniele, Fausto Giunchiglia, and Moshe Y. Vardi. Improved automata generation for linear temporal logic. In Nicolas Halbwachs and Doron Peled, editors, *Computer Aided Verification*, pages 249–260, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

5 Luc Devroye. A note on the height of binary search trees. *J. ACM*, 33(3):489–498, May 1986. doi:10.1145/5925.5930.

6 Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 – a framework for LTL and ω-automata manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*, volume 9938 of *Lecture Notes in Computer Science*, pages 122–129. Springer, October 2016. doi:10.1007/978-3-319-46520-3_8.

7 Philippe Flajolet, Xavier Gourdon, and Conrado Martinez. Patterns in random binary search trees. *Random Struct. Algorithms*, 11(3):223–244, 1997. doi:10.1002/(SICI)1098-2418(199710)11:3<223::AID-RSA2>3.0.CO;2-2.

8 Philippe Flajolet and Andrew M. Odlyzko. The average height of binary trees and other simple trees. *J. Comput. Syst. Sci.*, 25(2):171–213, 1982. doi:10.1016/0022-0000(82)90004-6.

9 Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.

10 Antoine Genitrini, Bernhard Gittenberger, Veronika Kraus, and Cécile Mailler. Associative and commutative tree representations for boolean functions. *Theoretical Computer Science*, 570:70–101, 2015. doi:10.1016/j.tcs.2014.12.025.

11 Florent Koechlin, Cyril Nicaud, and Pablo Rotondo. Uniform random expressions lack expressivity. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*, pages 51:1–51:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

12 Florent Koechlin, Cyril Nicaud, and Pablo Rotondo. On the degeneracy of random expressions specified by systems of combinatorial equations. In Natasa Jonoska and Dmytro Savchuk, editors, *Developments in Language Theory - 24th International Conference, DLT 2020, Tampa, FL, USA, May 11-15, 2020, Proceedings*, volume 12086 of *Lecture Notes in Computer Science*, pages 164–177. Springer, 2020.

13 Cyril Nicaud. On the Average Size of Glushkov's Automata. In Adrian-Horia Dediu, Armand-Mihai Ionescu, and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications, Third International Conference, LATA 2009, Tarragona, Spain, April 2-8, 2009. Proceedings*, volume 5457 of *Lecture Notes in Computer Science*, pages 626–637. Springer, 2009.

14 Cyril Nicaud, Carine Pivoteau, and Benoît Razet. Average Analysis of Glushkov Automata under a BST-Like Model. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 388–399, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSTTCS.2010.388.

15 Carine Pivoteau, Bruno Salvy, and Michèle Soria. Algorithms for combinatorial structures: Well-founded systems and newton iterations. *J. Comb. Theory, Ser. A*, 119(8):1711–1773, 2012. doi:10.1016/j.jcta.2012.05.007.

16 Richard P. Stanley. Differentiably finite power series. *Eur. J. Comb.*, 1(2):175–188, 1980. doi:10.1016/S0195-6698(80)80051-5.

17 Heikki Tauriainen. Automated testing of Büchi automata translators for linear temporal logic. Research Report A66, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, December 2000.

18 W. Wasow. *Asymptotic Expansions for Ordinary Differential Equations*. Dover Books on Mathematics. Dover Publications, 2018. URL: https://books.google.fr/books?id=NQNKDwAAQBAJ.

Cluster Editing Parameterized Above Modification-Disjoint P_3-Packings

Shaohua Li
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Marcin Pilipczuk
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Manuel Sorge
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland
Institute of Logic and Computation, TU Wien, Austria

Abstract

Given a graph $G = (V, E)$ and an integer k, the Cluster Editing problem asks whether we can transform G into a union of vertex-disjoint cliques by at most k modifications (edge deletions or insertions). In this paper, we study the following variant of Cluster Editing. We are given a graph $G = (V, E)$, a packing $\mathcal{H}$ of modification-disjoint induced P_3s (no pair of P_3s in $\mathcal{H}$ share an edge or non-edge) and an integer ℓ. The task is to decide whether G can be transformed into a union of vertex-disjoint cliques by at most $\ell + |\mathcal{H}|$ modifications (edge deletions or insertions). We show that this problem is NP-hard even when $\ell = 0$ (in which case the problem asks to turn G into a disjoint union of cliques by performing exactly one edge deletion or insertion per element of $\mathcal{H}$) and when each vertex is in at most 23 P_3s of the packing. This answers negatively a question of van Bevern, Froese, and Komusiewicz (CSR 2016, ToCS 2018), repeated by C. Komusiewicz at Shonan meeting no. 144 in March 2019. We then initiate the study to find the largest integer c such that the problem remains tractable when restricting to packings such that each vertex is in at most c packed P_3s. Van Bevern et al. showed that the case $c = 1$ is fixed-parameter tractable with respect to ℓ and we show that the case $c = 2$ is solvable in $|V|^{2\ell+O(1)}$ time.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Graph algorithms, fixed-parameter tractability, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.49

Related Version *Full Version*: https://arxiv.org/abs/1910.08517 [37]

Funding This research is part a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement 714704.

1 Introduction

Correlation Clustering is a well-known problem motivated by research in computational biology [6] and machine learning [5]. In this problem we aim to partition data points into groups or clusters according to their pairwise similarity and this has been intensively studied in the literature, see [1, 3, 4, 5, 6, 15], for example.

In this paper, we study Correlation Clustering from a graph-based point of view, resulting in the following problem formulation. A graph H is called a *cluster graph* if H is a union of vertex-disjoint cliques; we also call these cliques *clusters*. Given a graph $G = (V, E)$, in the optimization version of Cluster Editing we ask for a minimum-size *cluster-editing set* S, that is, a set $S \subseteq \binom{V}{2}$ of vertex pairs such that $G \triangle S := (V, E \triangle S)$ is a cluster graph. Here $E \triangle S$ is the symmetric difference of E and S, that is, $E \triangle S = (E \setminus S) \cup (S \setminus E)$. We also sometimes refer to vertex pairs as *edits*. Cluster Editing is NP-hard [43]. Constant-ratio

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 49; pp. 49:1–49:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

approximation algorithms have been found for the optimization variant [1, 5, 15] but it is also APX-hard [15]. We focus here on exact algorithms and the decision version of Cluster Editing.

Given a natural number k and a graph $G = (V, E)$, the decision version of Cluster Editing asks whether there exists a cluster-editing set S such that $|S| \leq k$. Exact parameterized algorithms for Cluster Editing and some of its variants have been extensively studied [28, 7, 42, 19, 12, 31, 10, 11, 21, 30, 8, 35, 24, 40, 13]. Cluster Editing is but one of a large group of edge modification problems that have been studied, see Crespelle et al. [17] for a recent survey. Perhaps it is one of the most important such problems because of the practical motivation. Barring few exceptions [19, 35, 24, 44], Cluster Editing has mainly been studied with respect to the solution-size parameter k. It is not hard to observe that Cluster Editing is fixed-parameter tractable with respect to k and a series of papers [28, 27, 7, 11, 8] continually improved the base in the exponential part of the running time, culminating in the current fastest fixed-parameter algorithm with running time $O(1.62^k + n + m)$ [8], where n is the number of vertices of the input graph and m its number of edges. Similarly, a series of papers [28, 20, 22, 29, 42, 14, 16] gave more and more effective problem kernels[1] until a problem kernel with $2k$ vertices was achieved [14, 16].

As mentioned, the interest in Cluster Editing is not merely theoretical. Indeed, parameterized techniques are implemented in standard clustering tools [41, 45]. Although practitioners report that in particular the parameterized data-reduction techniques are effective [10, 9], the parameter k is not very small in several real-world data sets [7, 10, 44]. For instance, Böcker et al. [7, Table 2] considered 26 graphs derived from biological data with 91 to 100 vertices on which the average number of needed edits is 315, despite noting that the Cluster Editing model outperformed other clustering models.

A technique to deal with such large parameters is *parameterization above lower bounds*. Herein, the parameter is of the form $\ell = k - h$ where h is a lower bound on the solution size, usually computable in polynomial time, and ℓ is the *excess* of the solution size above the lower bound. Research into parameterizations above lower bounds has been active and fruitful for several parameterized problems, not only on the theory-side (see [39, 18, 26, 38, 36], for example) but also in practice, as algorithms based on that approach yielded quite efficient implementations for Vertex Cover [2] and among the most efficient ones for Feedback Vertex Set [32, 34]. For Cluster Editing we are aware of only one research work considering parameterizations above lower bounds: Van Bevern, Froese, and Komusiewicz [44] studied edge-modification problems parameterized above the lower bound from packings of forbidden induced subgraphs and showed that Cluster Editing parameterized by the excess above the size of a given packing of *vertex-disjoint* P_3s is fixed-parameter tractable. Observe that a graph is a cluster graph if and only if it does not contain any P_3, a path on three vertices, as an induced subgraph. Consequently, one needs to perform at least one edge deletion or insertion per element of the packing.

As the P_3s in the above packing are vertex-disjoint, the value by which the packing can decrease the parameter is limited. In the previous example with 315 edits, subtracting the resulting lower bound would reduce the parameter by at most 33. In their conclusion, van Bevern et al. [44] asked whether Cluster Editing is fixed-parameter tractable when parameterized above a stronger lower bound, the size of a modification-disjoint packing

[1] A problem kernel is a formalization of provably effective and efficient data reduction. It is a polynomial-time self-reduction that produces instances of size bounded by some function of the parameter.

of P_3s. Here, a packing $\mathcal{H}$ of induced P_3s in G is *modification-disjoint* if every two P_3s in $\mathcal{H}$ do not share edges or non-edges, that is, they share at most one vertex. The formal problem definition is as follows.

> Cluster Editing above modification-disjoint P_3 packing (CEaMP)
> **Input:** A graph $G = (V, E)$, a modification-disjoint packing $\mathcal{H}$ of induced P_3s of G, and a non-negative integer ℓ.
> **Question:** Is there a cluster editing set, i.e. a set of vertex pairs $S \subseteq \binom{V}{2}$ so that $G \triangle S$ is a union of disjoint cliques, with $|S| - |\mathcal{H}| \leq \ell$?

We also say that a set S as above is a *solution*.

Our results. At Shonan Meeting no. 144 [33] Christian Komusiewicz re-iterated the question of van Bevern et al. [44] and it was also asked in Vincent Froese's dissertation [25]. In this paper, we answer this question negatively by showing the following.

▶ **Theorem 1.** *Cluster Editing above modification-disjoint P_3 packing is NP-hard even for $\ell = 0$ and when each vertex in the input graph is incident with at most 23 P_3s of $\mathcal{H}$.*

In other words, given a graph G and a packing $\mathcal{H}$ of modification-disjoint P_3s in G, it is NP-hard to decide if one can delete or insert exactly one edge per element of $\mathcal{H}$ to obtain a cluster graph. Proving Theorem 1 was surprisingly nontrivial. A straightforward approach would be to amend the known reductions [35, 23] that show NP-hardness for constant maximum vertex degree by specifying a suitable packing of P_3s. However, an argument based on the linear-programming relaxation of packing modification-disjoint P_3s shows that the graphs produced by these reductions do not admit tight P_3 packing bounds. We did not find a way around this issue and thus developed a novel reduction based on new gadgets.

The verdict spelt by Theorem 1 is unfortunately quite damning. It indicates that even just reaching the lower bound given by a modification-disjoint P_3 packing already captures the algorithmic hardness of the problem. However, there may be a way out of this conundrum: Call a modification-disjoint P_3 packing *$1/c$-integral* if each vertex is in at most c packed P_3s (and say *integral* in place of *1-integral* and *half-integral* in place of *$1/2$-integral*). As the case $c = 1$ is just the case of vertex-disjoint packings, van Bevern et al. [44] showed that Cluster Editing parameterized by the excess over integral P_3 packings is fixed-parameter tractable. Thus it becomes an intriguing question to find the largest $c < 23$ such that CEaMP remains tractable with respect to the excess over $1/c$-integral packings. We provide progress towards answering this question here. The problem Cluster Editing above half-integral P_3 packing (CEaHMP) is defined in the same way as CEaMP except that the input packing $\mathcal{H}$ is half-integral. It turns out that the complexity of the problem indeed drops when making the packing half-integral:

▶ **Theorem 2.** *Cluster Editing above half-integral P_3 packing parameterized by the number ℓ of excess edits is in XP. It can be solved in $O(n^{2\ell+O(1)})$ time, where n is the number of vertices in the input graph.*

A straightforward idea to prove Theorem 2 would be to adapt the fixed-parameter algorithm for vertex-disjoint packings given by van Bevern et al. [44]. Their main idea is to show that if a packed P_3 P of the input graph G admits a solution that is optimal for P and that respects certain conditions on the neighborhood of $V(P)$ in G then this solution can be used in an optimal cluster-editing set for G. Afterwards, each packed P_3 P either needs an excess edit in $V(P)$ or an edit incident with $V(P)$ in G. Since the P_3s in the packing are vertex-disjoint an edit incident with $V(P)$ will be in excess over the packing lower bound as well. It then follows that the overall number of edits is bounded by a function of the excess edits.

Unfortunately, the above idea fails for modification-disjoint packings for two reasons. First, the property that packed P_3s have an edit incident with them is not helpful anymore, because these edits may be part of other packed P_3s and hence not be in excess. Second, if we would like to preserve that these edits are excess, we need to check the special neighborhood properties of van Bevern et al. [44] for arbitrarily large connected components of packed P_3s efficiently. We did not see a way around these issues and instead designed an algorithm from scratch: A straightforward guessing of the excess edits reduces the problem to the case where we need to check for zero excess edits. This case is then solved by an extensive set of reduction rules that exploit the structure given by the half-integral packing. Essentially, we successively reduce the maximum size of clusters in the final cluster graph. This then allows us to reduce the problem to Cluster Deletion. Together with the properties of the packing, this problem allows a formulation as a 2SAT formula which we then solve in polynomial time.

Organization. After brief preliminaries in Section 2, we give an outline of the reduction used to show Theorem 1 in Section 3. Section 4 then contains an outline of the proof of Theorem 2. Due to space constraints, parts of the constructions and algorithms are deferred to a full version of the paper [37].

2 Preliminaries

In this paper, we denote an undirected graph by $G = (V, E)$, where $V = V(G)$ is the set of vertices, $E = E(G)$ is the set of edges, and $\binom{V}{2} \setminus E$ is the set of *non-edges*. An undirected edge between two vertices u and v will be denoted by uv where we put $uv = vu$. An undirected non-edge between two vertices x and y will be denoted by xy, where we put $xy = yx$, and we will explicitly mention that xy is a non-edge in case of confusion with the notation of an edge. If uv is an edge in the graph, we say u and v are *adjacent*. We denote a bipartite graph by $B = (U, W, E)$, where U, W are the two parts of the vertex set of B and E is the set of edges of B. We say that a bipartite graph is *complete* if for every pair of vertices $u \in U$ and $w \in W$, $uw \in E$. For a non-empty subset of vertices $X \subseteq V$, we denote the subgraph induced by X by $G[X]$. A *clique* Q in a graph G is a subgraph of G in which any two distinct vertices are adjacent. A *cluster graph* is a graph in which every connected component is a clique. A connected component in a cluster graph is called a *cluster*.

Let G' be a cluster graph and let S be a cluster editing set S such that $G \triangle S = G'$. We say that two cliques Q_1 and Q_2 of G are *merged* (in G') if they belong to the same cluster in G'. We say that Q_1 and Q_2 are *separated* (in G') if they belong to two different clusters in G'. When mentioning the edges or non-edges between the vertices of the clique Q_1 and the vertices of the clique Q_2, we refer to the edges or non-edges between the clique Q_1 and the clique Q_2 for short. Let $\ell, r \in \mathbb{N}$. We denote a path with ℓ vertices by P_ℓ and a cycle with r vertices by C_r.

Let x, y, z be vertices in a graph G. We say that xyz is an induced P_3 of G if $xy, yz \in E(G)$ and $xz \notin E(G)$. Vertex y is called the *center* of xyz. We say that vertices x, y, z *belong to* xyz or x, y, z are *incident with* xyz. We also say that xyz is *incident with* the vertices x, y and z. In this paper, all P_3s we mention are induced P_3s; we sometimes skip the qualifier "induced" for convenience.

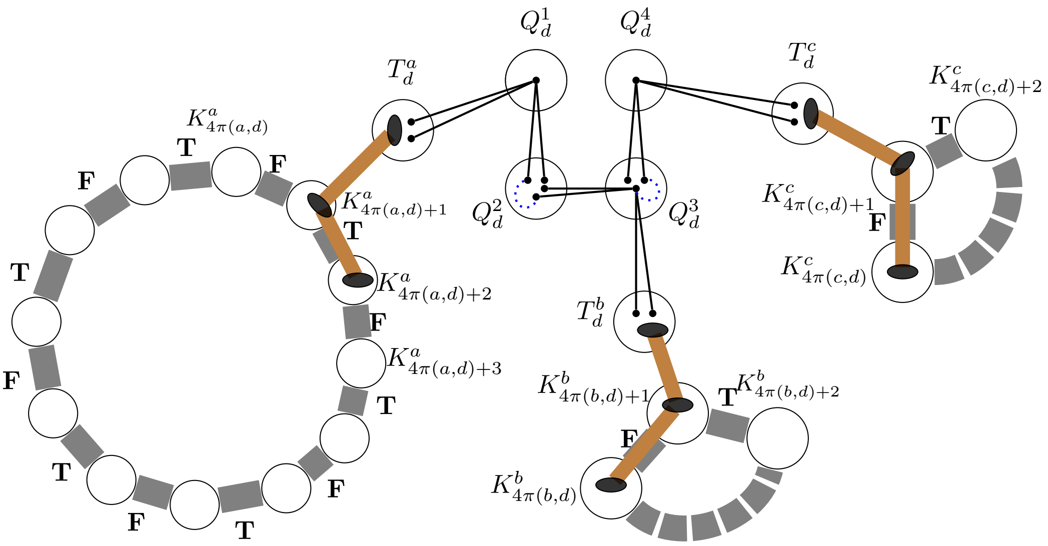


Figure 1 Basic structure of graph constructed from a 3-CNF formula. Depicted is a clause gadget for a clause $x_a \vee \neg x_b \vee \neg x_c$ and the variable gadgets for x_a, x_b, and x_c.

Given an instance $(G, \mathcal{H}, \ell)$ of CEAMP, if xyz is a P_3 in G and $xyz \in \mathcal{H}$, we say that xyz is *packed*, and we say that the edges xy, yz are *covered* by xyz and the non-edge xz is covered by xyz. If an edge xy is covered by some P_3 of $\mathcal{H}$, we say that xy is a *packed* edge. Otherwise we say that xy is a *non-packed* edge. If a non-edge uv is covered by some P_3 of $\mathcal{H}$, we say that uv is a *packed* non-edge. Otherwise we say that uv is a *non-packed* non-edge. If xyz is a P_3 in G and Q_1, Q_2, and Q_3 are pair-wise non-intersecting vertex sets of G, we say that *xyz connects Q_1 and Q_3 via Q_2* if the center y of xyz belongs to Q_2 and x, z belong to Q_1 and Q_3, respectively.

We sometimes need finite fields of prime order. Let p be some prime. By $\mathbb{F}_p$ we denote the finite field with the p elements $0, \ldots, p-1$ with addition and multiplication modulo p.

3 NP-hardness for tight modification-disjoint packings

Overview. In this section, we outline the proof of Theorem 1 that shows a reduction from the NP-hard problem of deciding satisfiability of 3-CNF formulas. Given a 3-CNF formula Φ with variables $x_0, \ldots, x_{n-1}$ and clauses $\Gamma_0, \ldots, \Gamma_{m-1}$ we construct a graph $G = (V, E)$ with a modification-disjoint packing $\mathcal{H}$ of induced P_3s such that Φ has a satisfying assignment if and only if G has a cluster editing set S which consists of exactly one edit in each P_3 in $\mathcal{H}$.

Let us start with Figure 1 which depicts the basic structure of the graph G. The fundamental building blocks of G and $\mathcal{H}$ are what we call proto-clusters, indicated by white circles. A *proto-cluster* is an induced subgraph H of G whose vertex set is maximal with respect to the property that H contains a spanning tree that consists entirely of non-packed edges. Note that the set of proto-clusters partitions the vertex set of G. As we cannot edit non-packed edges, the clusters in each solution that we may obtain induce a partition that is coarser than the partition given by the proto-clusters.

Our first concern is to interconnect the proto-clusters in such a way that a grouping into solution clusters implies a satisfying assignment of Φ – the construction is *sound*. To this end, a straightforward idea of modeling the truth-value of a variable comes to mind: Use an even-length cycle of proto-clusters and add P_3s to the packing $\mathcal{H}$ such that either the odd pairs or the even pairs of proto-clusters on the cycle need to be merged into clusters. The variable gadgets are represented by the three gray cycles in Figure 1. A clause gadget is slightly less obvious, because we need a three-way choice and straightforward constructions yield only two-way choices. A solution is shown in Figure 1: There are four proto-clusters, Q_d^1 through Q_d^4 such that there is a non-packed nonedge between Q_d^1 and Q_d^4 and a path in G from Q_d^1 over Q_d^2 and Q_d^3 to Q_d^4. Because of the non-packed nonedge, the proto-clusters Q_d^1 and Q_d^4 are in different solution clusters. Hence, we need to separate a pair of Q_d^i and Q_d^{i+1} for some $i \in [3]$. This models the choice of the variable that shall satisfy the clause. This choice is then transferred to the variable gadgets by suitable packed P_3s and further proto-clusters. A nontrivial issue in this transfer of choices is how to connect variable gadgets to the rest of the construction. On the one hand, we need to pack P_3s that are partly in the variable gadgets and partly outside so as to transfer the choice and on the other hand, we need a packing of P_3s inside the variable gadgets in order to allow both the merging of odd pairs and even pairs of proto-clusters in the gadget.

The most involved part of the construction is indeed how to ensure the *completeness*, that is, the property that a satisfying assignment for Φ gives a cluster-editing set with zero excess edits for G. This issue makes the construction that we obtain somewhat special: We need to pack P_3s into the above "skeleton" construction so as to allow for the merging and cutting of pairs of proto-cluster according to the satisfying assignment. We accomplish this by a careful implementation of the above gadgets such that the edges and non-edges that are covered by the packed P_3s of the skeleton construction have a special structure. We then use an algebraic construction that allows us to prove that the needed covering of the remaining edges and non-edges by modification-disjoint P_3s exists.

We now proceed to describing the variable and clause gadgets more formally and then show how we have resolved the above two issues.

Variable gadgets. As mentioned, a variable will be represented by a cycle of proto-clusters such that any solution needs to merge either each odd or each even pair of consecutive proto-clusters. These two options represent the truth value assigned to the variable. In order to enable both associated solutions with zero edits above the packing lower bound, we build an associated packing of P_3s such that all vertex pairs between consecutive proto-clusters are covered by a P_3 in the packing. Since we later on need to connect the variable gadgets to the clause gadgets, each proto-cluster will contain five vertices, giving us enough attachment points for later.

Let m_i denote the number of clauses that contain the variable x_i, $i = 0, 1, \ldots, n-1$. For each variable x_i, $i = 0, 1, \ldots, n-1$, we create $4m_i$ vertex-disjoint cliques with 5 vertices each, namely $K_0^i, \ldots, K_{4m_i-1}^i$. In each K_j^i, $j = 0, 1, \ldots, 4m_i - 1$, the vertices are $v_{j,0}^i, \ldots, v_{j,4}^i$. For each $j = 0, 2, \ldots, 4m_i - 2$, we create P_3s connecting K_j^i, K_{j+1}^i, and K_{j+2}^i as follows (here we identify K_0^i as $K_{4m_i}^i$).

We add pairwise modification-disjoint P_3s to cover all edges between the cliques K_j^i we have just introduced. Recall that $\mathbb{F}_5$ is the finite field of the integers modulo 5. We take three consecutive cliques and add P_3s with one vertex in each of the three cliques. To do this without overlapping two P_3s, we think about the cliques' vertices as elements of $\mathbb{F}_5$ and add a P_3 for each possible arithmetic progression. That is, in each added P_3 the difference of the first two elements of the P_3 is equal to the difference of the second two elements. In this way, each vertex pair is contained in a single P_3 since the third element is uniquely defined.

Formally, for every triple of elements $p, q, r \in \mathbb{F}_5$ satisfying the equality $q - p = r - q$ over $\mathbb{F}_5$, we add to the graph the edges $v^i_{j,p}v^i_{j+1,q}$ and $v^i_{j+1,q}v^i_{j+2,r}$ and to the packing $\mathcal{H}$ the P_3 given by $v^i_{j,p}v^i_{j+1,q}v^i_{j+2,r}$. Note that in this manner the clique K^i_{j+1} becomes fully adjacent to K^i_j and to K^i_{j+2} while K^i_{j+1} stays anti-adjacent to all other cliques $K^i_{j'}$.

Observe that the P_3s given by $v^i_{j,p}v^i_{j+1,q}v^i_{j+2,r}$ for $j = 0, 2, \ldots, 4m_i - 2$ such that $q - p = r - q$ are pairwise modification-disjoint: For each $j = 0, 2, \ldots, 4m_i - 2$, an arbitrary edge just introduced between K^i_j and K^i_{j+1} has the form $\{v^i_{j,p}, v^i_{j+1,q}\}$ for some $p, q \in \mathbb{F}_5$. It belongs to the unique P_3 given by $v^i_{j,p}v^i_{j+1,q}v^i_{j+2,r}$, where $r = 2q - p$. Similarly, an arbitrary edge $\{v^i_{j+1,q}, v^i_{j+2,r}\}$ for $q, r \in \mathbb{F}_5$ belongs to the unique P_3 given by $v^i_{j,2q-r}v^i_{j+1,q}v^i_{j+2,r}$ and an arbitrary non-edge $\{v^i_{j,p}, v^i_{j+2,r}\}$ for $p, r \in \mathbb{F}_5$ belongs to the unique P_3 given by $v^i_{j,p}v^i_{j+1,(p+r)\cdot 2^{-1}}v^i_{j+2,r}$, where 2^{-1} is the multiplicative inverse of 2 over $\mathbb{F}_5$, that is, $2^{-1} = 3$.

After this construction, we set the P_3 packing of the variable gadgets to

$$\mathcal{H}_{\mathsf{var}} = \{v^i_{j,p}v^i_{j+1,q}v^i_{j+2,r} \mid i = 0, \ldots, n-1;\ j = 0, 2, \ldots, 4m_i - 2;\ p, q, r \in \mathbb{F}_5; \text{ and } q - p = r - q\}.$$

This finishes the first stage of the construction. The truth values of the variable are represented as follows. For every variable x_i, $i = 0, \ldots, n-1$, if K^i_j and K^i_{j+1} are merged for $j = 0, \ldots, 4m_i - 2$, then this represents assigning false to the variable x_i. If K^i_{j+1} and K^i_{j+2} are merged for $j = 0, \ldots, 4m_i - 2$, then this represents variable x_i being true. We will make minor modifications to the variable gadgets and $\mathcal{H}_{\mathsf{var}}$ below so as to transmit the choice of truth value to the clause gadgets.

Skeleton of the clause gadget. In order to introduce the construction of the clause gadget, we first give a description of the skeleton of the clause gadget. The skeleton, depicted in Figure 1, is a subgraph of the final construction that allows us to prove the soundness. The construction is finalized later.

For each variable x_i, $i = 0, 1, \ldots, n-1$, of Φ we fix an arbitrary ordering of the clauses that contain x_i. If a clause Γ_j contains x_i, let $\pi(i, j) \in \{0, \ldots, m_i - 1\}$ denote the position of the clause Γ_j in this ordering. Let initially $\mathcal{H}_{\mathsf{tra}} = \emptyset$. For each clause Γ_d $(d = 0, \ldots, m-1)$ proceed as follows. We first introduce four cliques Q^1_d, Q^2_d, Q^3_d and Q^4_d. Let Γ_d contain the variables x_a, x_b and x_c. We introduce the cliques T^a_d, T^b_d and T^c_d, called *transferring cliques*. All of the cliques introduced are pairwise vertex disjoint and can be of different sizes. The concrete size will be determined later. Next, we introduce the following P_3s into G and $\mathcal{H}_{\mathsf{tra}}$ (see the center of Figure 1):

- P^1_d and P^2_d that both connect T^a_d and Q^2_d via Q^1_d and that share a vertex in Q^1_d.
- P^3_d and P^4_d that both connect T^b_d and Q^2_d via Q^3_d and that share a vertex in Q^3_d.
- P^5_d and P^6_d that both connect T^c_d and Q^3_d via Q^4_d and that share a vertex in Q^4_d.

All the P_3s above are pairwise vertex-disjoint except for the shared vertices explicitly mentioned in the definition. We call the P_3s of $\mathcal{H}_{\mathsf{tra}}$ *transferring P_3s*.

Connection to the variable gadgets. Next we connect the transferring cliques T^a_d, T^b_d, and T^c_d to the variable gadgets of x_a, x_b, and x_c, respectively. To avoid additional notation, we only explain the procedure for T^a_d and x_a, the other pairs are connected analogously. We connect T^a_d to the variable gadget of x_a by a set of four modification-disjoint P_3s as shown in Figure 2 and explained formally below. The centers of these P_3s are in $K^a_{4\pi(a,d)+1}$. For each of these four P_3s, exactly one endpoint is an arbitrary distinct vertex in T^a_d which is different from the endpoints of the P_3s connecting T^a_d to Q^1_d; we denote these endpoints as w_1, w_2, w_3, w_4. The other endpoint is in $K^a_{4\pi(a,d)+2}$ if x_a appears positively in Γ_d and the other endpoint is in $K^a_{4\pi(a,d)}$ otherwise. The precise centers and endpoints in the cliques

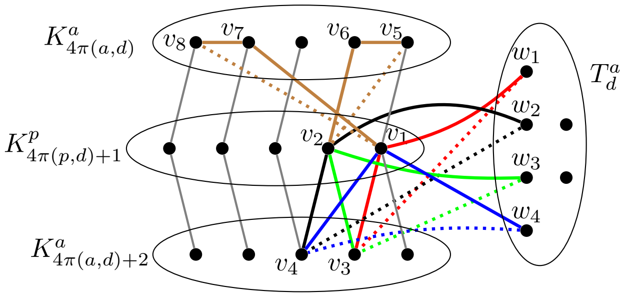


Figure 2 Connection of a clause gadget with a variable gadget for a variable x_a which appears positively in the clause. White ellipses represent cliques. The vertices in the cliques in the variable gadget are ordered from right to left according to the elements of $\mathbb{F}_5$ which they represent. For example, the rightmost vertex in $K^a_{4\pi(a,d)}$ is $v^a_{4\pi(a,d),0}$ (corresponding to $0 \in \mathbb{F}_5$) and the leftmost is $v^a_{4\pi(a,d),4}$ (corresponding to $4 \in \mathbb{F}_5$). The gray lines adjacent to cliques in the variable gadget represent some of the P_3s that were introduced into the variable gadgets in the beginning. In colors red, black, green, and blue we show the P_3s that connect the transferring clique T^a_d with the variable gadget of variable x_a. Herein, dotted lines are non-edges and solid lines are edges. Note that these connecting P_3s supplant some of the edges of previously present P_3s in the variable gadget – the previously present P_3s are then removed. For example the green P_3 replaces the edge v_2v_3 of the P_3 given by $v_6v_2v_3$ that was previously present. To maintain that each vertex pair between consecutive cliques in the variable gadget is covered by some P_3 in the packing, we add the brown P_3s.

$K^a_{4\pi(a,d)+2}$ or $K^a_{4\pi(a,d)}$ are specified below. Since these newly introduced P_3s use edges that belong to some P_3s in $\mathcal{H}_{\mathsf{var}}$ that were introduced while constructing the variable gadgets, we will remove such P_3s in the variable gadget from $\mathcal{H}_{\mathsf{var}}$, remove their corresponding edges from the graph, and add some new P_3s to $\mathcal{H}_{\mathsf{var}}$ as described below. As a result, the clique $K^a_{4\pi(a,d)+1}$ may no longer be fully adjacent to $K^a_{4\pi(a,d)}$ or $K^a_{4\pi(a,d)+2}$. We will however maintain the invariant that each vertex pair between $K^a_{4\pi(a,d)+1}$ and $K^a_{4\pi(a,d)}$ or $K^a_{4\pi(a,d)+2}$ is covered by a P_3 in the packing and that all the P_3s of $\mathcal{H}_{\mathsf{var}}$ are pairwise modification-disjoint.

Formally, if x_a appears positively in Γ_d, we denote:

$$\begin{array}{llll} v_1 = v^a_{4\pi(a,d)+1,0} & v_2 = v^a_{4\pi(a,d)+1,1} & v_3 = v^a_{4\pi(a,d)+2,1} & v_4 = v^a_{4\pi(a,d)+2,2} \\ v_5 = v^a_{4\pi(a,d),0} & v_6 = v^a_{4\pi(a,d),1} & v_7 = v^a_{4\pi(a,d),3} & v_8 = v^a_{4\pi(a,d),4}. \end{array}$$

If x_a appears negatively in Γ_d, we swap the roles of $K^a_{4\pi(a,d)}$ and $K^a_{4\pi(a,d)+2}$, that is:

$$\begin{array}{llll} v_1 = v^a_{4\pi(a,d)+1,0} & v_2 = v^a_{4\pi(a,d)+1,1} & v_3 = v^a_{4\pi(a,d),1} & v_4 = v^a_{4\pi(a,d),2} \\ v_5 = v^a_{4\pi(a,d)+2,0} & v_6 = v^a_{4\pi(a,d)+2,1} & v_7 = v^a_{4\pi(a,d)+2,3} & v_8 = v^a_{4\pi(a,d)+2,4}. \end{array}$$

As shown in Figure 2, we remove the P_3s given by $v_8v_1v_3$, $v_7v_1v_4$, $v_6v_2v_3$, and $v_5v_2v_4$ from $\mathcal{H}_{\mathsf{var}}$ and we remove their corresponding edges from the graph. Then we add the P_3s given by $v_5v_6v_2$ and $v_1v_7v_8$ to the graph and to $\mathcal{H}_{\mathsf{var}}$. Finally, we connect T^a_d via $K^a_{4\pi(a,d)+1}$ by adding the P_3s given by $w_1v_1v_3$, $w_2v_2v_4$, $w_3v_2v_3$, and $w_4v_1v_4$ to the graph and to $\mathcal{H}_{\mathsf{tra}}$. Note that, indeed, each vertex pair between $K^a_{4\pi(a,d)+1}$ and $K^a_{4\pi(a,d)}$ and between $K^a_{4\pi(a,d)+1}$ and $K^a_{4\pi(a,d)+2}$ remains covered by a P_3 in the packing after replacing all P_3s. This finishes the construction of the skeleton of the clause gadgets.

Intuitively, the skeleton ensures the soundness as follows. Recall from above that we need to delete at least one of three sets of edges in the solution, namely the edges between Q^1_d and Q^2_d, the edges between Q^2_d and Q^3_d, or the edges between Q^3_d and Q^4_d. Assume that the edges between Q_d^1 and Q_d^2 are deleted and the variable x_a appears positively in the clause Γ_d as in Figure 1. Because of the constraints imposed by the P_3s P_d^1 and P_d^2, cliques T_d^a and Q_d^1 have to be merged in the final cluster graph. Since $K^a_{4\pi(a,d)+1}$ cannot be merged with Q_d^1 (there are no edges between Q_d^1 and $K^a_{4\pi(a,d)+1}$, and no P_3s connecting Q_d^1 and $K^a_{4\pi(a,d)+1}$), we have to separate T_d^a from $K^a_{4\pi(a,d)+1}$. Then, the P_3s connecting T_d^a with $K^a_{4\pi(a,d)+2}$ force $K^a_{4\pi(a,d)+1}$ and $K^a_{4\pi(a,d)+2}$ to merge. This means x_a is true and it satisfies the clause Γ_d.

Merging model and P_3 padding. Above we have defined all proto-clusters of the final constructed graph. What remains is to ensure that the proto-clusters indeed can be merged as required to construct a solution from a satisfying assignment to Φ in the completeness proof. Intuitively, in this construction we have pairs of proto-clusters A and B which we would like to be able to either merge or separate without incurring excess edits. To achieve this, we add P_3s that have both an edge and a nonedge between A and B. If we are able to cover all vertex pairs between A and B with such P_3s, then merging or separating A and B will indeed not incur excess edits. The pairs of proto clusters that we want to be able to merge are captured in the *merging model*, a graph H that contains as vertices the cliques in the gadgets that we have introduced and the following edges:

- $\{\{K^i_j, K^i_{j+1}\} \mid i = 0, 1, \ldots, n-1 \text{ and } j = 0, 1, \ldots, 4m_i - 1\}$; these pairs are needed to be able to set the variable gadgets according to their truth values.
- $\{\{T^i_d, K^i_{4\pi(i,d)}\}, \{T^i_d, K^i_{4\pi(i,d)+1}\}, \{T^i_d, K^i_{4\pi(i,d)+2}\} \mid \text{variable } x_i \text{ occurs in clause } \Gamma_d\}$; these edges are needed to merge a transferring clique to its corresponding variable gadget if the variable does not satisfy the clause associated with the transferring clique.
- $\{\{Q^1_d, Q^2_d\}, \{Q^1_d, Q^3_d\}, \{Q^2_d, Q^3_d\}, \{Q^2_d, Q^4_d\}, \{Q^3_d, Q^4_d\} \mid d = 0, 1, \ldots, m-1\}$; in order to be able to merge proto-clusters in a clause gadget if they do not correspond to a variable that was chosen to satisfy the clause.
- $\{\{T^i_d, Q^k_d\} \mid \text{if variable } x_i \text{ occurs in } \Gamma_d \text{ and } T^i_d \text{ is adjacent in } G \text{ to } Q^k_d \text{ with } k \in \{1, 4\}\}$; in order to be able to merge a transferring clique to a proto-cluster in a clause gadget if the corresponding variable was chosen to satisfy the clause.
- $\{\{T^i_d, Q^3_d\}, \{T^i_d, Q^4_d\} \mid \text{if variable } x_i \text{ occurs in } \Gamma_d \text{ and } T^i_d \text{ is adjacent in } G \text{ to } Q^3_d\}$; ditto (the construction is asymmetric).

The aim is now to define a vertex partition of the merging model into levels and to pad P_3s between levels. The levels are as follows: L_0 contains all cliques in variable gadgets; L_1 contains Q^1_d and Q^4_d for each $d = 0, \ldots, m-1$; L_2 contains Q^3_d for each $d = 0, \ldots, m-1$; L_3 contains Q^2_d for each $d = 0, \ldots, m-1$; and L_4 contains all transferring cliques. Observe that apart from edges in L_0, all edges of H are between different levels. Moreover, orienting the edges in H from higher to lower level gives an acyclic orientation when ignoring the edges in level L_0 and each vertex in H on some level L_i is adjacent to only a constant number of vertices on a lower level L_j, $j < i$. Hence, we now go through the cliques in $V(H)$ in increasing order of levels and, for each clique Q, we pad P_3s between Q and its constant number of neighbors on lower levels. The padding is done so as to cover all vertex pairs between Q and the lower neighbors that are not covered by the skeleton yet. To show that such a covering exists, we need to analyze the structure of the vertex pairs that are already covered. We can show that these pairs form either P_3s (e.g. between T^a_d and Q^1_d in Figure 1) or cycles of length eight (e.g. between T^a_d and the two cliques $K^p_{4\pi(p,d)+1}$ and $K^p_{4\pi(p,d)+2}$ in Figure 2). Using this property, we can show that the desired padding of P_3s exists by proving the following result.[2] Note that the statement is about triangle packings; the triangles correspond to the vertex pairs covered by P_3s.

[2] To obtain the desired bound on the number of P_3s containing a fixed vertex we need a slightly more general result. See the details in the full version.

▶ **Lemma 3.** *Let p be a prime number with $p \geq 2$. Let $B = (V, W, E)$ be a complete bipartite graph such that $|V| = p$ and $|W| = 2p$. Let $F \subseteq E$ be a set of edges such that each connected component of $(V \cup W, F)$ is a either a P_3 with a center in V or a C_8. Then there exists an edge-disjoint triangle packing τ in $(V \cup W, E \setminus F \cup \binom{W}{2})$ which covers $E \setminus F$ such that every triangle in τ contains exactly one vertex of V, the graph $(W, \binom{W}{2} \setminus E(\bigcup \tau))$ is connected, and each vertex is in at most p triangles of τ.*

We apply Lemma 3 as follows: W is the clique for which we want to pad P_3s and V is the union of the cliques that are neighbors of W in H on lower layers. The set F contains the vertex pairs already covered by the skeleton. The packing τ corresponds to our desired P_3 packing. Finally, the connectedness property on $(W, \binom{W}{2} \setminus E(\bigcup \tau))$ ensures that W remains a proto-cluster. Using the padding we can ensure the soundness, concluding the proof of Theorem 1.

The proof of Lemma 3 works roughly as follows. We partition W into two parts W_1, W_2, each of size p. The triangles in the packing contain one vertex of each of W_1, W_2, and V; they are defined by interpreting W_1, W_2, and V each as the field F_p and taking three vertices $i \in W_1, j \in W_2, k \in V$ such that $j - i = k - j$. This defines a covering of all vertex pairs between V and W. The vertex pairs in F are avoided by covering them with specific triangles, which are then removed from the final packing.

4 XP-algorithm for half-integral packings

In this section, we study CEAMP in the special setting where every vertex is incident with at most two P_3s of the packing $\mathcal{H}$. We define this problem as Cluster Editing above half-integral P_3 packing (CEAHMP). We prove the following.

▶ **Theorem 2** (Restated). *Cluster Editing above half-integral P_3 packing parameterized by the number ℓ of excess edits is in XP. It can be solved in $O(n^{2\ell+O(1)})$ time, where n is the number of vertices in the input graph.*

The main tool in proving Theorem 2 is a polynomial-time algorithm for the case where $\ell = 0$:

▶ **Theorem 4.** *Cluster Editing above half-integral P_3 packing can be solved in polynomial time when $\ell = 0$, that is, when no excess edits are allowed.*

The proof of Theorem 4 will be given in the final part of this section. Using this we can prove Theorem 2 as follows: Essentially, the XP algorithm for Cluster Editing above half-integral P_3 packing guesses (by trying all possibilities) the number, ℓ_a, of excess edits that are not contained in any P_3 in $\mathcal{H}$ and guesses the concrete edits to be made. Then it guesses the P_3s in $\mathcal{H}$ that harbor the remaining excess edits and it guesses how these P_3s are resolved. Then it checks whether the remaining instance has a cluster-editing set without excess edits over the remaining P_3 packing $\mathcal{H}'$ using the algorithm from Theorem 4. The full proof for Theorem 2 is contained in the full version [37].

We outline the polynomial-time algorithm for CEAHMP when $\ell = 0$. It is based on a series of reduction rules that perform successive modifications to the input graph and P_3 packing to get an equivalent new instance. Whenever we state a reduction rule in the following, we assume that all the reduction rules before it have been applied exhaustively. We will omit the correctness proofs for most reduction rules and lemmas here – they are contained in the full version [37]. We first aim to decrease the maximum size of clusters in a solution cluster graph, then reduce to Cluster Deletion and then to 2SAT.

We again use the notion of proto-clusters as in the previous sections. We say a proto-cluster C is *isolated* from a proto-cluster D if there are no edges between C and D. We classify the P_3s of $\mathcal{H}$ into four types. For an induced P_3 $xyz \in \mathcal{H}$, if x, y belong to one proto-cluster and z belongs to another proto-cluster, or symmetrically y, z belong to one proto-cluster and x belongs to another proto-cluster, then xyz is a *type-α* P_3; if x, z belong to one proto-cluster and y belongs to another proto-cluster, then xyz is a *type-β* P_3; if x, y, z belong to three distinct proto-clusters, then xyz is a *type-γ* P_3; if x, y, z belong to one proto-cluster then xyz is a *type-δ* P_3.

The first five reduction rules are simple and their correctness follows almost immediately:

▶ **Reduction Rule 1.** *For any proto-cluster C, if there are two vertices $u, v \in V(C)$ such that uv is a non-packed non-edge, i.e., uv is not covered by any P_3 of $\mathcal{H}$, then return NO.*

▶ **Reduction Rule 2.** *If there is a type-β or type-δ P_3 $xyz \in \mathcal{H}$, insert the edge xz into G and remove xyz from $\mathcal{H}$.*

▶ **Reduction Rule 3.** *For any two proto-clusters A and B, if there is a non-packed non-edge uv such that $u \in V(A)$ and $v \in V(B)$, and there is a packed edge xy such that $x \in V(A)$ and $y \in V(B)$, then delete xy and remove the corresponding packed P_3 from $\mathcal{H}$.*

▶ **Reduction Rule 4.** *If there is a connected component $\mathcal{C}$ in the graph of size at most 6, then do brute force on $\mathcal{C}$ to check if there is a cluster-editing set F for $\mathcal{C}$ such that $|F|$ is equal to the number of packed P_3s incident with a vertex of C. If there is such a cluster-editing set F, then perform the operations of F to $\mathcal{C}$ and remove the corresponding packed P_3s from $\mathcal{H}$. Otherwise, if there is no such cluster-editing set F, return NO.*

▶ **Reduction Rule 5.** *If there is a proto-cluster C which is an isolated clique, then remove C from the graph.*

Already, the above simple rules effectively remove proto-clusters of size at least four:

▶ **Lemma 5.** *After applying Reduction Rules 1 - 5 exhaustively, if the algorithm did not return NO, then there is no proto-cluster of size at least 4.*

Proof. It is not hard to check that there are no isolated proto-clusters of size at least 4 after the previous reduction rules. Assume that there is a proto-cluster A of size at least 5 and a vertex $v \in V(G) \setminus V(A)$ such that v is adjacent to a vertex of A. There are at least five vertex pairs between v and $V(A)$ which are covered by packed P_3s because Reduction Rule 3 is not applicable. But v is incident with at most four packed edges or packed non-edges because of half-integrality, a contradiction. Next, assume that there is a proto-cluster B of size exactly 4 and a vertex $u \in V(G) \setminus V(B)$ such that u is adjacent to a vertex of B. There are four vertex pairs between u and $V(B)$ and, moreover, these are covered by two type-α P_3 since there is no type-β P_3 after Reduction Rule 2. We claim that $V(B) \cup \{u\}$ induces a connected component $\mathcal{C}$ in the graph. Suppose for contradiction that there is another vertex x adjacent to one vertex of $V(B)$. Then either Reduction Rule 3 can be applied if the vertex pairs between x and $V(B)$ are not all packed or otherwise B is not a proto-cluster: It would have to contain four packed edges and hence could not contain a spanning tree of non-packed edges. Thus Reduction Rule 4 applies to $\mathcal{C}$, again a contradiction. ◀

Next, only proto clusters of size at most three remain, but those of size exactly three have a very restricted structure. The following rule takes care of them.

▶ **Reduction Rule 6.** *After applying Reduction Rules 1 - 5 exhaustively, if there is a proto-cluster C of size 3, a proto-cluster B of size 1 and a proto-cluster A of size 1 such that C is not isolated from B, and a type-γ P_3 connects A and C via B, then delete the packed edge between A and B, insert an edge to the packed non-edge between C and B, and remove the corresponding P_3s from $\mathcal{H}$.*

▶ **Lemma 6.** *After applying Reduction Rules 1 - 6 exhaustively, there are no isolated cliques in the instance and every proto-cluster of the instance is of size at most 2. Moreover, every packed P_3 is a type-γ P_3.*

After applying Reduction Rules 1 - 6 exhaustively, suppose that the resulting instance $(G, \mathcal{H}, \ell = 0)$ of CEaHMP has a solution S. We now focus on the sizes of the clusters in $G \triangle S$. We can see that the maximum size of a cluster in $G \triangle S$ is six by some simple observation. The next lemma shows that clusters of size exactly six can be removed by Reduction Rule 4.

▶ **Lemma 7.** *Let $(G, \mathcal{H}, \ell = 0)$ be an instance of CEaHMP such that the size of every proto-cluster in G is at most 2 and let S be a solution to $(G, \mathcal{H}, \ell = 0)$. Suppose that A is a clique of size 6 in $G \triangle S$. Then the vertices of $V(A)$ belong to three proto-clusters C_1, C_2, and C_3 of size two in G. In addition, every vertex pair between C_1 and C_2, between C_1 and C_3, between C_2 and C_3 is covered by some P_3 of $\mathcal{H}$. Furthermore, $V(C_1) \cup V(C_2) \cup V(C_3)$ forms a connected component $\mathcal{C}$ in G.*

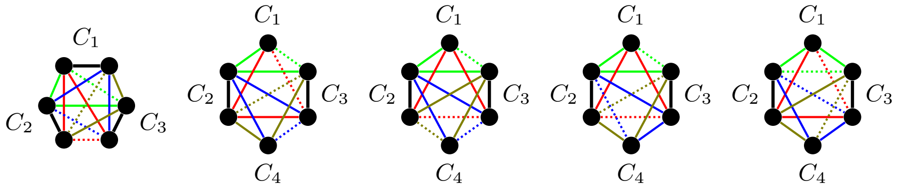


Figure 3 The first picture is an example of forming a clique of size 6 in $G \triangle S$ as in Lemma 7, the other pictures are potential examples of forming a clique of size 5 in $G \triangle S$ as in Lemma 8.

Clusters of size four and five also have restricted structures as shown by following lemmas.

▶ **Lemma 8.** *After applying Reduction Rules 1 – 3 exhaustively, let $(G, \mathcal{H}, \ell = 0)$ be an instance of CEaHMP such that the size of every proto-cluster in G is at most 2 and S is a solution to $(G, \mathcal{H}, \ell = 0)$. Suppose that A is a clique of size 5 in $G \triangle S$. Then the vertices of $V(A)$ belong to three proto-clusters C_1, C_2 and C_3 (or C_2, C_3 and C_4) in G such that $|C_1| = |C_4| = 1$ and $|C_2| = |C_3| = 2$. Every vertex pair between C_i and C_j $(i, j \in \{1, 2, 3, 4\}, i \neq j)$ is covered by a packed P_3 except that the vertex pair between C_1 and C_4 is a non-packed non-edge. In addition, $V(C_1) \cup V(C_2) \cup V(C_3) \cup V(C_4)$ forms a connected component $\mathcal{C}$ in G.*

▶ **Lemma 9.** *After applying Reduction Rules 1 – 6 exhaustively, let $(G, \mathcal{H}, \ell = 0)$ be an instance of CEaHMP such that the size of every proto-cluster in G is at most 2 and S is a solution to $(G, \mathcal{H}, \ell = 0)$. Suppose that A is a clique of size 4 in $G \triangle S$ and $V(A) = \{x, y, z_1, z_2\}$. Then three vertices of $V(A)$, say x, y, z_2 belong to one packed P_3 in G, and one vertex of x, y, z_2, say z_2, with z_1 forms a proto-cluster C_1 of size two in G while x and y form a proto-cluster C_2 of size one and a proto-cluster C_3 of size one in G respectively. Moreover, there are two vertices u and v such that x, u, z_1 belong to a packed P_3 in G, y, v, z_1 belong to another packed P_3 in G. u and v form a proto-cluster C_4 of size one and a proto-cluster C_5 of size one in G respectively.*

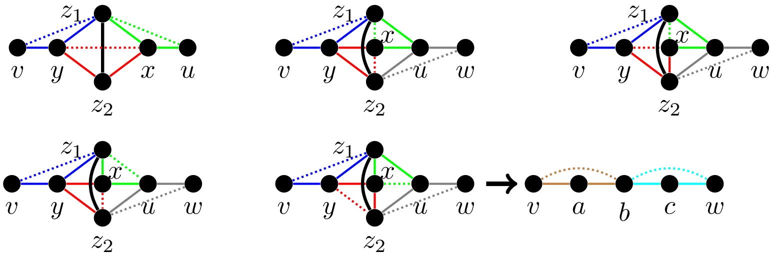


Figure 4 Subgraphs that may form clusters of size 4 and how to reduce them (bottom right).

Based on the previous lemmas, we design some reduction rules that handle all potential cases in which there are cliques of size at least 4 in $G \triangle S$. Due to space constraints they appear only in the full version [37]. We show some examples of these cases in Figures 3 and 4.

▶ **Lemma 10.** *After applying all reduction rules exhaustively, let $(G, \mathcal{H}, \ell = 0)$ be an instance of CEAHMP which has a solution S. Then there is no clique of size at least 4 in $G \triangle S$.*

Next, we introduce a new problem called Cluster Deletion above modification-disjoint P_3 packing (CDAMP), defined as follows: Given a graph $G = (V, E)$, a modification-disjoint packing $\mathcal{H}$ of induced P_3s of G, and a non-negative integer ℓ, decide whether there is a *cluster-deletion set*, that is, a set of edges $S \subseteq E$ so that $G' = (V, E \setminus S)$ is a union of disjoint cliques, with $|S| - |\mathcal{H}| \leq \ell$.

▶ **Lemma 11.** *Let $(G, \mathcal{H}, \ell = 0)$ be an instance of CEAHMP. After applying all reduction rules exhaustively, we get an instance $(G', \mathcal{H}', \ell = 0)$ of CEAHMP. Then $(G, \mathcal{H}, \ell = 0)$ is a YES-instance of CEAHMP if and only if $(G', \mathcal{H}', \ell = 0)$ is a YES-instance of CDAMP.*

Let $(G', \mathcal{H}', \ell = 0)$ be the resulting instance of CDAMP. Let $E_c \subseteq E(G')$ be the set of edges covered by some P_3 of $\mathcal{H}'$ and let $\lambda = 2|\mathcal{H}'|$. We fix an arbitrary ordering of the edges of E_c and label these edges by $e_0, e_1, ..., e_{\lambda-1}$ according to this ordering. We construct an instance of 2-SAT with λ variables $x_0, x_1, ..., x_{\lambda-1}$ as follows. First, initialize the 2-SAT formula $\Phi = \mathsf{true}$. For each induced P_3 $xyz \in \mathcal{H}'$, let $e_i = xy, e_j = yz$ and update $\Phi \leftarrow \Phi \wedge (x_i \vee x_j) \wedge (\neg x_i \vee \neg x_j)$. For each induced P_3 uvw in G' such that uv and vw belong to two distinct P_3s of $\mathcal{H}'$ respectively, if $uv = e_p$ and $vw = e_q$, then update $\Phi \leftarrow \Phi \wedge (x_p \vee x_q)$. This completes the construction of the 2-SAT instance.

▶ **Lemma 12.** *Let $(G', \mathcal{H}', \ell = 0)$ be an instance of CDAMPand construct a 2-SAT formula Φ as described above. Then $(G, \mathcal{H}, \ell = 0)$ is a YES-instance if and only if Φ is satisfiable.*

As a result we can reduce the problem to 2-SAT and solve it in polynomial time.

5 Conclusion

Unfortunately the lower bound that we have obtained is a major roadblock in designing fixed-parameter algorithms for Cluster Editing parameterized above modification-disjoint P_3s. On the positive side, Cluster Editing above half-integral P_3 packing (CEAHMP) admits an XP-algorithm with respect to the number of excess edits. We have left open whether CEAHMP is fixed-parameter tractable. Towards this, on the one hand the half-integral P_3 packings provide quite strong structure that can be exploited to design several branching rules. On the other hand, when attacking this question from several angles we discovered large grid-like structures that seemed difficult to overcome in fixed-parameter time, and a corresponding W[1]-hardness result would also not be surprising.

A different future research direction is to deconstruct our hardness reduction by examining which substructures it contains that are seldom in practical data. Forbidding such substructures may destroy the already somewhat fragile hardness construction, perhaps paving the way for fixed-parameter algorithms.

Finally, it would be interesting to see how modification-disjoint P_3 packings look in practice. If it is true that only few vertices are in a large number of packed P_3s and most of them are in a small constant number, then a strategy that settles the clustering around the vertices with large number of P_3s and then applies reduction rules from Section 4 could be efficient.

References

1 Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5):23:1–23:27, 2008. doi:10.1145/1411509.1411513.

2 Takuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/FPT algorithms in practice: A case study of vertex cover. *Theoretical Computer Science*, 609:211–225, 2016. doi:10.1016/j.tcs.2015.09.023.

3 Noga Alon, Konstantin Makarychev, Yury Makarychev, and Assaf Naor. Quadratic forms on graphs. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC 2005)*, pages 486–493. ACM, 2005. doi:10.1145/1060590.1060664.

4 Sanjeev Arora, Eli Berger, Elad Hazan, Guy Kindler, and Muli Safra. On non-approximability for quadratic programs. *Electronic Colloquium on Computational Complexity (ECCC)*, 058, 2005. URL: http://eccc.hpi-web.de/eccc-reports/2005/TR05-058/index.html.

5 Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004. doi:10.1023/B:MACH.0000033116.57574.95.

6 Amir Ben-Dor, Ron Shamir, and Zohar Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3/4):281–297, 1999. doi:10.1089/106652799318274.

7 S. Böcker, S. Briesemeister, Q.B.A. Bui, and A. Truss. Going weighted: Parameterized algorithms for cluster editing. *Theoretical Computer Science*, 410(52):5467–5480, 2009. doi:10.1016/j.tcs.2009.05.006.

8 Sebastian Böcker. A golden ratio parameterized algorithm for cluster editing. *Journal of Discrete Algorithms*, 16:79–89, 2012. doi:10.1016/j.jda.2012.04.005.

9 Sebastian Böcker and Jan Baumbach. Cluster editing. In Paola Bonizzoni, Vasco Brattka, and Benedikt Löwe, editors, *Proceedings of the 9th Conference on Computability in Europe (CiE 2013)*, volume 7921 of *Lecture Notes in Computer Science*, pages 33–44. Springer, 2013. doi:10.1007/978-3-642-39053-1_5.

10 Sebastian Böcker, Sebastian Briesemeister, and Gunnar W. Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60(2):316–334, 2011. doi:10.1007/s00453-009-9339-7.

11 Sebastian Böcker and Peter Damaschke. Even faster parameterized cluster deletion and cluster editing. *Information Processing Letters*, 111(14):717–721, 2011. doi:10.1016/j.ipl.2011.05.003.

12 Hans L. Bodlaender, Michael R. Fellows, Pinar Heggernes, Federico Mancini, Charis Papadopoulos, and Frances A. Rosamond. Clustering with partial information. *Theoretical Computer Science*, 411(7-9):1202–1211, 2010. doi:10.1016/j.tcs.2009.12.016.

13 Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. Multicut is FPT. *SIAM Journal on Computing*, 47(1):166–207, 2018. doi:10.1137/140961808.

14 Yixin Cao and Jianer Chen. Cluster editing: Kernelization based on edge cuts. *Algorithmica*, 64(1):152–169, 2012. doi:10.1007/s00453-011-9595-1.

15 Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. *Journal of Computer and System Sciences*, 71(3):360–383, 2005. doi:10.1016/j.jcss.2004.10.012.

16 Jianer Chen and Jie Meng. A $2k$ kernel for the cluster editing problem. *Journal of Computer and System Sciences*, 78(1):211–220, 2012. doi:10.1016/j.jcss.2011.04.001.

17 Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr A. Golovach. A survey of parameterized algorithms and the complexity of edge modification. *arXiv:2001.06867 [cs]*, 2020. arXiv:2001.06867.

18 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. On multiway cut parameterized above lower bounds. *ACM Transactions on Computation Theory*, 5(1):3:1–3:11, 2013. doi:10.1145/2462896.2462899.

19 Peter Damaschke. Fixed-parameter enumerability of cluster editing and related problems. *Theory of Computing Systems*, 46(2):261–283, 2010. doi:10.1007/s00224-008-9130-1.

20 Michael R. Fellows. The lost continent of polynomial time: Preprocessing and kernelization. In Hans L. Bodlaender and Michael A. Langston, editors, *Proceedings of the Second International Workshop on Parameterized and Exact Computation (IWPEC 2006)*, volume 4169 of *Lecture Notes in Computer Science*, pages 276–277. Springer, 2006. doi:10.1007/11847250_25.

21 Michael R. Fellows, Jiong Guo, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. Graph-based data clustering with overlaps. *Discrete Optimization*, 8(1):2–17, 2011. doi:10.1016/j.disopt.2010.09.006.

22 Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Peter Shaw. Efficient parameterized preprocessing for cluster editing. In Erzsébet Csuhaj-Varjú and Zoltán Ésik, editors, *Proceedings of the 16th International Symposium on Fundamentals of Computation Theory (FCT 2007)*, volume 4639 of *Lecture Notes in Computer Science*, pages 312–321. Springer, 2007. doi:10.1007/978-3-540-74240-1_27.

23 Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Yngve Villanger. Subexponential fixed-parameter tractability of cluster editing. *arXiv:1112.4419 [cs]*, 2013. arXiv:1112.4419.

24 Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Yngve Villanger. Tight bounds for parameterized complexity of cluster editing with a small number of clusters. *Journal of Computer and System Sciences*, 80(7):1430–1447, 2014. doi:10.1016/j.jcss.2014.04.015.

25 Vincent Froese. *Fine-Grained Complexity Analysis of Some Combinatorial Data Science Problems*. PhD thesis, Technische Universität Berlin, 2018. doi:10.14279/depositonce-7123.

26 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)*, pages 1152–1166. SIAM, 2016. doi:10.1137/1.9781611974331.ch80.

27 Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Automated generation of search tree algorithms for hard graphmodification problems. *Algorithmica*, 39(4):321–347, 2004. doi:10.1007/s00453-004-1090-5.

28 Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory of Computing Systems*, 38(4):373–392, 2005. doi:10.1007/s00224-004-1178-y.

29 Jiong Guo. A more effective linear kernelization for cluster editing. *Theoretical Computer Science*, 410(8):718–726, 2009. doi:10.1016/j.tcs.2008.10.021.

30 Jiong Guo, Iyad A. Kanj, Christian Komusiewicz, and Johannes Uhlmann. Editing graphs into disjoint unions of dense clusters. *Algorithmica*, 61(4):949–970, 2011. doi:10.1007/s00453-011-9487-4.

31 Jiong Guo, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. A more relaxed model for graph-based data clustering: s-plex cluster editing. *SIAM Journal on Discrete Mathematics*, 24(4):1662–1683, 2010. doi:10.1137/090767285.

32 Yoichi Iwata. Linear-time kernelization for feedback vertex set. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 68:1–68:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.68.

33 Bart M.P. Jansen, Christian Schulz, and Hisao Tamaki. NII shonan meeting report no. 144 parameterized graph algorithms and data reduction, 2019. URL: https://shonan.nii.ac.jp/docs/No.144.pdf.

34 Krzysztof Kiljan and Marcin Pilipczuk. Experimental evaluation of parameterized algorithms for feedback vertex set. In Gianlorenzo D'Angelo, editor, *Proceedings of the 17th International Symposium on Experimental Algorithms (SEA 2018)*, volume 103 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:12, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.SEA.2018.12.

35 Christian Komusiewicz and Johannes Uhlmann. Cluster editing with locally bounded modifications. *Discrete Applied Mathematics*, 160(15):2259–2270, 2012. doi:10.1016/j.dam.2012.05.019.

36 Stefan Kratsch. A randomized polynomial kernelization for vertex cover with a smaller parameter. *SIAM Journal on Discrete Mathematics*, 32(3):1806–1839, 2018. doi:10.1137/16M1104585.

37 Shaohua Li, Marcin Pilipczuk, and Manuel Sorge. Cluster editing parameterized above modification-disjoint P_3-packings. *arXiv:1910.08517 [cs]*, 2019. arXiv:1910.08517.

38 Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Transactions on Algorithms*, 11(2):15:1–15:31, 2014. doi:10.1145/2566616.

39 Meena Mahajan and Venkatesh Raman. Parameterizing above guaranteed values: Maxsat and maxcut. *Journal of Algorithms*, 31(2):335–354, 1999. doi:10.1006/jagm.1998.0996.

40 Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. *SIAM Journal on Computing*, 43(2):355–388, 2014. doi:10.1137/110855247.

41 John H. Morris, Leonard Apeltsin, Aaron M. Newman, Jan Baumbach, Tobias Wittkop, Gang Su, Gary D. Bader, and Thomas E. Ferrin. clusterMaker: a multi-algorithm clustering plugin for Cytoscape. *BMC Bioinformatics*, 12(1):436, 2011. doi:10.1186/1471-2105-12-436.

42 Fábio Protti, Maise Dantas da Silva, and Jayme Luiz Szwarcfiter. Applying modular decomposition to parameterized cluster editing problems. *Theory of Computing Systems*, 44(1):91–104, 2009. doi:10.1007/s00224-007-9032-7.

43 Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004. doi:10.1016/j.dam.2004.01.007.

44 René van Bevern, Vincent Froese, and Christian Komusiewicz. Parameterizing edge modification problems above lower bounds. *Theory of Computing Systems*, 62(3):739–770, 2018. doi:10.1007/s00224-016-9746-5.

45 Tobias Wittkop, Dorothea Emig, Sita Lange, Sven Rahmann, Mario Albrecht, John H. Morris, Sebastian Böcker, Jens Stoye, and Jan Baumbach. Partitioning biological data with transitivity clustering. *Nature Methods*, 7(6):419–420, 2010. doi:10.1038/nmeth0610-419.

Exploiting Dense Structures in Parameterized Complexity

William Lochet
Department of Informatics, University of Bergen, Norway

Daniel Lokshtanov
University of California Santa Barbara, CA, USA

Saket Saurabh
Institute of Mathematical Sciences, Chennai, India
Department of Informatics, University of Bergen, Norway

Meirav Zehavi
Ben Gurion University of the Negev, Beer Sheva, Israel

Abstract

Over the past few decades, the study of dense structures from the perspective of approximation algorithms has become a wide area of research. However, from the viewpoint of parameterized algorithm, this area is largely unexplored. In particular, properties of random samples have been successfully deployed to design approximation schemes for a number of fundamental problems on dense structures [Arora et al. FOCS 1995, Goldreich et al. FOCS 1996, Giotis and Guruswami SODA 2006, Karpinksi and Schudy STOC 2009]. In this paper, we fill this gap, and harness the power of random samples as well as structure theory to design kernelization as well as parameterized algorithms on dense structures. In particular, we obtain linear vertex kernels for Edge-Disjoint Paths, Edge Odd Cycle Transversal, Minimum Bisection, d-Way Cut, Multiway Cut and Multicut on everywhere dense graphs. In fact, these kernels are obtained by designing a polynomial-time algorithm when the corresponding parameter is at most $\Omega(n)$. Additionally, we obtain a cubic kernel for Vertex-Disjoint Paths on everywhere dense graphs. In addition to kernelization results, we obtain randomized subexponential-time parameterized algorithms for Edge Odd Cycle Transversal, Minimum Bisection, and d-Way Cut. Finally, we show how all of our results (as well as EPASes for these problems) can be de-randomized.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Dense graphs, disjoint paths, odd cycle transversal, kernels

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.50

Funding Lochet and Saurabh were supported by European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant no. 819416).
Daniel Lokshtanov: Lokshtanov is supported by BSF grant no. 2018302. and NSF award CCF-2008838.
Saket Saurabh: Saurabh also acknowledges Swarnajayanti Fellowship grant DST/SJF/MSA-01/2017-18.
Meirav Zehavi: Zehavi was supported by Israel Science Foundation (ISF) grant no. 1176/18, and United States – Israel Binational Science Foundation grant no. 2018302.

Acknowledgements We thank one of the referee of a different version of this paper for sketching a proof based on the random walk on expanders for Lemma 10.

1 Introduction

While several interesting optimization problems remain NP-complete even when restricted to sparse graphs or dense graphs, the restriction of a problem to these families of graphs is usually considerably more tractable algorithmically than the problem on general graphs.

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 50; pp. 50:1–50:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

With respect to graph classes, sparseness usually refers to families of planar graphs, graphs of bounded genus, graphs excluding some fixed graph H as a minor, graphs of bounded expansion and no-where dense graphs. Here, denseness usually refers to families of graphs with $\Omega(n^2)$ edges. Additionally, sparseness and denseness can be defined for structures beyond graphs – for example, dense 3-SAT instances are those for which the formula has $\Omega(n^3)$ clauses.

In this paper, we focus on designing *deterministic* kernelization algorithms and fixed-parameter tractable (FPT) algorithms for NP-hard problems on dense structures.

We start by defining some basic definitions from Parameterized Complexity, that we make use of. Formally, a *parameterization* of a problem is assigning an integer k to each input instance and we say that a parameterized problem is *fixed-parameter tractable (FPT)* if there is an algorithm that solves the problem in time $f(k) \cdot |I|^{O(1)}$, where $|I|$ is the size of the input and f is an arbitrary computable function depending on the parameter k only. We will also be studying polynomial time preprocessing or kernelization.

A parameterized problem Π is said to admit a *kernel* if there is a polynomial-time algorithm, called a *kernelization algorithm*, that reduces the input instance of Π down to an equivalent instance of Π whose size is bounded by a function $f(k)$ of k. (Here, two instances are equivalent if both of them are either Yes-instances or No-instances.) Such an algorithm is called an $f(k)$-*kernel* for Π. If $f(k)$ is a polynomial function of k, we say that the kernel is a *polynomial kernel.* For more background on Parameterized Complexity and Kernelization, we refer to the following books [21, 15, 23, 46, 25].

1.1 Context of Our Results and Overarching Goals

The algorithmic study of NP-hard problems on dense structures is two decade old and has a rich history. We start by giving definitions of (E)PTAS and denseness that will ease our discussion. A PTAS is an algorithm that takes an instance I of an optimization problem and a parameter $\epsilon > 0$, runs in time $n^{\mathcal{O}(f(1/\epsilon))}$, and produces a solution that is within a factor $1 + \epsilon$ of being optimal. A PTAS with running time $f(1/\epsilon) \cdot n^{\mathcal{O}(1)}$ is called an efficient PTAS (EPTAS).

▶ **Definition 1** ([7, 33]). *A graph on n vertices is δ-*dense *if it has $\delta n^2/2$ edges. It is* everywhere-δ-dense *if the minimum degree is δn. We abbreviate $\Omega(1)$-dense as dense and everywhere-$\Omega(1)$-dense as everywhere-dense.*

Arora, Karger and Karpinski [7] initiated the study of NP-hard problems on dense structures and designed PTASes for several NP-hard optimization problems. Among many other results, they showed that Bisection, k-Way Cut, and Separator admit PTASes on everywhere-dense instances and Max-Cut, Max-d-SAT, and Max-Hypercut(d) admit PTASes on dense instances. The main ingredients of these results are *exhaustive sampling* and its use in approximation of polynomial integer programs. These results lead to a flurry of new ideas and results in this area. Arora, Frieze, and Kaplan [6] used the exhaustive sampling idea to design additive approximation schemes for problems in which feasible solutions are permutations (such as the 0-1 Quadratic Assignment Problem). Frieze and Kannan [27] and, independently, Goldreich, Goldwasser, and Ron [29] showed that exhaustive sampling techniques apply because of certain regularity properties in dense graphs and used this observation to design linear time additive approximation schemes for most of the problems that were considered in [7]. In particular, [27, 29] made PTASes of [7] into EPTASes. Frieze and Kannan [27] also pointed out connections to constructive versions of Szemeredi's Regularity Lemma and Goldreich, Goldwasser, and Ron [29] found its connection in property testing and learning theory based on an idea of degree estimator.

This idea of degree estimator has been extremely useful in further developments in the area. In particular, Giotis and Guruswami [28] used this idea to design a PTAS for correlation clustering in general graphs, when the number of clusters is fixed. That is, they designed a PTAS for d-Correlation Clustering (given an undirected graph G, edit (delete or add) minimum number of edges so that the resulting graph becomes a disjoint union of d cliques) running in time $n^{\mathcal{O}(9^d/\epsilon^2)}\log n$. It is also important to note here that before the paper of Giotis and Guruswami [28], most of the earlier works largely focused on maximization problems. In 2009, Karpinski and Schudy [33] further used the idea of degree estimator and designed linear time EPTASes for several problems, such as d-Correlation Clustering and Fragile Min-d-CSP on everywhere-dense instances. Several other randomized PTASes and EPTAses based on different sets of ideas can be found in [43, 19, 32, 8, 2, 1, 5].

As we established above the algorithmic study of NP-hard problems on dense structures has been extremely rewarding from the perspective of Approximation Algorithms. Could this success be repeated in other algorithmic paradigms meant to cope up with NP-hard problems? In particular, in the field of Parameterized Complexity. This leads to the following question.

> Could we exploit the denseness of structures in designing significantly faster FPT algorithms and polynomial time kernelization algorithm for some of the fundamental problems in the field, the way it has been utilized in the field of approximation algorithms?

Our study shows that the answer is an assertive YES! In particular, we obtain linear kernels for Edge-Disjoint Paths, Edge Odd Cycle Transversal, Minimum Bisection, d-Way Cut, Multiway Cut and Multicut on everywhere dense graphs. In fact, these kernels are obtained by designing a polynomial-time algorithm when the corresponding parameter is $\Omega(n)$. Additionally, we obtain a cubic kernel for Vertex-Disjoint Paths on everywhere dense graphs. In addition to kernelization results, we obtain randomized subexponential-time parameterized algorithms for Edge Odd Cycle Transversal, Minimum Bisection, and d-Way Cut. Finally, we show how all of our results (as well as EPASes for these problems) can be de-randomized.

1.2 Our Results and Methods

In this section we give a brief overview of the problems we address and the results we obtain for these problems. This is complemented with a short discussion on techniques that we apply to design our algorithms.

For maximization problems such as Max Cut on dense graphs, a solution would have size $k = \Omega(n^2)$, which trivially yields solvability in subexponential-time (i.e. $2^{o(k)} \cdot n^{\mathcal{O}(1)}$-time) with respect to k. This is true about several maximization problems. However, this is not the case for well-studied minimization problems such as Edge Odd Cycle Transversal, Minimum Bisection, d-Way Cut, Multiway Cut and Multicut. Thus, a natural class of problems to consider are so called *cut-problems*. The other family of problems for which we do not immediately get an algorithm are *linkage problems*, namely, the Edge-Disjoint Paths and Vertex-Disjoint Paths problems.

We remark that the study of subexponential-time parameterized algorithms of *vertex* (rather than edge) modification problem on everywhere-dense graphs does not make sense for natural problems such as Vertex Cover as such problems become as hard as they are on general graphs (and hence do not admit such algorithms under the ETH). For example, given an instance G of Vertex Cover, create an instance G' of Vertex Cover on everywhere-dense graphs by adding an n-vertex clique whose vertices are all but one adjacent to every vertex of G. Then, the existence of an $2^{o(k)}n^{\mathcal{O}(1)}$-time algorithm for Vertex Cover on everywhere-dense graphs where k is the solution size would imply the existence of a subexponential-time algorithm for Vertex Cover on general graphs with respect to n.

1.2.1 Linkage Problems

The first two problems we address are extremely fundamental in the field of Parameterized Complexity. They are Edge-Disjoint Paths and Vertex-Disjoint Paths. In the Edge-Disjoint Paths problem, we are given a graph G, a set of request pairs $(s_1, t_1), \dots, (s_k, t_k)$, and the objective is to check whether there exist paths $P_1, \dots, P_k$, between s_i and t_i, such that they are pairwise edge disjoint. In the Vertex-Disjoint Paths problem, the input is same as the Edge-Disjoint Paths problem, but the paths $P_1, \dots, P_k$ are suppose to be pairwise vertex disjoint. Both, Edge-Disjoint Paths and Vertex-Disjoint Paths are famously FPT by the graph minor machinery of Robertson and Seymour [48]. However, the $f(k)$ in the running time in the algorithm of Robertson and Seymour [48] and its later improvement is at least triply exponential [36]. Only recently an algorithm with $f(k) = 2^{k^{\mathcal{O}(1)}}$ are designed when the input is restricted to planar graphs [39]. Further, Vertex-Disjoint Paths is not known not to admit a polynomial kernel on general graphs [10]. In this paper we show that both Edge-Disjoint Paths and Vertex-Disjoint Paths admit a polynomial kernel on α-dense graphs. In particular we get the following result about Edge-Disjoint Paths.

▶ **Theorem 2.** *Edge-Disjoint Paths admits an $\mathcal{O}(k)$ vertex kernel on everywhere α-dense graphs.*

Proof of Theorem 2 is obtained by designing a polynomial time algorithm for the Edge-Disjoint Paths problem in α-dense graphs, when the number of demands is small (but still linear) compared to αn. Once this result is proved we know that $k \geq \Omega(n)$, resulting in a linear vertex kernel for the problem.

To design the desired polynomial time algorithm, we use the following strategy. We start by showing that highly edge-connected (linear in n) parts will always contain a solution to an Edge-Disjoint Paths instance. Towards this we first show that if a graph G on n vertices with minimum degree at least cn, then for any pair of vertices x, y of G, if there exists a path between x and y, then there exists a path of length at most $4/c$. We use this result together with high connectivity of G to get the following: Let G be a graph with minimum degree αn, and cn edge-connected for some constant $c \leq \alpha/2$, then any instance of Edge-Disjoint Paths with $k \leq \frac{\alpha n}{8}$ has a solution. Moreover, this solution can be found in polynomial time. Next, we give a lemma that partitions the input graph into small number of parts such that each part has minimum degree and edge-connectivity linear in n.

▶ **Lemma 3.** *For any real α between 0 and 1, there exists a constant $c \leq \alpha/2$ such that, if G is a graph on n vertices and minimum degree αn, then there exists a partition $\mathcal{P}$ of the vertices $V(G)$ into $g \leq \frac{2}{\alpha}$ subsets $V_1, \cdots, V_g$ such that for all $i \in [g]$:*

- *$G[V_i]$ is cn edge-connected.*
- *$G[V_i]$ has minimum degree $\frac{\alpha n}{2}$.*

Moreover, such a partition can be found in polynomial time.

This decomposition is then utilized to complete the proof of Theorem 2.

Our kernelization algorithm for Vertex-Disjoint Paths is more involved, though follows the template outlined for Edge-Disjoint Paths. In particular we obtain the following result.

▶ **Theorem 4** ($\star$).[1] *Vertex-Disjoint Paths admits a vertex kernel of size $\mathcal{O}(k^3)$ on everywhere α-dense graphs.*

One of the main technical difficulty in proving Theorem 4 is in adapting the proof of Lemma 3 for Vertex-Disjoint Paths. The main reason being that for Vertex-Disjoint Paths we need to simulate Lemma 3 for vertex connectivity. That is, we need to find *cut-vertices instead of edges*. However, these vertices could have neighbors in many different parts and we cannot say that their relative degree inside a part increases, which is a critical component in the proof of Lemma 3. To mitigate this situation we introduce a vertex set V_0 in the partitioning, that contains all the cut vertices. The whole difficulty lies in carrying this V_0 throughout the process of obtaining the desired partition. However, unlike Edge-Disjoint Paths, getting the desired decomposition in itself does not result in the desired kernel. We need to put in significant technical work to reduce the graph. To achieve this we prove several structural properties of Vertex-Disjoint Paths and its interplay with the parts of $\mathcal{P}$ in order to get the desired kernel.

1.2.2 Cut-Problems

Arguably, a few of the most well-studied cut problems in the realm of Parameterized Complexity are Edge Odd Cycle Transversal, Minimum Bisection, d-Way Cut, Multiway Cut, and Multicut. Input to all these problems are an undirected graph G and an integer k, and the goal is following.

Edge Odd Cycle Transversal: Does there exist a set of at most k edges such that its deletion results in a bipartite graph?

Minimum Bisection: Does there exist a vertex partition (V_1, V_2), such that $||V_1| - |V_2|| \leq 1$, and there are at most k edges with one end-point in V_1 and the other in V_2?

d-Way Cut: Does there exist a set of at most k edges such that its deletion results in at least d connected components?

Multiway Cut: Here, we are also given a vertex subset $T \subseteq V(G)$ (called terminals) and the objective is to test if there exists a set of at most k edges such that after its deletion no two terminals belong to the same connected component.

Multicut: Here, we are also given a set of request $(s_1, t_1), \ldots, (s_\ell, t_\ell)$ and the objective is to test if there exists a set of at most k edges such that after its deletion no request belong to the same connected component.

All the aforementioned problems are extremely well studied [18, 16, 14, 47, 41, 42, 12, 13, 35, 11] and are known to be FPT. However, for most of these problems we know that there can not exist an algorithm with running time $2^{o(k)}n^{\mathcal{O}(1)}$ on general graphs assuming ETH. Further, Edge Odd Cycle Transversal admits a randomized polynomial kernel on general graphs [37, 38]; on the other hand Minimum Bisection and Multicut are known not to admit a polynomial kernel [17, 49]. The kernelization complexity of Multiway Cut is still open. In this paper we obtain the following results about these problems on everywhere dense graphs.

[1] Results marked with ($\star$) could be found in the extended version.

▶ **Theorem 5** (⋆). Edge Odd Cycle Transversal, Minimum Bisection, *d*-Way Cut, Multiway Cut, *and* Multicut *admit $\mathcal{O}(k)$ vertex kernel on everywhere α-dense graphs.*

▶ **Theorem 6** (⋆). Edge Odd Cycle Transversal, *and* Minimum Bisection *admit an algorithm with running time $2^{\mathcal{O}(\sqrt{k})}n^{\mathcal{O}(1)}$ on everywhere α-dense graphs. Further,* d-Way Cut *admits an algorithm with running time $2^{\mathcal{O}(\sqrt{k}\log k)}n^{\mathcal{O}(1)}$.*

These are the first subexponential time parameterized algorithms for Edge Odd Cycle Transversal, Minimum Bisection, and d-Way Cut on everywhere α-dense graphs. The proof of Theorem 5 is obtained by designing a polynomial time algorithm when the solution size for these problems is smaller than $\alpha \cdot n$ (for some α). This is similar to our kernelization strategy for the Edge-Disjoint Paths problem. For example, if the solution for Edge Odd Cycle Transversal is of size $k \leq \alpha \cdot n$ (for some α), then the problem can be solved in polynomial time, and otherwise $n < k/\alpha$ and hence we already have a kernel at hand.

The proof of these results (Theorems 5 and 6) are similar to each other. Thus, to illustrate our methods we focus on giving intuition for the proof of d-Way Cut. A more formal presentation is left to the extended version. The main ingredient of Theorems 5 and 6 is the following sampling primitive, a simple consequence of Chebyshev's inequality which has been extensively used in designing PTASes and EPTASes in everywhere α-dense graphs.

▶ **Lemma 7** (Degree Estimator Lemma). *For any constants ϵ_1 and ϵ_2, if U is a universe on n elements, $\mathcal{K}$ is a set of subsets of U and S is a multi-set obtained by doing $t(\epsilon_1, \epsilon_2) = \frac{1}{\epsilon_1^2\epsilon_2}$ independent and uniform random draws in U, then with probability at least $1/2$, the number of sets $X \in \mathcal{K}$ such that $\left|\frac{|S\cap X|n}{t} - |X|\right| \geq \epsilon_1 n$ is smaller than $\epsilon_2|\mathcal{K}|$.*

We next show how we use Degree Estimator Lemma for our purpose. Suppose that G is a graph on n vertices and A is a set of linear size $\Omega(n)$. We use Lemma 7 in order to guess the degree of the vertices of $V(G)$ in A *without knowing* the set. That is, to estimate the number of neighbors of a vertex that belong to the set A. Indeed, let us fix some constants ϵ_1 and ϵ_2 and pick uniformly at random a set S of $t = t(\epsilon_1, \epsilon_2) = \frac{1}{\epsilon_1^2\epsilon_2}$ vertices from $V(G)$. Since A is of linear size, with constant probability, all the elements of S belong to A. If this event is satisfied, then by applying Lemma 7 with $U = A$ and $\mathcal{K}$ being the set of neighborhood inside A, we have that with probability at least $1/2$, the number of vertices x such that $\left|\frac{|S\cap N(x)||A|}{t} - d_A(x)\right| \geq \epsilon_1 n$ is smaller than $\epsilon_2 n$. In other words, without knowing A, the value $\frac{|S\cap N(x)||A|}{t}$ provides a good estimation of the degree in A for a large fraction of the vertices in $V(G)$.

Let us now see how we use the aforementioned argument for d-Way Cut. Let (G, k) be an instance of d-Way Cut, where G is a everywhere α-dense graph. Further assume that we are looking for a solution, S, where k is small, say $k \leq \frac{\alpha n}{200}$. Let $(A_1, \dots, A_d)$ be the connected components after removing the edges in S. Since, $k \leq \frac{\alpha n}{200}$ and every vertex has degree at least αn, this implies that every vertex $x \in A_i$ has degree at least $\alpha n - \frac{\alpha n}{200} \geq \frac{\alpha n}{2}$ in A_i, and degree less than $\frac{\alpha n}{200}$ in the other A_j, for $j \neq i$. It means that $|A_i| \geq \frac{\alpha n}{2}$ for every i, and thus $d \leq \frac{2}{\alpha}$.

The idea now is to estimate the degree of every vertex inside each A_i in two rounds. For the first round we sample d sets $M_1, \dots, M_d$ of $t = t(\alpha/200, \alpha^2/400)$ vertices each. By applying Lemma 7, with constant probability (because each A_i is linear), each M_i will be a subset of A_i such that the set X_i of vertices x for which $\left|\frac{|M_i\cap N(x)||A_i|}{t} - d_{A_i}(x)\right| \geq n\alpha/200$ is smaller than $n\alpha^2/400$. Assume that this is the case for every i, and let us denote $X = \cup_{i\in[d]} X_i$. Since $d \leq 2/\alpha$, we have that $|X| \leq \alpha n/200$. This means that apart from this small set X, all the other vertices x of G are such that $\frac{|M_i \cap N(x)||A_i|}{t}$ is a good estimate of its degree inside A_i[2]. Let us make our first guess of A_i: for every $i \in [d]$, let A'_i be the set x of vertices of G such that $\frac{|M_i \cap N(x)||A_i|}{t} \geq d(x) - \frac{\alpha n}{25}$. We can then show the following.

▷ Claim 8. For every $i \in [d]$, $(A_i \setminus X) \subseteq A'_i$.

Indeed, for every $x \in (A_i \setminus X)$, we have that $\frac{|M_i \cap N(x)||A_i|}{t} \geq d_{A_i}(x) - n\alpha/200 \geq (d(x) - k) - n\alpha/200 \geq d(x) - \alpha/n$ because $x \notin X_i$. Moreover, for every $j \neq i$, $\frac{|M_j \cap N(x)||A_j|}{t} \leq d_{A_j}(x) + n\alpha/200 \leq n\alpha/50$ because $x \in A_i$ and $x \notin X_j$.

For our second round, we use $d_{A'_i}(x)$ as an estimate for $d_{A_i}(x)$. Indeed, if $x \in A_i$, then Claim 8 implies that $d_{A'_i}(x) \geq d_{A_i}(x) - |X|$, *even if x belongs to X*. However, since $d_{A_i}(x) \geq d(x) - \alpha n/100$, we have that $d_{A'_i}(x) \geq d(x) - \alpha n/50$. Similarly, $d_{A'_j}(x) \leq d_{A_j}(x) + |X| \leq \alpha n/50$. Because $d(x) \geq \alpha n$ for every $x \in G$, we have the following claim.

▷ Claim 9. For every i, A_i is exactly the set of vertices x of G such that $d_{A'_i}(x) \geq d(x) - \alpha n/50$.

This ends the proof of a polynomial algorithm in the case $k \leq \alpha n/100$, which implies the proof of a linear kernel. The proofs for Edge Odd Cycle Transversal, Minimum Bisection, Multiway Cut, and Multicut are almost identical.

When $k \geq \alpha n/100$, we have to be more careful with respect to vertices that are incident to many edges of the solution, say more than $\alpha n/200$. Let us note that all of these problems admit an exact algorithm, by doing a dynamic programming algorithm over subset and applying fast subset-convolution, running in time $2^n n^{\mathcal{O}(1)}$ [9]. Thus, if $k \geq (\alpha n/200)^2$, then $2^n = 2^{\mathcal{O}(\sqrt{k})}$ and this algorithm is a subexponential time algorithm. If $k \leq (\alpha n/200)^2$, then we can show that the set L of vertices of G that are adjacent to more than $\alpha n/200$ edge of the *solution* is such that $|L| \leq \sqrt{k}$. Now by doing essentially the same argument as in the case $k \leq \alpha n/100$ we will be able to recover the position of every vertex x, except for a set $R \subseteq L$. To conclude, the algorithm then tries all the partitions of R. This part takes $|L|^{|L|} = 2^{\mathcal{O}(\sqrt{k}\log k)}$, resulting in the desired algorithm.

1.2.3 Derandomization

We first abstract out the main properties of Degree Estimator Lemma 7 that have been used in several applications in [7, 27, 29, 28, 33] and several other articles.

Let U be a universe of size n and t be a constant. A random sample S of t elements of U has the following properties:

Property A. For every subset A of the universe of $\Omega(n)$ elements, the probability that the sample S is a subset of A is constant;

Property B. Conditioned on the sample S being a subset of A, we have that for every subset B of A of size $\Omega(n)$, $\frac{|S \cap B||A|}{t}$ is a good estimator of $|B|$ with probability close to 1.

These two properties of random samples have been successfully deployed to design randomized approximation schemes for a number of fundamental problems on dense structures [7, 27, 29, 28, 33]. Typically, algorithms based on this approach can be de-randomized by going over all possible subsets S of size t, and observing that at least one of them has the

[2] We assume here that $|A_i|$ is known. In fact, an approximation to the size will be enough for our purpose.

desired property. Unfortunately, this leads to an overhead of roughly n^t in the running time (which typically yields deterministic PTASes in place of randomized EPTASes). We present an efficient way to derandomize most of the algorithms based on the procedure. Our main derandomization tool is the following lemma.

▶ **Lemma 10** ($\star$)**.** *For any constants ϵ_1, ϵ_2 and ϵ_3 smaller than 1, and U a universe on n elements, there exists a set $\mathcal{T}$ of $\mathcal{O}(2^{100/(\epsilon_1^2\epsilon_2)}n)$ subsets of U, such that if A is a subset of at least $\epsilon_3 n$ elements of U and $\mathcal{K}$ a collection of subsets of A, then there exists a set $T \in \mathcal{T}$ such that the number of sets X of $\mathcal{K}$ such that $||T \cap X| - \frac{|T||X|}{|A|}| \geq \epsilon_1|T|$ is smaller than $\epsilon_2|\mathcal{K}|$. Moreover, the set $\mathcal{T}$ can be computed deterministically in $n^{\mathcal{O}(1)}$ time.*

Therefore, in all the proof using Lemma 7, we can replace the random sampling by trying all the elements of the family $\mathcal{T}$ provided by the Lemma 10. The proof involves using the known construction of pairwise (2-wise) independent permutations (see [4] for more details). The proof can also be done via expander random walk method (see Section 3.2 of [30]).

1.3 Related Works

Over the last two decade, the design of parameterized subexponential-time algorithms for problems on sparse graphs has been extremely fruitful. However, the same could not be said about research on dense graphs. The first problem on dense graphs shown to admit a parameterized subexponential-time algorithm is the Feedback Arc Set on Tournaments (FAST) problem [3]. The design of this algorithm exhibited a new method to develop parameterized algorithms called chromatic coding, which is now textbook material [15]. Subsequently, there appeared several other works on the design of parameterized subexponential-time algorithms for problems on tournaments, see e.g. [26, 22, 34]. Afterwards, dense classes of digraphs that are not tournaments have also been considered in the same context [45, 40]. Also, d-Correlation Clustering is known to admit a subexponential-time parameterized algorithm [24]. When d is not fixed, the problem is known not to admit a parameterized subexponential-time algorithm under the Exponential Time Hypothesis (ETH) [24].

2 Preliminaries

A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a fixed, finite alphabet. Let L be a parameterized problem. For an instance (x, k) of L, k is called the *parameter*. A *polynomial kernel* on L is an algorithm which, for any given instance (x, k) of L outputs, in polynomial time in the size of (x, k), an instance (x', k') of L with the following properties:

- (x', k') is a yes-instance $\iff$ (x, k) is a yes-instance.
- $|x'|, k' \leq h(k)$, where h is a polynomial function.

For further notions related to parameterized algorithm, we refer the reader to [15].

We follow the standard graph theory notations from [20]. Let $G = (V(G), E(G))$ be a graph and $x \in V(G)$. Then, $N(x)$ denotes the neighborhood of x, and $d(x) = |N(x)|$ its degree. If A is a subset of $V(G)$, then $d_A(x) = |N(x) \cap A|$ denotes the degree of x inside A. If A and B are two subsets of vertices in $V(G)$, then $E(A, B)$ denotes the set of edges with exactly one endpoint in A and one endpoint in B. A set of edges S is said to be a *d-cut* if $G - S$ has exactly d connected components.

A graph G is said to be *k-edge connected* (resp. *k-vertex connected*) if for any pair of vertices x and y in G, there exists k edge-disjoint (resp. vertex-disjoint) paths between x and y. For a graph G and two vertices x and y, a set of edges A is said to be an (x, y)-edge cut if $G - A$ does not contain any path between x and y. Likewise, a set of vertices S is said to be a (x, y)-vertex cut if $G - S$ does not contain any path between x and y. Let us cite the celebrated Menger's Theorem [44].

▶ **Theorem 11.** *Let G be a graph and x, y two vertices of G. The maximum number of vertex-disjoint (resp. edge-disjoint) paths between x and y is equal to the minimum size of a (x, y)-vertex cut (resp. (x, y)-edge cut).*

Let G be a graph and X a set of vertices, the graph obtained by *contracting X and keeping multiedges*, is the graph G' obtained from G by removing X, adding a new vertex x, and for every $v \in G$ such that v is adjacent to k vertices in X adding k multi-edges between x and v. Let U be a universe. Then, 2^U denotes all subsets of U and $\binom{U}{t}$ denotes all the subsets of size t of U. For an integer k, $[k]$ denotes the set $\{1, \dots, k\}$. For any real numbers a, b and c we write $a = b \pm c$ if $b - c \leq a \leq b + c$. The following easy observation will be used throughout the paper.

▶ **Observation 12.** *If c is a real in $[0, 1/2]$ and $x = 1 \pm c$, then $\frac{1}{x} = (1 \pm 2c)$.*

To construct estimators deterministically, we rely on the well known notion of k-wise independence, in the particular setting of permutations.

▶ **Definition 13.** *Let $n, k \in \mathbb{N}$. A family $\mathcal{S}$ of permutations of $\mathcal{S}_n$ is* k-wise independent *if, for any k-tuple of distinct elements $(x_1, \dots, x_k)$, the distribution $(f(x_1), f(x_2), \dots, f(x_k))$ where $f \in \mathcal{S}$ is chosen uniformly at random and the distribution $(f'(x_1), f'(x_2), \dots, f'(x_k))$ where $f' \in \mathcal{S}_n$ is chosen uniformly at random, are such that*

$$\sum_{(a_1,\dots,a_k)\in[n]^k} |Pr(f(x_1), \dots, f(x_k) = (a_1, \dots, a_k)) - Pr(f'(x_1), \dots, f'(x_k) = (a_1, \dots, a_k))| = 0.$$

Efficient construction of a k-wise independent family of permutations are known for $k = 2$ and $k = 3$ but open for $k > 4$ (see [4] for more details). In particular, there exists for every n, a family $\mathcal{S}(n)$ of $\mathcal{O}(n)$ pairwise (2-wise) independent permutations. This family will be sufficient for our derandomization purposes.

Throughout this paper, we will make an extensive use of Chebyshev's inequality:

▶ **Proposition 14.** *Let X be a random variable with expected value μ and variance σ^2. Then for any real number $k > 0$, $Pr[|X - \mu| \geq k\sigma] \leq \frac{1}{k^2}$.*

3 Edge-disjoint paths in everywhere dense graphs

In this section we design a linear vertex kernel for Edge-Disjoint Paths on everywhere α-dense graphs. We first present a polynomial time algorithm for the Edge-Disjoint Paths problem in α-dense graphs, when the number of demands is small (but still linear) compared to αn. Towards this, we start-by showing that highly edge-connected parts will always contain a solution to an Edge-Disjoint Paths instance.

▶ **Lemma 15.** *Let c be a constant between 0 and 1, and G be a graph on n vertices with minimum degree at least cn. For any pair of vertices x, y of G, if there exists a path between x and y, then there exists a path of length at most $4/c$.*

Proof. Let P be a shortest path between x and y. If there exists a vertex $u \in G$ such that u is adjacent to 4 vertices of P, then two of these vertices will be at distance at least 3 in the path. Denoting x_1 and x_2 these vertices, replacing the subpath of P between x_1 and x_2

by the path x_1ux_2 gives a path between x and y shorter than P, which is a contradiction. Therefore, the sum of the degree of the vertices of P is smaller than $4n$ and thus $|P|cn \leq 4n$ which implies $|P| \leq \frac{4}{c}$. ◀

▶ **Lemma 16.** *Let G be a graph with minimum degree αn, and cn edge-connected for some constant $c \leq \alpha/2$. Any instance of* Edge-Disjoint Paths *with $k \leq \frac{\alpha cn}{8}$ has a solution. Moreover, this solution can be found in polynomial time.*

Proof. Let $(G, (s_1, t_1), \cdots, (s_k, t_k))$ be an instance of the Edge-Disjoint Paths problem. For every pair (s_i, t_i), since G is cn-edge connected, there exists cn edge-disjoint paths $P_1, \dots, P_{cn}$ between s_i and t_i. Moreover, we can assume that all these paths are shorter than $\frac{8}{\alpha}$. Indeed, removing the edges of all but one path P_j leaves G with minimum degree at least $\alpha n - cn \geq \frac{\alpha n}{2}$ and Lemma 15 implies that P_j can actually be taken shorter than $\frac{8}{\alpha}$. This means that we can select a solution for the Edge-Disjoint Paths problem greedily using these paths. Indeed, each path is of length smaller than $\frac{8}{\alpha}$, so the path selected between s_i and t_i intersects at most $\frac{8}{\alpha}$ of the paths between s_j and t_j. Since $k \leq \frac{\alpha cn}{8}$, there is always one path available between s_i and t_i. ◀

For the proof of Lemma 16, we could have used a previously known result [31]. However, we still give the proof here, as it is simple on dense graphs, and helps in a complete understanding of the algorithm. The next lemma is an essential part of the proof. The goal is to find a partition of the vertices of $V(G)$ into a bounded number of parts, such that each part induces a graph with large edge-connectivity.

▶ **Lemma 3.** *For any real α between 0 and 1, there exists a constant $c \leq \alpha/2$ such that, if G is a graph on n vertices and minimum degree αn, then there exists a partition $\mathcal{P}$ of the vertices $V(G)$ into $g \leq \frac{2}{\alpha}$ subsets $V_1, \cdots, V_g$ such that for all $i \in [g]$:*

- *$G[V_i]$ is cn edge-connected.*
- *$G[V_i]$ has minimum degree $\frac{\alpha n}{2}$.*

Moreover, such a partition can be found in polynomial time.

Proof. Let t be an integer such that $\frac{\alpha}{(1-\alpha/3)^t} > 2/3$, and c be a sufficiently small constant such that $tc < \alpha/6$, $\alpha/2 \geq c$ and for all $i < t$:

$$cn < \frac{\alpha^2 n}{(1-\alpha/3)^{i-1}} \left(\frac{1}{1-\alpha/2} - \frac{1}{1-\alpha/3} \right)$$

We inductively build a sequence of partitions of $V(G)$: $\mathcal{P}_1, \dots, \mathcal{P}_t$. Each $\mathcal{P}_{i+1}$ is obtained from $\mathcal{P}_i$ by applying a set of operations. Further, either a part of $\mathcal{P}_i$ remains a part in $\mathcal{P}_{i+1}$ or breaks into several parts in $\mathcal{P}_{i+1}$. In particular, $\mathcal{P}_{i+1}$ is a *finer* partition than $\mathcal{P}_i$. Let each $\mathcal{P}_i$ consists of $V_1^i, \cdots, V_{l_i}^i$ as its parts. Throughout the proof these parts satisfy the following invariants. That is, for all $j \in [l_i]$:

Invariant 1: $G[V_j^i]$ has minimum degree $(\alpha - ci)n$.

Invariant 2: Either $G[V_j^i]$ is cn edge-connected; or every vertex of $v \in V_j^i$ has more than $\frac{\alpha}{(1-\alpha/3)^{i-1}}|V_j^i|$ neighbours in $G[V_j^i]$ (note that, $\frac{\alpha}{(1-\alpha/3)^{i-1}} \geq \alpha$ and thus, $G[V_j^i]$ is denser than G).

Note that, as we chose t such that $\frac{\alpha}{(1-\alpha/3)^t} > 2/3$, and c such that $tc < \alpha/2$, if the previous properties are satisfied, then $\mathcal{P}_t$ is the partition that we are looking for. Indeed, the second condition tells us that, if $G[V_j^t]$ is not cn-edge connected, then every vertex of V_j^t has more than $2/3|V_j^t|$ neighbors in $G[V_j^t]$. Since $|V_j^t| \geq (\alpha - ct)n \geq \alpha n/2$, it means that any pair of vertices in V_j^t have more than $\alpha n/6$ common neighbors in V_j^t, which implies that $G[V_j^t]$ cn-edge connected. Moreover, since $|V_j^t| \geq \alpha n/2$, this partition has less than $\frac{2}{\alpha}$ parts.

What remains to show is that indeed there exists a sequence of partitions of $V(G)$: $\mathcal{P}_1, \dots, \mathcal{P}_t$. We show the existence of the partition $\mathcal{P}_i$ by induction on i, setting $\mathcal{P}_1 = V(G)$ which trivially satisfies all the properties. Suppose now that we have constructed the partition $\mathcal{P}_i = V_1^1, \cdots, V_{l_i}^i$ for some $i < t$. For each $j \in l_i$, we define a partition of V_j^i into $H_j^1, \dots, H_j^{x_j}$ for some $x_j < (2/\alpha)$ as follows: If $G[V_j^i]$ is cn-edge connected, then $x_j = 1$ and $H_j^1 = V_j^i$. If not, let $H_j^1, \dots, H_j^{x_j}$ be the connected components of $G[V_j^i]$ after removing the edges of a cut of size smaller than cn. Note that every vertex has degree at least $(\alpha - ci)n - cn \geq \frac{\alpha n}{2}$ after removing the cut edges, which implies Invariant 1. This means that the size of each component is at least $\frac{\alpha n}{2}$. This means in particular that the number of components is smaller than $(2/\alpha)$. Moreover, let w be a vertex in one of the connected components, H_j^r, we know that the degree of w in $G[V_j^i]$ is greater than $\frac{\alpha}{(1-\alpha/3)^{i-1}}|V_j^i|$. Since the cut is of size cn, it means that the degree of w in $G[H_j^r]$ is greater than $\frac{\alpha}{(1-\alpha/3)^{i-1}}|V_j^i| - cn$. Since, there is at least *one other component*, we have that $|H_j^r| < |V_j^i| - \frac{\alpha n}{2} < (1 - \frac{\alpha}{2})|V_j^i|$. This means that the degree of w in $G[H_j^r]$ is greater than $\frac{\alpha}{(1-\alpha/3)^{i-1}}(\frac{1}{1-\alpha/2}|H_j^r|) - cn$, which by the choice of c is greater than $\frac{\alpha}{(1-\alpha/3)^{i}}|H_j^r|$. Finally, we take $\mathcal{P}_{i+1}$ as the union of all the H_j^r for all $j \in [l_i]$ and $r \in [x_j]$. That is, $\mathcal{P}_{i+1}$ consists of either a part from $\mathcal{P}_i$, or connected components of a part that has a cut of size smaller than cn. By the above description, it follows that $\mathcal{P}_i$ satisfies both the invariants. This completes the proof. ◀

▶ **Lemma 17.** *The* Edge-Disjoint Paths *problem can be solved in time $k^\rho n^{\mathcal{O}(1)}$ on everywhere α-dense graphs, when $k \leq \frac{\alpha c n}{16}$. Here, c is the constant defined in Lemma 3 and $\rho = 2^{\frac{2}{\alpha}}\frac{2}{\alpha}!$.*

Proof. Let $(G, (s_1, t_1), \dots, (s_k, t_k))$ be an instance of the Edge-Disjoint Paths problem in an everywhere α-dense graph G of size n, where $k \leq \frac{\alpha c n}{8}$. Let $\mathcal{P} = V_1, \dots, V_g$, $g \leq \frac{2}{\alpha}$, be the partition of $V(G)$ obtained by applying Lemma 3.

▷ Claim 18. If $(G, (s_1, t_1), \dots, (s_k, t_k))$ is a yes-instance of Edge-Disjoint Paths, then there exists a path system $\tilde{P}_1, \dots, \tilde{P}_k$, connecting s_i to t_i such that the intersection of any path $\tilde{P}_j$ with any V_i for $i \in [g]$ is a subpath (possibly empty) of $\tilde{P}_j$.

Proof. Let $(P_1, \dots, P_k)$ be a solution. For every $j \in [g]$, we say that $(P_1, \dots, P_k)$ satisfies the property $\mathcal{H}_j$ if $P_i \cap V_j$ is a subpath of P_i for every $i \in [k]$.

Suppose that the solution $(P_1, \dots, P_k)$ does not satisfy property $\mathcal{H}_j$. For every $i \in [k]$ denote by h_i and l_i, the first and the last vertex of P_i in V_j, respectively. If P_i does not intersect V_j, then we assign h_i and l_i to $\emptyset$. Furthermore, h_i could be equal to l_i. Observe that $(G[V_j], (h_1, l_1), \dots, (h_k, l_k))$ is an instance of Edge-Disjoint Paths with $k \leq \frac{\alpha c n}{16}$. By Lemma 16, there is a solution $(P_1', \dots, P_k')$ to this problem in $G[V_j]$. Let $(P_1^1, \dots, P_k^1)$ denote the solution obtained from $(P_1, \dots, P_k)$ by replacing each subpath of P_i from h_i to l_i by P_i'.

Clearly the solution $(P_1^1, \dots, P_k^1)$ satisfies property $\mathcal{H}_j$. Moreover, let us show that if $(P_1, \dots, P_k)$ satisfies property $\mathcal{H}_{j'}$ for some $j' \in [g]$ $j \neq j'$, then so does $(P_1^1, \dots, P_k^1)$. This would conclude our proof of the lemma, as it means we can apply the previous procedure for every $j \in [g]$, iteratively.

Let i be an index of $[k]$ and suppose that $P_i \cap V_{j'}$ is a subpath of P_i. We want to show that $P_i^1 \cap V_{j'}$ is also a subpath of P_i^1. If $P_i \cap V_{j'}$ is empty, then so is $P_i^1 \cap V_{j'}$ as the vertices of $P_i^1 \setminus P_i$ belong to V_j and $j \neq j'$. Suppose now that $P_i^1 \cap V_{j'}$ is a subpath and denote by a_i and b_i the first and the last vertex of this path. Remember that h_i and l_i denote the first and the last vertex of $P_i \cap V_j$. If $P_i \cap V_j$ is empty, then $P_i^1 = P_i$ and there is nothing to prove, so let us assume it is not. Since the subpath of P_i between a_i and b_i is in $V_{j'}$ it means that h_i and l_i do not belong to this subpath. Therefore we are in one of the following three cases.

- h_i and l_i appear before a_i on P_i
- h_i and l_i appear after b_i on P_i
- h_i appears before a_i on P_i and l_i after b_i

In the first two cases, $P_i^1 \cap V_{j'} = P_i \cap V_{j'}$, which is still a subpath of P_i^1. In the last case, $P_i^1 \cap V_{j'}$ becomes empty. This concludes the proof. ◁

Consider the graph G' obtained from G by contracting every part V_j of the partition $\mathcal{P}$ into one vertex v_j (keeping multi-edges). That is, although the number of vertices in G' is g, the number of parallel edges between v_i and v_j is same as the number of edges between V_i and V_j. Thus, there is a one-to-one correspondence between edges in G' and the edges between a pair of vertices $w_1 \in V_i$ and $w_2 \in V_j$ such that $i \neq j$. For every $i \in [k]$, let s_i' (resp. t_i') denote the vertex of G' corresponding to the part containing s_i (resp. t_i) in G. Notice that same pair of v_i and v_j could be assigned to several pairs of s_i and t_i. In fact, if both s_i and t_i belong to the same part, say V_j, then $s_i' = v_j$ and $t_i' = v_j$. In this case it just means that the path must be completely contained inside the graph $G[V_j]$.

▷ Claim 19. $(G, (s_1, t_1), \ldots, (s_k, t_k))$ is a yes-instance of Edge-Disjoint Paths if and only if $(G', (s_1', t_1'), \ldots, (s_k', t_k'))$ is a yes-instance of Edge-Disjoint Paths.

Proof. Forward direction follows from Claim 18. Indeed, as explained before, if there is a solution in G, then we can assume that this solution is such that the intersection of any path with any part V_j is a subpath. Therefore, contracting the V_i along these paths create paths in G' and these paths are a solution to the problem in G'. Suppose now that we have a solution $P_1', \ldots, P_k'$ to the Edge-Disjoint Paths problem in G'. For every i, let $u_1^i, \ldots, u_{r_i}^i$ denote the sequence of edge in P_i'. Note that each of these edge corresponds to a specific edge in G. For every $j \in [r_1]$ such that v_j is an inner vertex of P_i', let us define $a_j^i \in V(G)$ and $b_j^i \in V(G)$ as the extremities of the two edges among $u_1^i, \ldots, u_{r_i}^i$ which are incident to v_j. For the first vertex v_s of P_i', we define similarly a_s^i as s_i and b_s^i is the extremity of the only edge of P_i' adjacent to v_j. Likewise, we can define $a_t^i \in V(G)$ and $b_t^i \in V(G)$ for the last vertex v_t of the path. Overall, replacing each v_j by a path from a_j^i to b_j^i gives a path from s_i to t_i in G. However, for every $j \in [g]$, $(G[V_j], (a_1^i, b_1^i), \ldots, (a_k^i, b_k^i))$ defines an instance of Edge-Disjoint Paths. Since $G[V_j]$ satisfies the properties of Lemma 16, in polynomial time we can find a solution to our instance. For every $i \in [k]$ and $j \in [g]$, let Q_j^i denote the path from a_j^i to b_j^i in this solution. Finally, for each $i \in [k]$, let P_i denote the path obtained from P_1' by replacing each v_j by Q_j^i. Thus, $P_1, \ldots, P_k$ forms a solution to the instance $(G, (s_1, t_1), \ldots, (s_k, t_k))$, which in particular implies that such a solution exists. ◁

Claim 19 shows that it is enough to solve our problem on the instance $(G', (s_1', t_1'), \ldots, (s_k', t_k'))$. Let us now explain how to solve this problem in G'. Recall that G' is a graph on a finite (at most $\frac{2}{\alpha}$) number of vertices. In particular it means that there is at most $2^{\frac{2}{\alpha}} \frac{2}{\alpha}!$ different paths in G', where a path may appear multiple times[3]. (First, choose the subset of vertices that appear in the path and then guess the permutation of the chosen vertices). Thus, the number of paths is upper bounded by $\rho = 2^{\frac{2}{\alpha}} \frac{2}{\alpha}!$. Therefore, a solution to this problem consists of assigning to each of these paths an integer of value at most k, which denotes the number of requests that will be resolved using this path. It means that the number of possible "distributions" of the requests among these paths is upper bounded by k^{ρ}. Moreover, once we have chosen the distribution of the requests among these paths, then testing whether this

[3] Here we see a path as a sequence of vertices.

distribution is indeed a solution requires only to count the number of times each multi-edge is used. So in total, to find a solution to the problem in G', we only need to check the $\mathcal{O}(k^{\rho})$ possible distributions. Since, we can test each distribution in $n^{\mathcal{O}(1)}$ time, the running time of the algorithm follows. ◀

Lemma 17 implies the following result.

▶ **Theorem 2.** *Edge-Disjoint Paths admits an $\mathcal{O}(k)$ vertex kernel on everywhere α-dense graphs.*

Proof. Let $(G, (s_1, t_1), \ldots, (s_k, t_k))$ be an instance of Edge-Disjoint Paths. Further, let c be the constant defined in Lemma 3. If $k \leq \frac{\alpha c n}{16}$, then we apply Lemma 17 and solve the problem in time $k^{\mathcal{O}(\frac{2}{\alpha}!)} n^{\mathcal{O}(1)}$. Based on the answer of Lemma 17, we either return a solution or a trivial no-instance of the problem. However, now we have that $k \geq \frac{\alpha c n}{16}$, and hence $n \leq \frac{16k}{\alpha c} = \mathcal{O}(k)$. This concludes the proof. ◀

4 Conclusion

Inspired by the success of designing of PTASes and EPTASes for computationally intractable problems on everywhere dense graphs (every vertex has minimum degree at least αn, for some fixed constant $\alpha > 0$), in this paper we undertook a study for computationally intractable problems on dense graphs in the realm of Parameterized Complexity on dense graphs. We obtained linear kernels for Edge-Disjoint Paths, Edge Odd Cycle Transversal, Minimum Bisection, d-Way Cut, Multiway Cut and Multicut on everywhere dense graphs. Additionally, we obtained a cubic kernel for Vertex-Disjoint Paths on everywhere dense graphs. In addition to kernelization results, we obtained subexponential-time parameterized algorithms for Edge Odd Cycle Transversal, Minimum Bisection, and d-Way Cut. Finally, we showed how all of our results (as well as EPASes for these problems) can be de-randomized. Studying other NP-hard problems on dense graphs is an interesting research avenue. We conclude our paper with some concrete open problems.

1. Does Vertex-Disjoint Paths admit a linear vertex kernel on everywhere α-dense graphs?
2. Does Edge-Disjoint Paths and Vertex-Disjoint Paths admit an algorithm with running time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ on everywhere α-dense graphs?

References

1 Noga Alon, Wenceslas Fernandez de la Vega, Ravi Kannan, and Marek Karpinski. Random sampling and approximation of max-csps. *J. Comput. Syst. Sci.*, 67(2):212–243, 2003. doi:10.1016/S0022-0000(03)00008-4.

2 Noga Alon, Alan M. Frieze, and Dominic Welsh. Polynomial time randomized approximation schemes for tutte-gröthendieck invariants: The dense case. *Random Struct. Algorithms*, 6(4):459–478, 1995. doi:10.1002/rsa.3240060409.

3 Noga Alon, Daniel Lokshtanov, and Saket Saurabh. Fast FAST. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, pages 49–58, 2009.

4 Noga Alon and Shachar Lovett. Almost k-wise vs. k-wise independent permutations, and uniformity for general group actions. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 350–361, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

5 Gunnar Andersson and Lars Engebretsen. Property testers for dense constraint satisfaction programs on finite domains. *Random Struct. Algorithms*, 21(1):14–32, 2002. doi:10.1002/rsa.10041.

6 Sanjeev Arora, Alan M. Frieze, and Haim Kaplan. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. *Math. Program.*, 92(1):1–36, 2002. doi:10.1007/s101070100271.

7 Sanjeev Arora, David R. Karger, and Marek Karpinski. Polynomial time approximation schemes for dense instances of np-hard problems. *J. Comput. Syst. Sci.*, 58(1):193–210, 1999. doi:10.1006/jcss.1998.1605.

8 Cristina Bazgan, Wenceslas Fernandez de la Vega, and Marek Karpinski. Polynomial time approximation schemes for dense instances of minimum constraint satisfaction. *Random Struct. Algorithms*, 23(1):73–91, 2003. doi:10.1002/rsa.10072.

9 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009.

10 Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theor. Comput. Sci.*, 412(35):4570–4578, 2011.

11 Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. Multicut is FPT. *SIAM J. Comput.*, 47(1):166–207, 2018.

12 Yixin Cao, Jianer Chen, and Jia-Hao Fan. An $O(1.84^k)$ parameterized algorithm for the multiterminal cut problem. *Inf. Process. Lett.*, 114(4):167–173, 2014.

13 Jianer Chen, Yang Liu, and Songjian Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55(1):1–13, 2009.

14 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.

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

16 Marek Cygan, Pawel Komosa, Daniel Lokshtanov, Michal Pilipczuk, Marcin Pilipczuk, and Saket Saurabh. Randomized contractions meet lean decompositions. *CoRR*, abs/1810.06864, 2018. arXiv:1810.06864.

17 Marek Cygan, Stefan Kratsch, Marcin Pilipczuk, Michal Pilipczuk, and Magnus Wahlström. Clique cover and graph separation: New incompressibility results. *ACM Trans. Comput. Theory*, 6(2):6:1–6:19, 2014.

18 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Minimum bisection is fixed-parameter tractable. *SIAM J. Comput.*, 48(2):417–450, 2019.

19 Wenceslas Fernandez de la Vega and Marek Karpinski. Polynomial time approximation of dense weighted instances of MAX-CUT. *Random Struct. Algorithms*, 16(4):314–332, 2000.

20 Reinhard Diestel. *Graph theory.* Springer Publishing Company, Incorporated, 2018.

21 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013.

22 Uriel Feige. Faster fast(feedback arc set in tournaments). *CoRR*, abs/0911.5094, 2009. arXiv:0911.5094.

23 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

24 Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk, Michal Pilipczuk, and Yngve Villanger. Tight bounds for parameterized complexity of cluster editing with a small number of clusters. *J. Comput. Syst. Sci.*, 80(7):1430–1447, 2014. doi:10.1016/j.jcss.2014.04.015.

25 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization. Theory of parameterized preprocessing.* Cambridge University Press, Cambridge, 2019.

26 Fedor V. Fomin and Michal Pilipczuk. Subexponential parameterized algorithm for computing the cutwidth of a semi-complete digraph. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 505–516, 2013.

27 Alan M. Frieze and Ravi Kannan. The regularity lemma and approximation schemes for dense problems. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 12–20. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548459.

28 Ioannis Giotis and Venkatesan Guruswami. Correlation clustering with a fixed number of clusters. *Theory of Computing*, 2(13):249–266, 2006. doi:10.4086/toc.2006.v002a013.

29 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998. doi:10.1145/285055.285060.

30 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006.

31 Andreas Huck. A sufficient condition for graphs to be weaklyk-linked. *Graphs and Combinatorics*, 7(4):323–351, 1991.

32 Marek Karpinski. Polynomial time approximation schemes for some dense instances of np-hard optimization problems. *Algorithmica*, 30(3):386–397, 2001. doi:10.1007/s00453-001-0012-z.

33 Marek Karpinski and Warren Schudy. Linear time approximation schemes for the gale-berlekamp game and related minimization problems. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 313–322. ACM, 2009. doi:10.1145/1536414.1536458.

34 Marek Karpinski and Warren Schudy. Faster algorithms for feedback arc set tournament, kemeny rank aggregation and betweenness tournament. In *Algorithms and Computation - 21st International Symposium, ISAAC2010, Jeju Island, Korea, December 15-17, 2010, Proceedings, Part I*, pages 3–14, 2010.

35 Ken-ichi Kawarabayashi and Mikkel Thorup. The minimum k-way cut of bounded size is fixed-parameter tractable. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 160–169. IEEE Computer Society, 2011.

36 Ken-ichi Kawarabayashi and Paul Wollan. A shorter proof of the graph minor algorithm: the unique linkage theorem. 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 687–694. ACM, 2010.

37 Stefan Kratsch and Magnus Wahlström. Compression via matroids: A randomized polynomial kernel for odd cycle transversal. *ACM Trans. Algorithms*, 10(4):20:1–20:15, 2014.

38 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *J. ACM*, 67(3):16:1–16:50, 2020.

39 Daniel Lokshtanov, Pranabendu Misra, Michal Pilipczuk, Saket Saurabh, and Meirav Zehavi. An exponential time parameterized algorithm for planar disjoint paths. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1307–1316. ACM, 2020.

40 Jayakrishnan Madathil, Roohani Sharma, and Meirav Zehavi. A sub-exponential fpt algorithm and a polynomial kernel for minimum directed bisection on semicomplete digraphs. In *44th International Symposium on Mathematical Foundations of Computer Science MFCS 2009, Aachen, Germany, August 26-30, 2019*, 2019.

41 Dániel Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351(3):394–406, 2006.

42 Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. *SIAM J. Comput.*, 43(2):355–388, 2014.

43 Claire Mathieu and Warren Schudy. Yet another algorithm for dense max cut: go greedy. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 176–182. SIAM, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347102.

44 Karl Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 10(1):96–115, 1927.

45 Pranabendu Misra, Saket Saurabh, Roohani Sharma, and Meirav Zehavi. Sub-exponential time parameterized algorithms for graph layout problems on digraphs with bounded independence number. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, pages 35:1–35:19, 2018.

46 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

47 Marcin Pilipczuk, Michal Pilipczuk, and Marcin Wrochna. Edge bipartization faster than 2^k. *Algorithmica*, 81(3):917–966, 2019.

48 Neil Robertson and Paul D. Seymour. Graph minors XIII. the disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

49 René van Bevern, Andreas Emil Feldmann, Manuel Sorge, and Ondrej Suchý. On the parameterized complexity of computing graph bisections. In Andreas Brandstädt, Klaus Jansen, and Rüdiger Reischuk, editors, *Graph-Theoretic Concepts in Computer Science - 39th International Workshop, WG 2013, Lübeck, Germany, June 19-21, 2013, Revised Papers*, volume 8165 of *Lecture Notes in Computer Science*, pages 76–87. Springer, 2013.

A Definition of the studied problems

We now define all the problems mentioned in the paper.

EDGE-DISJOINT PATHS **Parameter:** k
Input: A graph G and a set of request pairs $(s_1, t_1), \dots, (s_k, t_k)$.
Question: Does there exist a set of paths $P_1, \dots, P_k$, between s_i and t_i, such that they are pairwise edge disjoint?

VERTEX-DISJOINT PATHS **Parameter:** k
Input: A graph G and a set of request pairs $(s_1, t_1), \dots, (s_k, t_k)$.
Question: Does there exist a set of paths $P_1, \dots, P_k$, between s_i and t_i, such that they are pairwise vertex disjoint?

EDGE ODD CYCLE TRANSVERSAL **Parameter:** k
Input: A graph G and an integer k.
Question: Does there exists $S \subseteq E(G)$ of size at most k such that $G - S$ is bipartite?

MINIMUM BISECTION **Parameter:** k
Input: A graph G and an integer k.
Question: Does there exists a partition (A, B) of $V(G)$ such that $||A| - |B|| \leq 1$ and $E(A, B) \leq k$?

MULTIWAY CUT **Parameter:** k
Input: A graph G, a set $T \subseteq V(G)$ and an integer k.
Question: Does there exists a set $S \subseteq E(G)$ of size at most k such that every vertex of T lies in a different connected component of $G - S$?

Multicut **Parameter:** k

Input: A graph G, a set of pairs $(s_i, t_i)_{i=1}^{\ell}$ and an integer k.

Question: Does there exists $S \subseteq E(G)$ of size at most k such that for every $i \in [\ell]$, vertices s_i and t_i lie in different connected components of $G - S$?

d-Way Cut **Parameter:** k

Input: A graph G and an integer k.

Question: Does there exists a set $S \subseteq E(G)$ of size at most k such that $G - S$ has at least d connected components?

Subgroup Membership in GL(2,Z)

Markus Lohrey
Universität Siegen, Germany

Abstract

It is shown that the subgroup membership problem for a virtually free group can be decided in polynomial time where all group elements are represented by so-called power words, i.e., words of the form $p_1^{z_1} p_2^{z_2} \cdots p_k^{z_k}$. Here the p_i are explicit words over the generating set of the group and all z_i are binary encoded integers. As a corollary, it follows that the subgroup membership problem for the matrix group $\mathsf{GL}(2, \mathbb{Z})$ can be decided in polynomial time when all matrix entries are given in binary notation.

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

Keywords and phrases free groups, virtually free groups, subgroup membership, matrix groups

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.51

Funding *Markus Lohrey*: Funded by DFG project LO 748/12-1.

1 Introduction

The subgroup membership problem (aka generalized word problem) for a group G asks whether for given group elements $g_0, g_1, \ldots, g_k \in G$, g_0 belongs to the subgroup $\langle g_1, \ldots, g_k \rangle$ generated by $g_1, \ldots, g_k$. To make this a well-defined computational problem, one has to fix an input representation of group elements. Here, a popular choice is to restrict to finitely generated (f.g. for short) groups. In this case, group elements can be encoded by finite words over a finite set of generators. The subgroup membership problem is one of the best studied problems in computational group theory. Let us survey some important results on subgroup membership problems.

For symmetric groups S_n, Sims [33] has developed a polynomial time algorithm for the uniform variant of the subgroup membership problem, where n is part of the input. In this paper, we always consider non-uniform subgroup membership problems, where we consider a fixed infinite f.g. group G. For a f.g. free group, the subgroup membership problem can be solved using Nielsen reduction (see e.g. [23]); a polynomial time algorithm was found by Avenhaus and Madlener [1]. In fact, in [1] it is shown that the subgroup membership problem for a f.g. free group is P-complete. Another polynomial time algorithm uses Stallings's folding procedure [34]; an almost linear time implementation can be found in [35]. An extension of Stallings's folding for fundamental groups of certain graphs of groups was developed in [15]. The folding procedure from [15] can be used to show that subgroup membership is decidable for right-angled Artin groups with a chordal independence graph. Moreover, Friedl and Wilton [10] used the results of [15] in combination with deep results from 3-dimensional topology in order to decide the subgroup membership problem for 3-manifold groups. Other extensions of Stallings's folding and applications to subgroup membership problems can be found in [16, 25, 31]. Using completely different (more algebraic) techniques, the subgroup membership problem has been shown to be decidable for polycyclic groups [2, 24] and f.g. metabelian groups [29, 30].

On the undecidability side, Mihaĭlova [26] has shown that the subgroup membership problem is undecidable for the direct product $F_2 \times F_2$ (where F_2 is the free group of rank two). This implies undecidability of the subgroup membership problem for many other groups, e.g., $\mathsf{SL}(4,\mathbb{Z})$ (the group of 4×4 integer matrices with determinant one) or the 5-strand braid group B_5. Rips [28] constructed hyperbolic groups with an undecidable subgroup membership problem.

Apart from the above mentioned result of Avenhaus and Madlener [1] for free groups, the authors are not aware of other precise complexity results for subgroup membership problems in infinite groups. The P-completeness result for free groups from [1] assumes that group elements are represented by finite words over the generators of the free group. In recent years, group theoretic decision problems have been also studied with respect to more succinct representations of group elements. For instance, the so-called compressed word problem, where the input group element is represented by a so-called straight-line program (a context-free grammar that produces exactly one string) has received a lot of attention; see [3, 20] for a survey. For the subgroup membership problem in free groups, Gurevich and Schupp studied in [12] a succinct variant, where input group elements are of the form $a_1^{z_1} a_2^{z_2} \cdots a_k^{z_k}$. Here, the a_i are from a fixed free basis of the free group and the z_i are binary encoded integers. Based on an adaptation of Stallings's folding, they show that this succinct membership problem can be solved in polynomial time. Then, Gurevich and Schupp proceed in [12] by showing that their succinct folding algorithm for free groups can be adapted so that it works for the free product $\mathbb{Z}/2\mathbb{Z} * \mathbb{Z}/3\mathbb{Z}$. The particular interest in this group comes from the fact that it is isomorphic to the modular group $\mathsf{PSL}(2,\mathbb{Z})$, which is the quotient of $\mathsf{SL}(2,\mathbb{Z})$ by $\langle -\mathsf{Id}_2 \rangle \cong \mathbb{Z}/2\mathbb{Z}$ (Id_2 is the 2×2 identity matrix). As an application of the succinct folding algorithm for $\mathbb{Z}/2\mathbb{Z} * \mathbb{Z}/3\mathbb{Z}$, Gurevich and Schupp show that the subgroup membership problem for $\mathsf{PSL}(2,\mathbb{Z})$ is decidable in polynomial time when all matrix entries are encoded in binary notation.

The polynomial time algorithm for the succinct membership problem for $\mathbb{Z}/2\mathbb{Z} * \mathbb{Z}/3\mathbb{Z}$ from [12] is tailored towards this group, and it is not clear how to adapt the algorithm to related groups. The latter is the goal of this paper. For this it turnes out to be useful to consider a more succinct representation of input elements for free groups. Recall that Gurevich and Schupp use words of the form $a_1^{z_1} a_2^{z_2} \cdots a_k^{z_k}$, where the integers z_i are given in binary notation and the a_i are generators from a free basis. Here, we represent group elements by so-called *power words* which were studied in [21] in the context of group theory. A power word has the form $p_1^{z_1} p_2^{z_2} \cdots p_k^{z_k}$, where as above the integers z_i are given in binary notation but the p_i are arbitrary words over the group generators. In [21] it was shown that the so-called power word problem (does a given power word represent the group identity?) for a f.g. free group F is AC^0-reducible to the ordinary word problem for F (and hence in logspace). In this paper, we prove that the power-compressed subgroup membership problem (i.e., the subgroup membership problem with all group elements represented by power words) for a free group can be solved in polynomial time by using a folding procedure à la Stallings (Theorem 12). This generalizes the above mentioned result of Gurevich and Schupp. At first sight, the step from power words of the form $a_1^{z_1} a_2^{z_2} \cdots a_k^{z_k}$ (with the a_i generators) to general power words as defined above looks not very spectacular. But apart from the quite technical details, the power-compressed subgroup membership problem has a major advantage over the restricted version of Gurevich and Schupp: we show that if G is a f.g. group and H is a finite index subgroup of G then the power-compressed subgroup membership problem for G is polynomial time reducible to the power-compressed subgroup membership problem for H (Lemma 13). Hence, the power-compressed subgroup membership problem for every f.g. virtually free group (a finite extension of a f.g. free group) can be solved in polynomial time. This result opens up new applications to matrix group algorithms. It is well-known that the group $\mathsf{GL}(2,\mathbb{Z})$ (the group of all 2×2 integer matrices with determinant ± 1) is f.g. virtually free. Moreover, given a matrix $A \in \mathsf{GL}(2,\mathbb{Z})$ with binary encoded entries one can compute a power word (over a fixed finite generating set of $\mathsf{GL}(2,\mathbb{Z})$) that represents A. Hence, the subgroup membership problem for $\mathsf{GL}(2,\mathbb{Z})$ with binary encoded matrix entries can be decided in polynomial time.

Related work. Related to the subgroup membership problem is the more general *rational subset membership problem.* A rational subset in a group G is given by a finite automaton, where transitions are labelled with elements of G; such an automaton accepts a subset of G in the natural way. In the rational subset membership problem for G the input consists of a rational subset $L \subseteq G$ and an element $g \in G$ and the question is, whether $g \in L$. This problem was shown to be decidable for free groups by Benois [5] via an automata saturation procedure that moreover can be implemented in cubic time [6]. Stallings's folding can be viewed as a special case of Benois's construction.

Rational subset membership problems (and special cases) for matrix groups are a very active research field. Some recent results can be found in [4, 7, 9, 18, 27]. Closest to our work is [4], where it is shown that the identity problem for $\mathsf{SL}(2,\mathbb{Z})$ (does the identity matrix belong to a finitely generated subsemigroup of $\mathsf{SL}(2,\mathbb{Z})$?) and the rational subset membership problem for $\mathsf{PSL}(2,\mathbb{Z})$ are NP-complete (when matrix entries are given in binary notation). For this, the authors of [4] use the ideas of Gurevich and Schupp [12]. In [7, 9], first steps towards $\mathsf{GL}(2,\mathbb{Q})$ are taken: in [9] the authors prove decidability of membership in so-called flat rational subsets of $\mathsf{GL}(2,\mathbb{Q})$, whereas [7] establishes the decidability of the full rational subset membership problem for the Baumslag-Solitar groups $\mathsf{BS}(1,q) < \mathsf{GL}(2,\mathbb{Q})$ with $q \geq 2$.

2 Preliminaries

General notations. For an integer $z \in \mathbb{Z}$ we define its signum as usual: $\mathsf{sign}(0) = 0$, and for $z > 0$, $\mathsf{sign}(z) = 1$ and $\mathsf{sign}(-z) = -1$. As usual, Σ^* denotes the set of all finite words over an alphabet Σ, ε denotes the empty word, and $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ is the set of all non-empty words. The length of a word w is denoted by $|w|$. The word $u \in \Sigma^*$ is a *factor* of the word $w \in \Sigma^*$ if $w = sut$ for some $s, t \in \Sigma^*$.

Groups. For a group G and a subset $A \subseteq G$, we denote with $\langle A \rangle$ the subgroup of G generated by A. It is the set of all products of elements from $A \cup A^{-1}$. We only consider *finitely generated (f.g.) groups* G, for which there is a finite set $A \subseteq G$ such that $G = \langle A \rangle$; such a set A is called a *finite generating set* for G. If $A = A^{-1}$ then we say that A is a *finite symmetric generating set* for G. Clearly, G is f.g. if and only if there exists a finite alphabet Γ and a surjective monoid homomorphism $\pi \colon \Gamma^* \to G$. We also say that the word $w \in \Gamma^*$ represents the group element $\pi(w)$. For words $u, v \in \Gamma^*$ we say that $u = v$ in G if $\pi(u) = \pi(v)$. Sometimes, we also identify a word $w \in \Gamma^*$ with the corresponding group element $\pi(w)$.

Fix a finite set Σ of symbols and let $\Sigma^{-1} = \{a^{-1} \mid a \in \Sigma\}$ be a set of formal inverses of the symbols in Σ with $\Sigma \cap \Sigma^{-1} = \emptyset$. Let $\Gamma = \Sigma \cup \Sigma^{-1}$. We define an involution on Γ^* by setting $(a^{-1})^{-1} = a$ for $a \in \Sigma$ and $(a_1 a_2 \cdots a_k)^{-1} = a_k^{-1} \cdots a_2^{-1} a_1^{-1}$ for $a_1, \ldots, a_k \in \Gamma$. A word $w \in \Gamma^*$ is called *freely reduced* or *irreducible* if it neither contains a factor aa^{-1} nor $a^{-1}a$ for $a \in \Sigma$. With $\mathsf{red}(\Gamma^*)$ we denote the set of all irreducible words. For every word $w \in \Gamma^*$ one obtains a unique irreducible word that is obtained from w by deleting factors aa^{-1} and $a^{-1}a$ ($a \in \Sigma$) as long as possible. We denote this word with $\mathsf{red}(w)$.

The *free group* generated by Σ, $F(\Sigma)$ for short, can identified with the set $\mathsf{red}(\Gamma^*)$ together with the multiplication defined by $u \cdot v = \mathsf{red}(uv)$ for $u, v \in \mathsf{red}(\Gamma^*)$. A group G that has a free subgroup of finite index in G is called *virtually free*.

3 Stallings's folding for power-compressed words

In this section we present our succinct version of Stallings's folding. We start with the definition of power words and power-compressed graphs. These graphs are basically finite automata where the transitions are labelled with power words. We prefer to use the the term "graph" instead of "automaton", since the former is more common in the literature on Stallings's folding.

A *power word* over an alphabet Σ is a sequence $(p_1, n_1)(p_2, n_2) \cdots (p_k, n_k)$ of pairs where $p_1, \ldots, p_n \in \Sigma^+$ and $n_1, \ldots, n_k \in \mathbb{N} \setminus \{0\}$. Such a power word represents the ordinary word $p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k}$ and we usually identify a power word with the word it represents. In the case of an alphabet $\Gamma = \Sigma \cup \Sigma^{-1}$ we may also allow negative exponents in a power word. Of course, p^{-n} stands for $(p^{-1})^n$. When a power word is part of the input for a computational problem, we always assume that the exponents n_i are given in binary notation, whereas the words p_i (also called the *periods* of the power word) are written down explicitly by listing all symbols in the words. Therefore, we define the input length $\|w\|$ of the power word $w = (p_1, n_1)(p_2, n_2) \cdots (p_k, n_k)$ as $\sum_{i=1}^{k} |p_i| + \log n_i$. A power word should be seen as a succinct representation of the word it represents.

Consider a f.g. group G with the finite generating set Σ. The *power-compressed subgroup membership problem* for G is the following problem:

input: Power words $w_0, w_1, \ldots, w_n$ over the alphabet $\Sigma \cup \Sigma^{-1}$.
question: Does g_0 belong to the subgroup $\langle g_1, \ldots, g_n \rangle \leq G$, where g_i is the group element represented by w_i?

The concrete choice of the finite generating set Σ has no influence on the complexity of the power-compressed subgroup membership problem: If Θ is another finite generating set, then every generator $a \in \Sigma \cup \Sigma^{-1}$ can be expressed as word $w_a \in (\Theta \cup \Theta^{-1})^*$. Hence, from a power word w over $\Sigma \cup \Sigma^{-1}$ one can compute a power word w' over $\Theta \cup \Theta^{-1}$ such that w and w' represent the same group element. For this, one only has to apply the homomorphism $a \mapsto w_a$ to all periods p of the power word w, which can be done in TC^0 [19].

The goal of this section is to show that the power-compressed subgroup membership problem can be decided in polynomial time for a f.g. free group. In Section 4 we will extend this result to f.g. virtually free groups.

Our main tool for solving the power-compressed subgroup membership problem for f.g. free groups is an extension of Stallings's folding procedure for power-compressed words. First we need some combinatorial results for words. Fix a finite alphabet Σ with the inverse alphabet Σ^{-1} for the rest of Section 3 and let $\Gamma = \Sigma \cup \Sigma^{-1}$.

3.1 Combinatorics on words

We fix an arbitrary linear order $<$ on Γ. In order to simplify notation later, it is convenient to require that $a < a^{-1}$ for every $a \in \Sigma$. With $\preceq$ we denote the lexicographic order with respect to $<$. Let $\Omega \subseteq \mathsf{red}(\Gamma^*)$ denote the set of all irreducible words w such that

- w is non-empty,
- w is cyclically reduced (i.e, w cannot be written as aua^{-1} for $a \in \Gamma$),
- w is primitive (i.e, w cannot be written as u^n for some $n \geq 2$),
- w is lexicographically minimal among all cyclic permutations of w and w^{-1} (i.e., $w \preceq uv$ for all $u, v \in \Gamma^*$ with $vu = w$ or $vu = w^{-1}$).

Note that $\Sigma \subseteq \Omega$ and $\Sigma^{-1} \cap \Omega = \emptyset$ (since $a < a^{-1}$ for $a \in \Sigma$). Since $w \in \Omega$ is irreducible and cyclically reduced, also every power w^n is irreducible. The following lemma can be found in [21, Lemma 11].

▶ **Lemma 1.** *Let $p, q \in \Omega$, $x, y \in \mathbb{Z}$ and let u be a factor of p^x and v a factor of q^y. If $uv = 1$ in $F(\Sigma)$ and $|u| = |v| \geq |p| + |q| - 1$, then $p = q$.*

We also need the following statement:

▶ **Lemma 2.** *If $p \in \Omega$, $u, v \in \Gamma^*$, $x \in \{-1, 1\}$ and $up^x v = pp$ then $x = 1$ and $u = \varepsilon$ or $v = \varepsilon$.*

Proof. First assume that $upv = pp$ such that $u \neq \varepsilon$ and $v \neq \varepsilon$. We obtain a factorization $p = qr$ such that $q \neq \varepsilon$, $r \neq \varepsilon$ and $p = rq = qr$. Hence, $q, r \in s^*$ for some string $s \in \Gamma^+$ (see e.g. [22, Proposition 1.3.2]), which implies that p is not primitive, a contradiction.

Now assume that $up^{-1}v = pp$. If $u = \varepsilon$ or $v = \varepsilon$ then $p = p^{-1}$ which implies $p \notin \mathsf{red}(R)$. If $u \neq \varepsilon$ and $v \neq \varepsilon$ then we obtain a factorization $p = qr$ such that $q \neq \varepsilon$, $r \neq \varepsilon$ and $p^{-1} = rq$. Hence, $qr = p = q^{-1}r^{-1}$, which implies $q = q^{-1}$ and $r = r^{-1}$. But the latter implies $q, r \notin \mathsf{red}(R)$ and hence $p \notin \mathsf{red}(R)$, a contradiction. ◀

3.2 Power-compressed graphs

A *power-compressed graph* is a tuple $\mathcal{G} = (V, E, \iota, \tau, \lambda, v_0)$, where V is the set of vertices, E is the set of edges ($V \cap E = \emptyset$), $\iota\colon E \to V$ maps an edge to its source vertex, $\tau\colon E \to V$ maps an edge to its target vertex, $\lambda\colon E \to \Gamma^+ \times (\mathbb{Z} \setminus \{0\})$ assigns to every edge its label, and v_0 is the so-called *base point.* Moreover, for every edge e such that $\iota(e) = u$, $\tau(e) = v$, and $\lambda(e) = (p, z)$ there is an inverse edge $e^{-1} \neq e$ such that $\iota(e^{-1}) = v$, $\tau(e^{-1}) = u$, $\lambda(e^{-1}) = (p, -z)$, and $(e^{-1})^{-1} = e$. When we describe a power-compressed graph we often specify for a pair of edges e, e^{-1} only one of them and implicitly assume the existence of its inverse edge. An edge e is called *short* if $\lambda(e) \in \Gamma \times \{-1, 1\}$, otherwise it is called *long.* If $\mathcal{G}$ only contains short edges, then $\mathcal{G}$ is called an *uncompressed graph*, or just *graph.* We define the input length of $\mathcal{G}$ as $|\mathcal{G}| = \sum_{e \in E} \|\lambda(e)\|$ (here, we view $\lambda(e) = (p, z)$ as a power word consisting of a single power).

A *path* in $\mathcal{G}$ is a sequence $\rho = [v_1, e_1, v_2, e_2, \dots, v_k, e_k, v_{k+1}]$ where $e_1, \dots, e_k \in E$, $\iota(e_i) = v_i$ and $\tau(e_i) = v_{i+1}$ for $1 \leq i \leq k$. If $v_i \neq v_j$ for all i, j with $1 \leq i < j \leq k + 1$ then ρ is called a *simple path.* If $v_1 = v_{k+1}$ then ρ is a *cycle.* If $v_i \neq v_j$ for all i, j with $1 \leq i < j \leq k$ and $v_1 = v_{k+1}$ then ρ is a *simple cycle.* Let $\iota(\rho) = v_1$ and $\tau(\rho) = v_{k+1}$. If $\lambda(e_i) = (p_i, z_i)$ then we define $\lambda(\rho)$ as the power word $(p_1, z_1)(p_2, z_2) \cdots (p_k, z_k)$. The path ρ is *oriented* if $\mathsf{sign}(z_i) = \mathsf{sign}(z_j)$ for all i, j. The path ρ is *without backtracking* if $e_{i+1} \neq e_i^{-1}$ for all $1 \leq i \leq k - 1$.

In the following, we identify a pair $(p, z) \in \Gamma^+ \times (\mathbb{Z} \setminus \{0\})$ with the power p^z. In particular, in an uncompressed graph every edge is labelled with a symbol from Γ. With a power-compressed graph $\mathcal{G}$ we can associate an uncompressed graph $\mathsf{decompress}(\mathcal{G})$ that is obtained by replacing in $\mathcal{G}$ every p^z-labelled edge e by a path ρ of short edges from $\iota(e)$ to $\tau(e)$ and such that $\lambda(\rho) = p^z$. Moreover, if $\iota(e) \neq \tau(e)$ then ρ is a simple path and if $\iota(e) = \tau(e)$ then ρ is a simple cycle.

A power-compressed graph $\mathcal{G} = (V, E, \iota, \tau, \lambda, v_0)$ should be viewed as an automaton over the alphabet Γ, where transition labels are succinct words of the form p^z with z given in binary notation: V is the set of states, an edge e corresponds to a transition from $\iota(e)$ to $\tau(e)$ with label $\lambda(e)$ and v_0 is the unique initial and final state. We denote with $L(\mathcal{G})$ the set of all words $w \in \Gamma^*$ accepted by the automaton $\mathcal{G}$. With $F(\mathcal{G})$ we denote the image of $L(\mathcal{G})$ in the free group $F(\Sigma)$. Since every edge of $\mathcal{G}$ has an inverse edge, it is easy to see that $F(\mathcal{G})$ is a subgroup of $F(\Sigma)$.

3.3 Folding uncompressed graphs

Before we continue with power-compressed graphs let us first explain Stallings's folding procedure [34] for uncompressed graphs, which is one of the most powerful techniques for subgroups of free groups. Let $\mathcal{G}$ and $\mathcal{H}$ be two uncompressed graphs as defined in Section 3.2. We say that $\mathcal{G}$ can be *folded* into $\mathcal{H}$ if there exist two edges $e \neq e'$ in $\mathcal{G}$ such that $\iota(e) = \iota(e')$ and $\lambda(e) = \lambda(e')$ and $\mathcal{H}$ is obtained from $\mathcal{G}$ by merging the two vertices $\tau(e)$ and $\tau(e')$ (note that we may have already $\tau(e) = \tau(e')$ in $\mathcal{G}$) into a single vertex and removing the edges e and e^{-1} (this is an arbitrary choice; we could also keep e and e^{-1} and remove e' and ${e'}^{-1}$) from the graph. One can easily show that $F(\mathcal{G}) = F(\mathcal{H})$ holds in this situation. Every vertex of $\mathcal{G}$ is mapped to a vertex of $\mathcal{H}$ in the natural way ($\tau(e)$ and $\tau(e')$ are mapped to the same vertex of $\mathcal{H}$). If a graph $\mathcal{G}$ cannot be folded further then we say that $\mathcal{G}$ is *folded.* In this case, $\mathcal{G}$ is a deterministic automaton and $w \in L(\mathcal{G})$ implies $\mathsf{red}(w) \in L(\mathcal{G})$.

To a given finite set of words $A = \{w_1, \ldots, w_n\} \subseteq \Gamma^+$ we can associate a so-called bouquet graph $\mathcal{B}(A)$ such that $F(\mathcal{B}(A)) = \langle g_1 \ldots, g_n\rangle \leq F(\Sigma)$, where $g_i = \mathsf{red}(w_i) \in F(\Sigma)$ is the free group element represented by w_i): to a non-empty word $w = a_1 a_2 \cdots a_k$, where $a_i \in \Gamma$, we associate the cycle graph $\mathcal{C}(w) = (\{v_0, \ldots, v_{k-1}\}, \{e_i^{\pm 1} \colon 1 \leq i \leq k\}, \iota, \tau, v_0)$, where $\iota(e_i) = v_{i-1}$, $\lambda(e_i) = a_i$, and $\tau(e_i) = v_{i \bmod k}$ for $1 \leq i \leq k$. Then we define the bouquet graph $\mathcal{B}(A)$ by merging in the disjoint union of the cycle graphs $\mathcal{C}(w_i)$ the base points.

Let $\mathcal{S}(A)$ be the graph obtained by folding $\mathcal{B}(A)$ as long as possible (the outcome of this procedure is in fact unique up to graph isomorphism). The graph $\mathcal{S}(A)$ is sometimes called the Stallings's graph for A. Note that as an automaton, $\mathcal{S}(A)$ is deterministic. The above discussion leads to the following crucial fact (see also [14] for a more detailed discussion):

▶ **Lemma 3.** *Let $g \in \mathsf{red}(\Gamma^*)$ be an irreducible word and hence an element of $F(\Sigma)$. Then g is accepted by $\mathcal{S}(A)$ if and only if $g \in \langle g_1 \ldots, g_n\rangle \leq F(\Sigma)$.*

3.4 Folding power-compressed graphs

Fix a power-compressed graph $\mathcal{G} = (V, E, \iota, \tau, \lambda, v_0)$ for the rest of this section and let P be the set of all words p such that $\lambda(e) = p^z$ for some $e \in E$ and $z \in \mathbb{Z} \setminus \{0\}$. Let us define the following numbers:

- $\alpha := \max\{|p| \colon p \in P\} \geq 1$,
- $\beta := 2\alpha - 1 \geq 1$,
- $\gamma := 2(\alpha + \beta) \geq 4$.

We say that $\mathcal{G}$ is *normalized* if

- $P \subseteq \Omega$ (where Ω is defined in Section 3.1), and
- for every $e \in E$, if e is long and $\lambda(e) = p^z$ then $|z| \geq \gamma$.

Let E_ℓ be the set of long edges of $\mathcal{G}$.

▶ **Lemma 4.** *From a given power-compressed graph $\mathcal{G}$ we can compute in polynomial time a normalized power-compressed graph $\mathcal{G}'$ such that $F(\mathcal{G}) = F(\mathcal{G}')$.*

Proof. We first modify $\mathcal{G}$ such that for every edge label $\lambda(e) = p^z$ we have $p \in \Omega$. This can be done in polynomial time by [21, Lemma 12] which states that a given power word w over the alphabet Γ can be transformed in polynomial time (in fact, even in logspace) into a power word w' over the alphabet Γ such that (i) all periods of w' belong to Ω and (ii) $w = w'$ in $F(\Sigma)$. We finally replace every long edge e with $\lambda(e) = p^z$ and $|z| < \gamma$ by a simple path (or simple cycle) ρ of short edges such that $\lambda(\rho) = p^z$. ◀

We say that $\mathcal{G}$ is *weakly folded* if none of the following two conditions A and B holds:

Condition A: There exist two (long or short) edges $e_1 \neq e_2$ such that $\iota(e_1) = \iota(e_2)$, $\lambda(e_1) = p^{z_1}$ and $\lambda(e_2) = p^{z_2}$ for some $p \in \Omega$ and $z_1, z_2 \in \mathbb{Z} \setminus \{0\}$ with $\mathsf{sign}(z_1) = \mathsf{sign}(z_2)$.

Condition B: There exist a long edge e with $\lambda(e) = p^z$ and a path ρ consisting of short edges such that $\iota(e) = \iota(\rho)$, $\lambda(\rho) = p^x$, $x \in \{-1, 1\}$, and $\mathsf{sign}(x) = \mathsf{sign}(z)$.

We say that $\mathcal{G}$ is *strongly folded* if the graph $\mathsf{decompress}(\mathcal{G})$ is folded in the sense of Section 3.3. Clearly, if $\mathcal{G}$ is strongly folded then $\mathcal{G}$ is also weakly folded.

▶ **Lemma 5.** *A given normalized power-compressed graph $\mathcal{G} = (V, E, \iota, \tau, \lambda, v_0)$ can be folded in polynomial time into a normalized and weakly folded power-compressed graph $\mathcal{G}'$. We have $F(\mathcal{G}) = F(\mathcal{G}')$.*

Proof. In order to estimate the complexity of our algorithm, we use two termination parameters: the number $|E_\ell|$ of long edges and the total number of edges $|E|$. The algorithm performs a sequence of folding steps that are explained below. In each step, the value $|E_\ell|$ will not increase. If $|E_\ell|$ does not change then $|E|$ will not increase, but if $|E_\ell|$ decreases then $|E|$ may increase by at most $\gamma - 1$. The situation becomes difficult because it may happen that in a folding step neither $|E_\ell|$ nor $|E|$ changes. We distinguish the following three types of folding steps, where $\mathcal{G} = (V, E, \iota, \tau, \lambda, v_0)$ is the power-compressed graph before the folding step and $\mathcal{G}' = (V', E', \iota', \tau', \lambda', v_0')$ is the power-compressed graph after the folding step.

decreasing (p-edge) fold: If condition A holds with $z_1 = z_2$ then we can merge $\tau(e_1)$ and $\tau(e_2)$ into a single vertex (let us call it v) and replace the two edges e_1 and e_2 by a single edge from $\iota(e_1) = \iota(e_2)$ to v with label p^{z_1}.
More formally: If we define $\equiv_V$ to be the smallest (with respect to inclusion) equivalence relation on V with $\tau(e_1) \equiv_V \tau(e_2)$ and $\equiv_E$ to be the smallest equivalence relation on E with $e_1 \equiv_E e_2$ then we can identify V' (respectively, E') with the set of equivalence classes $\{[v]_{\equiv_V} : v \in V\}$ (respectively, $\{[e]_{\equiv_E} : e \in V\}$). Moreover $\iota'([e]_{\equiv_E}) = [\iota(e)]_{\equiv_V}$, $\tau'([e]_{\equiv_E}) = [\tau(e)]_{\equiv_V}$, $\lambda'([e]_{\equiv_E}) = \lambda(e)$ (all these mappings are well-defined). The surjective mapping μ with $\mu(v) = [v]_{\equiv_V}$ is called the *merging function* associated with the merging step. Note that some of (or all) the vertices $\iota(e_1)$, $\tau(e_1)$, $\tau(e_2)$ can be equal.

nondecreasing (p-edge) fold: If condition A holds with (w.l.o.g.) $|z_1| < |z_2|$ then we can fold the two edges e_1 and e_2 by first setting $V' = V$, $E' = E$, $\tau' = \tau$, $\iota'(e_2) = \tau(e_1)$ and $\lambda'(e_2) = p^{z_2 - z_1}$. On all other arguments, ι' (respectively, λ') coincides with ι (respectively, λ). The resulting graph $\mathcal{G}'$ may be not normalized, namely if e_2 is long (in $\mathcal{G}'$) and $|z_2 - z_1| < \gamma$. In this case we replace e_2 by a simple path (or cycle, in case $\iota'(e_2) = \tau'(e_2)$) of fresh short edges from $\iota'(e_2)$ to $\tau'(e_2)$ spelling the word p^{z-x}. Note that after this modification we have $V \subseteq V'$ and $E \subseteq E'$. We define the merging function $\mu : V \to V'$ as the canonical inclusion mapping.

nondecreasing (p-path) fold: If the situation in condition B occurs, then we first set $V' = V$, $E' = E$, $\tau' = \tau$, $\iota'(e) = \tau(\rho)$ and $\lambda'(e) = p^{z-x}$. On all other arguments, ι' (respectively, λ') coincides with ι (respectively, λ). If in the resulting graph $\mathcal{G}'$, e is long and $|z - x| < \gamma$ then we replace the edge e by a simple path (or cycle) of short fresh edges spelling the word p^{z-x}. Again we define the merging function $\mu : V \to V'$ as the canonical inclusion mapping.

Note that each of the above folding steps simulates several folding steps in the corresponding uncompressed graph. Figure 1 shows some folding steps.

Assume we make a sequence of k folding steps, where $\mathcal{G}$ is the initial graph, $\mathcal{G}'$ is the final graph and μ_i ($1 \leq i \leq k$) is the merging function for the i-th folding step. Then we can define the composition $\mu = \mu_1 \circ \mu_2 \circ \cdots \circ \mu_k$ (where μ_1 is applied first); it maps every vertex

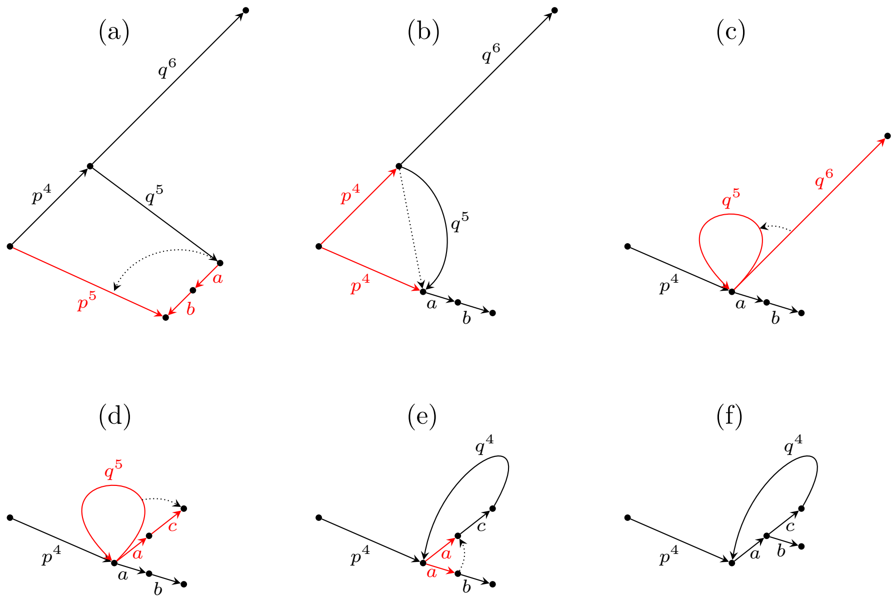


Figure 1 Some folding steps, where $p = ab \in \Omega$ and $q = ac \in \Omega$. We assume that $\gamma = 4$ and that all inverse edges are implicitly present. The edges involved in the folding steps are red; dotted arrows only indicate the direction of foldings and are not part of the graph.

- (a) to (b): nondecreasing p-path fold
- (b) to (c): decreasing p-edge fold
- (c) to (d): nondecreasing q-edge folds (the q^6-labelled edge coils once around the q^5-labelled loop and the remaining q-labelled edge is replaced by the two short edges labelled with a and c).
- (d) to (e): nondecreasing q-path fold
- (e) to (f): decreasing a-edge fold

The finally graph is weakly folded.

v of $\mathcal{G}$ to a vertex $\mu(v)$ of $\mathcal{G}'$. We then say that *vertex v is mapped to vertex $\mu(v)$ during the folding*. For two vertices u, v of $\mathcal{G}$ with $\mu(u) = \mu(v)$ we say that *u and v are merged during the folding*.

Note that every folding step preserve the property of being normalized. Clearly, a decreasing fold does not increase $|E_\ell|$ but decreases $|E|$ (and possibly $|E_\ell|$ in case e_1 and e_2 are long edges). Therefore, we can always perform decreasing folds if possible. A nondecreasing fold can reduce the number of long edges in which case the number of short edges increases by at most $\alpha \cdot (\gamma - 1)$. If a nondecreasing fold does not reduce the number of long edges then both $|E|$ and $|E_\ell|$ stay the same. Hence, the total number of decreasing folds is bounded by $|E| + \alpha \cdot \gamma \cdot |E_\ell|$. Bounding the number of nondecreasing folds is not so easy. If we just iteratively fold then we may obtain an exponential running time. In order to ensure termination in polynomial time, we arrange the folding steps as follows: Assume that $P = \{p_1, p_2, \ldots, p_n\}$. We say that the current graph if *folded with respect to* p_j if neither condition A nor condition B holds with $p = p_j$. For the following algorithm it is useful to consider the graph $\mathcal{G}_p$ where the edge set of $\mathcal{G}_p$ contains all long edges from E that are labelled with a power of p. In addition, $\mathcal{G}_p$ contains a p^1-labelled edge from u to v if $\mathcal{G}$ contains a path ρ of short edges from u to v and such that $\lambda(\rho) = p$ (note that $\mathcal{G}_p$ is in general not normalized). Such an edge should be only viewed as an abbreviation of the corresponding path ρ (which is unique if no decreasing folds are possible in $\mathcal{G}$).

Algorithm 1 (The main folding algorithm).

```
  Data: normalized power-compressed graph $\mathcal{G}$
1 $i := 1$
2 while true do
3     fold $\mathcal{G}$ with respect to $p_i$     /* this is explained in the main text     */
4     if $\mathcal{G}$ is weakly folded then
5         return $\mathcal{G}$
6     else
7         $i :=$ smallest $j$ such that $\mathcal{G}$ is not folded with respect to $p_j$
8     end
9 end
```

The main structure of the folding algorithm is shown in Algorithm 1. In the following, we always perform decreasing folds when possible without mentioning this explicitly.

We now explain how to fold the current graph $\mathcal{G}$ with respect to some $p = p_i$ (line 3 of Algorithm 1). We consider each connected component of the graph $\mathcal{G}_p$ separately. For the following consideration, we can assume that $\mathcal{G}_p$ is connected. We claim that $\mathcal{G}_p$ can be folded either into a simple oriented path or a simple oriented cycle. Moreover, if $\mathcal{G}_p$ is a tree then it is folded into a simple oriented path. The case that $\mathcal{G}_p$ consists of a single edge is clear. If $\mathcal{G}_p$ has more than one edge then we consider the following cases.

Case 1. $\mathcal{G}_p$ is a tree: Choose an edge e with $\iota(e) = u$ and $\tau(e) = v$ where v is a leaf. Let $\mathcal{G}'$ be the connected graph obtained from $\mathcal{G}_p$ by removing e, e^{-1} and v. By induction, $\mathcal{G}'$ can be folded into a simple oriented path $\rho = [v_1, e_1, v_2, e_2, \ldots, v_k, e_k, v_{k+1}]$, where w.l.o.g. $\lambda(e_i) = p^{a_i}$ with $a_i > 0$ for all i. Let v_i be the vertex to which $u = \iota(e)$ is mapped during the folding. Assume that $\lambda(e) = p^b$ with $b > 0$ (the case $b < 0$ is analogous). If there exists $j \geq i$ such that $b = a_i + \cdots + a_j$ then nothing has to be done (the vertex v is mapped to v_{j+1} during the folding). If there is no such j then we have to add a vertex to the path: if there is $j \geq i$ such that $a_i + \cdots + a_{j-1} < b < a_i + \cdots + a_j$ then we replace the edge e_j by an edge from v_j to a fresh vertex v' and an edge from v' to v_{j+1}. The label of the first edge is $p^{b-(a_i+\cdots+a_{j-1})}$ and the label of the second edge is $p^{a_i+\cdots+a_j-b}$. If $a_i + \cdots + a_k < b$ then we add an edge from v_{k+1} to the new vertex v' with label $p^{b-(a_i+\cdots+a_k)}$. In both cases the vertex $v = \tau(e)$ is mapped to the new vertex v' during the folding.

Case 2. $\mathcal{G}_p$ is not a tree. Then we choose an edge e such that $\mathcal{G}' := \mathcal{G}_p \setminus e$ (the graph obtained from $\mathcal{G}_p$ by removing the edges e and e^{-1}) is still connected.

Case 2.1. $\mathcal{G}'$ is folded into a simple oriented path $\rho = [v_1, e_1, v_2, e_2, \ldots, v_k, e_k, v_{k+1}]$, where w.l.o.g. $\lambda(e_i) = p^{a_i}$ with $a_i > 0$ for all i. Let v_i (respectively, v_l) be the vertex to which $\iota(e)$ (respectively, $\tau(e)$) is mapped during the folding. We proceed as in case 1. In case there exists $j \geq i$ with $b = a_i + \cdots + a_j$ then we additionally merge v_{j+1} and v_l (we may have already $v_{j+1} = v_l$ in which case we end up with a simple oriented path). If there is no such j then we add a new vertex v' to the path as in case 1 and merge v' with v_l. In both cases we get a simple oriented path to which a simple oriented cycle is attached. We then fold the two ends of the simple path onto the cycle (by coiling them around the cycle) and obtain a simple oriented cycle.

Case 2.2. $\mathcal{G}'$ is folded into a simple oriented cycle $\mathcal{C}$. We proceed analogously to case 2.1. We either obtain a single simple oriented cycle or two simple oriented cycles ρ_1 and ρ_2 that are glued together in a single vertex v (to see this, one can first remove an arbitrary edge from the cycle $\mathcal{C}$, which yields a simple oriented path, then carries out the construction from case 2.1 and finally adds the removed edge again). Such a pair of cycles can be replaced by a single cycle as follows: Let $\lambda(\rho_1) = p^{z_1}$ and $\lambda(\rho_2) = p^{z_2}$ with $z_1, z_2 > 0$. Then one can replace the two cycles by a single cycle ρ with $\lambda(\rho) = z := \gcd(z_1, z_2)$ (folding the cycles into a single cycle actually corresponds to Euclid's algorithm). Of course, we also have to map the vertices of ρ_1 and ρ_2 into the cycle ρ. For this we start with a p^z-labelled loop at vertex v. If $v' \neq v$ is a vertex belonging to say ρ_1 and the simple path from v to v' on the cycle ρ_1 is labelled with p^y, $y > 0$, then we compute $r := y \bmod z$ and subdivide the loop into an edge from v to v' with label p^r and an edge from v' back to v with label p^{z-r}. We continue in this way with the other vertices on ρ_1 and ρ_2.

Let $\mathcal{H}_p$ be the outcome of the above procedure. It is a disjoint union of simple oriented paths and simple oriented cycles and hence folded with respect to p. The running time of the computations in case 1 and 2 is polynomial in $\|\mathcal{G}_p\|$ and due to the recursion this running time has to be charged for every edge of $\mathcal{G}_p$. Recall that edges labelled with p^1 in $\mathcal{H}_p$ actually correspond to paths of short edges in the original graph $\mathcal{G}$. This concludes the description of line 3 in Algorithm 1.

It remains to argue that we make only polynomially many iterations of the while-loop in Algorithm 1. For this assume that the current graph (call it $\mathcal{G}'$) is folded with respect to p_i and that we fold the graph with respect to some p_j with $j > i$. Let us denote the sequence of folding steps with respect to p_j with $\mathcal{F}_j$ and let $\mathcal{G}''$ be the graph after the execution of $\mathcal{F}_j$. Moreover, assume that $\mathcal{G}''$ is no longer folded with respect to p_i. We argue that this implies that during the execution of $\mathcal{F}_j$ we made progress in the sense that $|E|$ or $|E_\ell|$ decreases. Since $\mathcal{G}'$ is folded with respect to p_i but $\mathcal{G}''$ is not, we must have $\mathcal{G}'_{p_i} \neq \mathcal{G}''_{p_i}$. But this implies that either $|E|$ or $|E_\ell|$ must decrease during $\mathcal{F}_j$. Otherwise we only make non-decreasing p_j-edge and p_j-path folds that do not eliminate long edges. Such folds only change the source and target vertices of p_j^z-labelled long edges, which does not modify the graph $\mathcal{G}'_{p_i}$.

Since we have already bounded the number of decreasing folds by $|E| + \alpha \cdot \gamma \cdot |E_\ell|$ and the number of long edges never increases, the index i in Algorithm 1 can only decrease a polynomial number of times (more precisely: $|E| + (\alpha \cdot \gamma + 1) \cdot |E_\ell|$ times). ◀

It remains to convert a weakly folded power-compressed graph in polynomial time into a strongly folded power-compressed graph. For this, we need several lemmas.

▶ **Lemma 6.** *Let $\mathcal{G}$ be an uncompressed graph and assume that $\mathcal{G}$ is folded into $\mathcal{G}'$ by a sequence of folding steps. If thereby two vertices u and v of $\mathcal{G}$ are merged to a single vertex of $\mathcal{G}'$, then there must exist a path ρ without backtracking in $\mathcal{G}$ from u to v such that $\lambda(\rho) = 1$ in $F(\Sigma)$.*

Proof. The lemma can be shown by a straightforward induction over the number of folding steps from $\mathcal{G}$ to $\mathcal{G}'$. Note that if two different vertices v_1 and v_2 of an uncompressed graph are merged in a single folding step, then there exist two different edges $e_1 \neq e_2$ such that $\iota(e_1) = \iota(e_2)$, $\tau(e_1) = v_1$, $\tau(e_2) = v_2$, and $\lambda(e_1) = \lambda(e_2) = a$ for some $a \in \Gamma$. Hence, the path $\rho = [v_1, e_1^{-1}, \iota(e_1), e_2, v_2]$ is without backtracking and satisfies $\lambda(\rho) = a^{-1}a = 1$ in $F(\Sigma)$. ◀

▶ **Lemma 7.** *Consider a word $p^y w q^z \in \Gamma^*$ such that the following hold, where $a = \mathsf{sign}(y)$ and $b = \mathsf{sign}(z)$:*

- *$p, q \in P$,*
- *$w \in \mathsf{red}(\Gamma^*)$,*
- *$|y| = |z| = \alpha + \beta = \gamma/2 \geq 2$,*

- *if $w = \varepsilon$, then $p \neq q$ or $a = b$,*
- *p^{-a} is not a prefix of w and q^{-b} is not a suffix of w.*

Then $\mathsf{red}(p^y w q^z)$ starts with a non-empty prefix of p^a and ends with a non-empty suffix of q^b.

Proof. Since p^y, w and q^z are irreducible, reductions can only occur at the two borders between p^y, w and q^z. Let us start to reduce the word $p^y w q^z$. Since p^{-a} is not a prefix of w and q^{-b} is not a suffix of w, the reductions at the two borders can only consume $|p| - 1 < \alpha$ symbols from the prefix of w and $|q| - 1 < \alpha$ symbols from the suffix of w. If w is not completely cancelled during the reduction, we obtain an irreducible word of the form $p^{y-a} r s t q^{z-b}$, where r is a prefix of p^a, t is a suffix of q^b and s is a non-empty factor of w. The conclusion of the lemma clearly holds in this case.

Let us now assume that w is completely cancelled during the reduction. Since w is irreducible, we obtain factorizations $w = u^{-1}v^{-1}$, $p^a = ru$, and $q^b = vs$. Moreover, $p^y w q^z$ is reduced to $p^{y-a} r s q^{z-b}$. We distinguish several cases:

- $p \neq q$: then the reduction of $p^{y-a} r s q^{z-b}$ can proceed for at most $|p| + |q| - 2 < \beta$ steps (otherwise we obtain a contradiction to Lemma 1).
- $p = q$ and $|r| \neq |s|$: then the reduction of $p^{y-a} r s q^{z-b}$ can proceed for at most $|p| - 1 < \alpha$ steps (otherwise we obtain a contradiction to Lemma 2).
- $p = q$, $|r| = |s|$, and $a = b$: then the reduction of $p^{y-a} r s q^{z-b}$ can proceed for at most $|r| \leq \alpha$ steps (otherwise p would be not cyclically reduced).
- $p = q$, $|r| = |s|$, and $a = -b$: w.l.o.g. assume that $y > 0$ and $z < 0$. We obtain $p = ru$ and $p^{-1} = vs$, i.e., $ru = s^{-1}v^{-1}$. Since $|r| = |s| = |s^{-1}|$ we have $r = s^{-1}$ and $u = v^{-1}$. Therefore $w = u^{-1}v^{-1} = u^{-1}u$. Since $w \in \mathsf{red}(\Gamma^*)$, we must have $w = \varepsilon$. But we have excluded this case in the assumptions of the lemma.

In total, the reduction of $p^y w q^z$ consumes strictly less than $\alpha + \beta = \gamma/2$ symbols from p^y as well as from q^z. Hence, $\mathsf{red}(p^y w q^z)$ starts with a non-empty prefix of p^a and ends with a non-empty suffix of q^b. ◀

▶ **Lemma 8.** *Let $w = s p_1^{z_1} w_1 p_2^{z_2} w_2 \cdots p_{k-1}^{z_{k-1}} w_{k-1} p_k^{z_k} t$ be a word with $k \geq 2$ and let $a_i = \mathsf{sign}(z_i)$. Assume that the following conditions hold:*

- *$p_1, \ldots, p_k \in P$,*
- *$z_1, \ldots, z_k \in \mathbb{Z}$,*
- *$|z_1|, |z_k| \geq \alpha + \beta = \gamma/2$,*
- *$|z_2|, \ldots, |z_{k-1}| \geq \gamma$,*
- *$w_1, \ldots, w_{k-1} \in \mathsf{red}(\Gamma^*)$,*
- *s is a suffix of $p_1^{a_1}$, t is a prefix of $p_k^{a_k}$,*
- *if $w_i = \varepsilon$, then $p_i \neq p_{i+1}$ or $a_i \neq -a_{i+1}$ $(1 \leq i \leq k-1)$,*
- *$p_i^{-a_i}$ is not a prefix of w_i and $p_{i+1}^{-a_{i+1}}$ is not a suffix of w_i $(1 \leq i \leq k-1)$.*

Then $w \neq 1$ in $F(\Sigma)$, i.e., $\mathsf{red}(w) \neq \varepsilon$.

Proof. For $1 \leq i \leq k$ let c_i be such that $|c_i| = \gamma/2$ and $\mathsf{sign}(c_i) = a_i$. Let $u_i = p_i^{c_i} w_i p_{i+1}^{c_{i+1}}$ for $1 \leq i \leq k-1$. We can reduce $w = s p_1^{z_1 - c_1} u_1 p_2^{z_2 - 2c_2} u_2 \cdots p_{k-1}^{z_{k-1} - 2c_{k-1}} u_{k-1} p_k^{z_k - c_k} t$ to

$$w' := s p_1^{z_1 - c_1}\ \mathsf{red}(u_1)\ p_2^{z_2 - 2c_2}\ \mathsf{red}(u_2) \cdots p_{k-1}^{z_{k-1} - 2c_{k-1}}\ \mathsf{red}(u_{k-1})\ p_k^{z_k - c_k} t.$$

By Lemma 7, $\mathsf{red}(u_i)$ starts with a non-empty prefix of $p_i^{a_i}$ and ends with a non-empty suffix of $p_{i+1}^{a_{i+1}}$. This implies that w' is irreducible and non-empty, which shows $w \neq 1$ in $F(\Sigma)$. ◀

We also need the following variant of Lemma 8.

▶ **Lemma 9.** *Let $w = sp_1^{z_1} w_1 p_2^{z_2} w_2 \cdots p_k^{z_k} w_k$ be a word with $k \geq 1$ and let $a_i = \mathsf{sign}(z_i)$. Assume that the following conditions hold:*

- $p_1, \ldots, p_k \in P$,
- $z_1, \ldots, z_k \in \mathbb{Z}$,
- $|z_1| \geq \alpha + \beta = \gamma/2$,
- $|z_2|, \ldots, |z_k| \geq \gamma$,
- $w_1, \ldots, w_k \in \mathsf{red}(\Gamma^*)$,
- *s is a suffix of $p_1^{a_1}$,*
- *if $w_i = \varepsilon$, then $p_i \neq p_{i+1}$ or $a_i \neq -a_{i+1}$ $(1 \leq i \leq k-1)$,*
- *$p_i^{-a_i}$ is not a prefix of w_i $(1 \leq i \leq k)$ and $p_{i+1}^{-a_{i+1}}$ is not a suffix of w_i $(1 \leq i \leq k-1)$.*

Then $w \neq 1$ in $F(\Sigma)$, i.e., $\mathsf{red}(w) \neq \varepsilon$.

Proof. The proof is almost the same as for Lemma 8. For $1 \leq i \leq k$ let c_i be such that $|c_i| = \gamma/2$ and $\mathsf{sign}(c_i) = a_i$. Let $u_i = p_i^{c_i} w_i p_{i+1}^{c_{i+1}}$ for $1 \leq i \leq k-1$ and $u_k = p_k^{a_k} w_k$. We can reduce $w = sp_1^{z_1-c_1} u_1 p_2^{z_2-2c_2} u_2 \cdots p_{k-1}^{z_{k-1}-2c_{k-1}} u_{k-1} p_k^{z_k-c_k-1} u_k$ to

$$w' := sp_1^{z_1-c_1}\ \mathsf{red}(u_1)\ p_2^{z_2-2c_2}\ \mathsf{red}(u_2) \cdots p_{k-1}^{z_{k-1}-2c_{k-1}}\ \mathsf{red}(u_{k-1})\ p_k^{z_k-c_k-1}\ \mathsf{red}(u_k).$$

By Lemma 7, every $\mathsf{red}(u_i)$ with $1 \leq i \leq k-1$ starts with a non-empty prefix of $p_i^{a_i}$ and ends with a non-empty suffix of $p_{i+1}^{a_{i+1}}$. Moreover, $\mathsf{red}(u_k)$ starts with a non-empty prefix of $p_k^{a_k}$ (since $p_k^{-a_k}$ is not a prefix of w_k). This implies that w' is irreducible and non-empty, which shows $w \neq 1$ in $F(\Sigma)$. ◀

▶ **Lemma 10.** *A given normalized and weakly folded power-compressed graph $\mathcal{G}$ can be folded in polynomial time into a strongly folded power-compressed graph $\mathcal{G}'$. We have $F(\mathcal{G}) = F(\mathcal{G}')$.*

Proof. We first construct a power-compressed graph $\mathcal{H}$ by partially decompressing $\mathcal{G}$. Consider a long edge e in $\mathcal{G}$. Let $\iota(e) = u$, $\tau(e) = v$ and $\lambda(e) = p^z$. W.l.o.g. assume that $z > 0$. Since $\mathcal{G}$ is normalized, we have $z \geq \gamma$. We then replace e by

- a simple path ρ_1 of new short edges going from u to a new vertex u' and such that $\lambda(\rho_1) = p^{\gamma/2} = p^{\alpha+\beta}$,
- a new edge from u' to another new vertex v' with label $p^{z-\gamma}$ (if $z = \gamma$ then $u' = v'$ and the new edge is not present), and
- a simple path ρ_2 of new short edges going from v' to v and such that $\lambda(\rho_2) = p^{\gamma/2} = p^{\alpha+\beta}$.

We then fold $\mathcal{H}$ as long as possible. By Lemmas 6, 8 and 9 we can thereby only fold short edges. In other words: if $\mathcal{H}' = \mathsf{decompress}(\mathcal{H})$ (which is the same as $\mathsf{decompress}(\mathcal{G})$) then a vertex of $\mathcal{H}'$ that arises from decompressing a long edge of $\mathcal{H}$ cannot be merged with another vertex during the folding. To see this, assume the contrary: let u be a vertex of $\mathcal{H}'$ that arises from decompressing a long edge of $\mathcal{H}$ and that is merged with a vertex $v \neq u$ during the folding. By Lemma 6 there must exist a path ρ in $\mathcal{H}'$ from u to v without backtracking such that $\lambda(\rho) = 1$ in $F(\Sigma)$. But since $\mathcal{G}$ is weakly folded the word $\lambda(\rho)$ must be a word w as considered in Lemma 8 (if also v arises from decompressing a long edge of $\mathcal{H}$) or Lemma 9 (if v is already a vertex in $\mathcal{H}$). The w_i in Lemma 8 (resp., Lemma 9) correspond to the maximal subpaths of ρ consisting of short edges and the $p_i^{z_i}$ correspond to the long edges on the path). Hence, $\lambda(\rho) \neq 1$ in $F(\Sigma)$ which is a contradiction.

By the above consideration, if we fold short edges in $\mathcal{H}$ as long as possible we obtain a strongly folded graph $\mathcal{G}'$ which proves the lemma. ◀

Lemmas 4, 5 and 10 finally yield the main technical result of Section 3.4:

▶ **Corollary 11.** *A given power-compressed graph $\mathcal{G}$ can be folded in polynomial time into a strongly folded power-compressed graph $\mathcal{G}'$. We have $F(\mathcal{G}) = F(\mathcal{G}')$.*

3.5 Power-compressed subgroup membership problem for free groups

We can now show the main result of Section 3:

▶ **Theorem 12.** *The power-compressed subgroup membership problem for a f.g. free group can be solved in polynomial time.*

Proof. Let $w_0, w_1, \ldots, w_n$ be the input power words. We construct from $w_1, \ldots, w_n$ a power-compressed bouquet graph in the same way as in Section 3.3 for uncompressed graphs: to a non-empty power word $w = p_1^{z_1} p_2^{z_2} \cdots p_k^{z_k}$ we associate the power-compressed cycle graph $\mathcal{C}(w) = (\{v_0, \ldots, v_{k-1}\}, \{e_i^{\pm 1} \colon 1 \leq i \leq k\}, \iota, \tau, v_0)$, where $\iota(e_i) = v_{i-1}$, $\lambda(e_i) = p_i^{z_i}$, and $\tau(e_i) = v_{i \bmod k}$. We then construct the power-compressed bouquet graph $\mathcal{B}$ by taking the disjoint union of $\mathcal{C}(w_1), \ldots, \mathcal{C}(w_n)$ and then merging their base points. Using Corollary 11 we can fold $\mathcal{B}$ in polynomial time into a strongly folded power-compressed graph $\mathcal{G}$. Let v_0 be its base point. As explained at the end of Section 3.2 we can view $\mathcal{G}$ as a finite automaton, where transitions are labelled with succinct words of the form p^z with z given in binary notation. By Lemma 3, $\mathcal{G}$ accepts an irreducible word $g \in \mathsf{red}(\Gamma^*)$ if and only if g represents an element from $\langle g_1, \ldots, g_n \rangle \leq F(\Sigma)$ (where w_i represents the group element g_i). Since $\mathcal{G}$ is strongly folded, it is a deterministic automaton in the sense that the labels of two outgoing transitions of a state do not have a non-empty common prefix.

For the rest of the proof it is convenient to switch from power words to straight-line programs. A straight-line program is a context-free grammar $\mathcal{A}$ that produces exactly one word that is denoted with $\mathrm{val}(\mathcal{A})$. By repeated squaring, our given power word w_0 can be easily transformed in polynomial time into an equivalent straight-line program. Moreover, from a given straight-line program $\mathcal{A}$ over the alphabet $\Gamma = \Sigma \cup \Sigma^{-1}$ one can compute in polynomial time a new straight-line program $\mathcal{A}'$ such that $\mathrm{val}(\mathcal{A}') = \mathsf{red}(\mathrm{val}(\mathcal{A}))$; see [20, Theorem 4.11]. Hence, we can compute in polynomial time a straight-line program $\mathcal{A}'$ for $\mathsf{red}(w_0)$. The transition labels of the automaton $\mathcal{G}$ can be also transformed into equivalent straight-line programs; such automata with straight-line compressed transition labels were investigated in [13]. It remains to check in polynomial time whether the deterministic automaton $\mathcal{G}$ accepts $\mathrm{val}(\mathcal{A}')$. This is possible in polynomial time by [13, Theorem 1]. ◀

4 Power-compressed subgroup membership for virtually free groups

A main advantage of the power-compressed subgroup membership is that its complexity is preserved under finite index group extensions. The proof of the following lemma follows [11], where it is shown that the complexity of the (ordinary) subgroup membership problem is preserved under finite index group extensions. In order to extend this result to the power-compressed setting, we make us of the conjugate collection process for power words from [21, Theorem 6].

▶ **Lemma 13.** *Let G be a fixed f.g. group and H a fixed subgroup of finite index in G (thus, H must be f.g. as well). The power-compressed subgroup membership problem for G is polynomial time reducible to the power-compressed subgroup membership problem for H.*

Proof. Using the following standard trick we can assume that H is a normal subgroup of finite index in G: Let N be the intersection of all conjugate subgroups $g^{-1}Hg$. Then $N \leq H$ and N has still finite index in G (the later is a well-known fact). Since $N \leq H$, the power-compressed subgroup membership problem for N is polynomial time reducible to the power-compressed subgroup membership problem for H. Hence, it suffices to show that the power-compressed subgroup membership problem for G is polynomial time reducible to the power-compressed subgroup membership problem for N.

By the above consideration, we can assume that H is a normal subgroup of finite index in G. Let us fix a symmetric generating Θ for H and let $R \subseteq G$ be a (finite) set of coset representatives for H with $1 \in R$. Then $\Sigma := \Theta \cup (R \setminus \{1\})$ generates G. On R we can define the structure of the quotient group G/H by defining $r \cdot r' \in R$ and $\overline{r} \in R$ for $r, r' \in R$ such that $rr' \in H(r \cdot r')$ and $\overline{r} \in Hr^{-1}$. Recall that G and H are fixed groups, hence $r \cdot r'$ and $\overline{r}$ can be computed in constant time. In [21, Theorem 6] it is shown that the power word problem for G can be reduced in polynomial time (in fact, in NC^1) to the power word problem for H. The proof shows the following fact:

Fact 1. Given a power word w over the alphabet Σ we can compute in polynomial time a power word w' over the alphabet Θ and $r \in R$ such that $w = w'r$ in G.

Let now take finite list of power words $w_0, w_1, \ldots, w_n$ over the alphabet Σ and let $g_i \in G$ be the group element represented by w_i. We want to check whether $g_0 \in A := \langle g_1, \ldots, g_n \rangle$. In the following we will not distinguish between g_i and w_i.

First we use Fact 1 and rewrite in polynomial time each power word w_i as $w_i' r_i$ with $w_i' \in \Theta^*$ a power word and $r_i \in R$. Let w_i' represent $g_i' \in H$. By computing the closure of $\{r_1, \overline{r}_1, \ldots, r_n, \overline{r}_n\}$ with respect to the multiplication $\cdot$ on R we obtain the set of all representatives $r \in R$ such that $Hr \cap A \neq \emptyset$. Let us denote this closure with V. Clearly, $1 \in V$. If $r_0 \notin V$ then we have $w_0 = w_0' r_0 \notin A$.

Let us now assume that $r_0 \in V$. First assume that $r_0 = 1$, i.e., $w_0 = w_0' \in H$. Hence, $w_0 \in A$ if and only if $w_0 \in H \cap A$. We now compute a finite list of generators for $H \cap A$ written as power words over Θ. For this we follow [11]: we compute a power-compressed graph $\mathcal{G} = (V, E, \iota, \tau, \lambda, 1)$ (in the sense of Section 3.2) by taking V as the set of vertices. We draw an edge from $r \in V$ to $r' \in V$ labelled with the power word w_i (respectively, w_i^{-1}) iff $r \cdot r_i = r'$ (respectively, $r \cdot \overline{r}_i = r'$). Note that every edge has an inverse edge. The label of a path from $1 \in V$ back to $1 \in V$ in the graph $\mathcal{G}$ is a word over $\{w_1, w_1^{-1}, \ldots, w_n, w_n^{-1}\}$ and hence can be viewed as a power word over the alphabet Σ. As such, it represents an element of the group $H \cap A$.

Let T be a spanning tree of $\mathcal{G}$ and let $E \setminus T$ be the set of edges that do not belong to T. We then obtain a set of generators for $H \cap A$ by taking for every edge $e \in E \setminus T$ the circuit in $\mathcal{G}$ obtained by following the unique simple path in T from 1 to $\iota(e)$, followed by the edge e, followed by the unique simple path in T from $\tau(e)$ back to 1. Let $x_e \in \{w_1, w_1^{-1}, \ldots, w_n, w_n^{-1}\}^*$ be the label of this circuit. Every x_e represents an element of $H \cap A$ and the set of all these elements (for $e \in E \setminus T$) is a generating set of $H \cap A$; see [11] for details. Moreover, every x_e can be written as power word over the alphabet Σ of polynomial length. Using Fact 1 we can rewrite this power word in polynomial time into $x_e' r_e$ where x_e' is a power word over the alphabet Θ and $r_e \in R$. But since x_e represents an element of H, we must have $r_e = 1$. This concludes the case that $r_0 = 1$.

Finally, the case that $r_0 \in V$ but $r_0 \neq 1$ can be easily reduced to the case $r_0 = 1$: we use the same graph $\mathcal{G}$ defined above. Since $r_0 \in V$, there is a path from 1 to r_0. Let $x \in \{w_1, w_1^{-1}, \ldots, w_n, w_n^{-1}\}^*$ be the label of this path. It is a power word over Σ and by Fact 1 x can be rewritten into the form yr for a power word y over Θ and $r \in R$. Clearly, we must have $r = r_0$. In the group G we have $w_0 x^{-1} = w_0' r_0 r_0^{-1} y^{-1} = w_0' y^{-1}$, where the latter can be written as a power word over Θ. Since the word x represents an element of A we have $w_0 \in A$ if and only if $w_0 x^{-1} \in A$ if and only if $w_0' y^{-1} \in A$. This concludes the proof. ◀

From Theorem 12 and Lemma 13 we immediately obtain the following corollary:

▶ **Corollary 14.** *The power-compressed subgroup membership problem for a fixed f.g. virtually free group can be solved in polynomial time.*

The group $\mathsf{GL}(2,\mathbb{Z})$ consists of all (2×2)-matrices over the integers with determinant -1 or 1. It is a well-known example of a f.g. virtually free group [32].

▶ **Lemma 15.** *From a given matrix $A \in \mathsf{GL}(2,\mathbb{Z})$ with binary encoded entries one can compute in polynomial time a power word over a fixed finite generating set of $\mathsf{GL}(2,\mathbb{Z})$, which evaluates to the matrix A.*

Proof. For the group $\mathsf{SL}(2,\mathbb{Z})$ of all (2×2)-matrices over the integers with determinant 1 the result is shown in [12], see also [8, Proposition 15.4]. Now, $\mathsf{SL}(2,\mathbb{Z})$ is a normal subgroup of index two in $\mathsf{GL}(2,\mathbb{Z})$. Fix a matrix $B \in \mathsf{GL}(2,\mathbb{Z})$ with determinant -1. Given a matrix $A \in \mathsf{GL}(2,\mathbb{Z})$ with binary encoded entries and determinant -1 we first compute the matrix $AB^{-1} \in \mathsf{SL}(2,\mathbb{Z})$. Using [12] we can compute in polynomial time a power word w for AB^{-1}. Hence, wB (where B is taken as an additional generator) is a power word for A. ◀

▶ **Corollary 16.** *The subgroup membership problem for $\mathsf{GL}(2,\mathbb{Z})$ can be solved in polynomial time when matrix entries are given in binary encoding.*

Proof. Since $\mathsf{GL}(2,\mathbb{Z})$ is f.g. virtually free, the power-compressed subgroup membership problem for $\mathsf{GL}(2,\mathbb{Z})$ can be solved in polynomial time by Corollary 14. By Lemma 15 this shows the corollary. ◀

5 Future work

There is not much hope to generalize Corollary 16 to higher dimensions. For $\mathsf{SL}(4,\mathbb{Z})$ the subgroup membership problem is undecidable and decidability of the subgroup membership problem for $\mathsf{SL}(3,\mathbb{Z})$ is a long standing open problem [17].

A more feasible problem concerns the rational subset membership problem for free groups when transitions are labelled with power words. It is easy to see that this problem is NP-hard (reduction from subset sum) and we conjecture that there exists an NP algorithm. As a consequence this would show that the rational subset membership problem for $\mathsf{GL}(2,\mathbb{Z})$ is NP-complete when the transitions of the automaton are labelled with binary encoded matrices. The corresponding statement for $\mathsf{PSL}(2,\mathbb{Z})$ was shown in [4].

Another interesting problem is whether the subgroup membership problem for a free group can be solved in polynomial time, when all group elements are represented by straight-line programs (which can be more succinct than power words). One might try to show this using an adaptation of Stallings's folding, but controlling the size of the graph during the folding seems to be more difficult when the transition labels are represented by straight-line programs instead of power words.

References

1 Jürgen Avenhaus and Klaus Madlener. The Nielsen reduction and P-complete problems in free groups. *Theoretical Computer Science*, 32(1-2):61–76, 1984.

2 Jürgen Avenhaus and Dieter Wißmann. Using rewriting techniques to solve the generalized word problem in polycyclic groups. In *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation, ISSAC 1989*, pages 322–337. ACM Press, 1989.

3 Frédérique Bassino, Ilya Kapovich, Markus Lohrey, Alexei Miasnikov, Cyril Nicaud, Andrey Nikolaev, Igor Rivin, Vladimir Shpilrain, Alexander Ushakov, and Pascal Weil. Compression techniques in group theory. In *Complexity and Randomness in Group Theory*, chapter 4. De Gruyter, 2020.

4 Paul C. Bell, Mika Hirvensalo, and Igor Potapov. The identity problem for matrix semigroups in $\mathrm{SL}_2(\mathbb{Z})$ is NP-complete. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 187–206. SIAM, 2017.

5 Michèle Benois. Parties rationnelles du groupe libre. *Comptes rendus hebdomadaires des séances de l'Académie des sciences*, Séries A, 269:1188–1190, 1969.

6 Michèle Benois and Jacques Sakarovitch. On the complexity of some extended word problems defined by cancellation rules. *Information Processing Letters*, 23(6):281–287, 1986.

7 Michaël Cadilhac, Dmitry Chistikov, and Georg Zetzsche. Rational subsets of Baumslag-Solitar groups. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming, ICALP 2020*, volume 168 of *LIPIcs*, pages 116:1–116:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

8 Volker Diekert and Murray Elder. Solutions of twisted word equations, EDT0L languages, and context-free groups. *CoRR*, abs/1701.03297, 2017. arXiv:1701.03297.

9 Volker Diekert, Igor Potapov, and Pavel Semukhin. Decidability of membership problems for flat rational subsets of GL(2, Q) and singular matrices. In *Proceedings of the 45th International Symposium on Symbolic and Algebraic Computation, ISSAC 2020*, pages 122–129. ACM, 2020.

10 Stefan Friedl and Henry Wilton. The membership problem for 3-manifold groups is solvable. *Algebraic & Geometric Topology*, 16(4):1827–1850, 2016.

11 Zeph Grunschlag. *Algorithms in Geometric Group Theory*. PhD thesis, University of California at Berkley, 1999.

12 Yuri Gurevich and Paul E. Schupp. Membership problem for the modular group. *SIAM Journal on Computing*, 37(2):425–459, 2007.

13 Artur Jeż. The complexity of compressed membership problems for finite automata. *Theory of Computing Systems*, 55(4):685–718, 2014.

14 Ilya Kapovich and Alexei Myasnikov. Stallings foldings and subgroups of free groups. *Journal of Algebra*, 248(2):608–668, 2002.

15 Ilya Kapovich, Richard Weidmann, and Alexei Myasnikov. Foldings, graphs of groups and the membership problem. *International Journal of Algebra and Computation*, 15(1):95–128, 2005.

16 Olga G. Kharlampovich, Alexei G. Myasnikov, Vladimir N. Remeslennikov, and Denis E. Serbin. Subgroups of fully residually free groups: algorithmic problems. In *Group theory, statistics, and cryptography*, volume 360 of *Contemporary Mathematics*, pages 63–101. AMS, Providence, RI, 2004.

17 Evgeny I. Khukhro and Victor D. Mazurov. Unsolved problems in group theory. the Kourovka notebook. *CoRR*, arXiv:1401.0300v19, 2020. Problem 12.50. arXiv:1401.0300v19.

18 Sang-Ki Ko, Reino Niskanen, and Igor Potapov. On the identity problem for the special linear group and the Heisenberg group. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 132:1–132:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

19 Klaus-Jörn Lange and Pierre McKenzie. On the complexity of free monoid morphisms. In *Proceedings of the 9th International Symposium on Algorithms and Computation, ISAAC 1998*, number 1533 in Lecture Notes in Computer Science, pages 247–256. Springer, 1998.

20 Markus Lohrey. *The Compressed Word Problem for Groups*. SpringerBriefs in Mathematics. Springer, 2014.

21 Markus Lohrey and Armin Weiß. The power word problem. In *Proceedings of the 44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019*, volume 138 of *LIPIcs*, pages 43:1–43:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

22 M. Lothaire. *Combinatorics on Words*. Cambridge University Press, 1997.

23 Roger C. Lyndon and Paul E. Schupp. *Combinatorial Group Theory*. Springer, 1977.

24 Anatolij I. Mal'cev. On homomorphisms onto finite groups. *American Mathematical Society Translations, Series 2*, 119:67–79, 1983. Translation from Ivanov. Gos. Ped. Inst. Ucen. Zap. 18 (1958) 49–60.

25 Luda Markus-Epstein. Stallings foldings and subgroups of amalgams of finite groups. *International Journal of Algebra and Compution*, 17(8):1493–1535, 2007.

26 K. A. Mihaĭlova. The occurrence problem for direct products of groups. *Math. USSR Sbornik*, 70:241–251, 1966. English translation.

27 Igor Potapov and Pavel Semukhin. Decidability of the membership problem for 2 × 2 integer matrices. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 170–186. SIAM, 2017.

28 Eliyahu Rips. Subgroups of small cancellation groups. *Bulletin of the London Mathematical Society*, 14:45–47, 1982.

29 Nikolai S. Romanovskiĭ. Some algorithmic problems for solvable groups. *Algebra i Logika*, 13(1):26–34, 1974.

30 Nikolai S. Romanovskiĭ. The occurrence problem for extensions of abelian groups by nilpotent groups. *Sibirskii Matematicheskii Zhurnal*, 21:170–174, 1980.

31 Paul E. Schupp. Coxeter groups, 2-completion, perimeter reduction and subgroup separability. *Geometriae Dedicata*, 96:179–198, 2003.

32 Jean-Pierre Serre. *Trees*. Springer, 1980.

33 Charles C. Sims. Computation with permutation groups. In *Proceedings of SYMSAC 1971*, pages 23–28. Association for Computing Machinery, 1971.

34 John R. Stallings. Topology of finite graphs. *Inventiones Mathematicae*, 71(3):551–565, 1983.

35 Nicholas W. M. Touikan. A fast algorithm for Stallings' folding process. *International Journal of Algebra and Computation*, 16(6):1031–1045, 2006.

Lower Bounds for Graph-Walking Automata

Olga Martynova
Department of Mathematics and Computer Science, St. Petersburg State University, Russia

Alexander Okhotin
Department of Mathematics and Computer Science, St. Petersburg State University, Russia

Abstract

Graph-walking automata (GWA) traverse graphs by moving between the nodes following the edges, using a finite-state control to decide where to go next. It is known that every GWA can be transformed to a GWA that halts on every input, to a GWA returning to the initial node in order to accept, as well as to a reversible GWA. This paper establishes lower bounds on the state blow-up of these transformations: it is shown that making an n-state GWA traversing k-ary graphs return to the initial node requires at least $2(n-1)(k-3)$ states in the worst case; the same lower bound holds for the transformation to halting automata. Automata satisfying both properties at once must have at least $4(n-1)(k-3)$ states. A reversible automaton must have at least $4(n-1)(k-3)-1$ states. These bounds are asymptotically tight to the upper bounds proved using the methods from the literature.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Models of computation

Keywords and phrases Finite automata, graph-walking automata, halting, reversibility

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.52

Funding Research supported by the Russian Science Foundation, project 18-11-00100.

1 Introduction

Graph-walking automata (GWA) are finite automata that traverse labelled undirected graphs.

On the one hand, this is a model of a robot with limited memory navigating a discrete environment. There is an early result by Budach [2] that for every automaton there is a graph that it cannot fully explore; a short proof of this fact was later given by Fraigniaud et al. [5]. This work has influenced the current research on algorithms for graph traversal using various small-memory models, equipped with a limited number of pebbles, etc. [3, 4].

On the other hand, GWA naturally generalize such important models as tree-walking automata [1] (TWA) and two-way finite automata (2DFA). More generally, a GWA can represent various models of computation, if a graph is regarded as the space of memory configurations, and every edge accordingly represents an operation on the memory. This way, quite a few models in automata theory and in complexity theory, such as multi-head and multi-tape automata and space-bounded complexity classes, can be regarded as GWA, Then, some results on GWA apply to all these models.

Among such results, there are transformations of GWA to several important subclasses: to automata that halt on every input graph; to automata that return to the initial node in order to accept; to reversible automata. Such transformations have earlier been established for various automaton models, using a general method discovered by Sipser [15], who constructed a halting 2DFA that traverses the tree of computations of a given 2DFA leading to an accepting configuration, in search of an initial configuration. Later, Kondacs and Watrous [7] ensured the reversibility and optimized this construction for the number of states, motivated by the study of quantum automata. Sipser's idea has been adapted to proving that reversible space equals deterministic space [11], to making tree-walking automata halt [13], to complementing 2DFA [6], to making multi-head automata reversible [12], etc. Each transformation leads to a certain blow-up in the number of states, usually between linear and quadratic. No lower bounds on the transformation to halting have been established yet. For the transformation to reversible, a lower bound exists for the case of 2DFA [8], but it is quite far from the known upper bound.

For the general case of GWA, constructions of halting, returning and reversible automata were given by Kunc and Okhotin [9], who showed that an n-state GWA operating on graphs with k edge labels can be transformed to a returning GWA with $3nk$ states and to a reversible GWA with $6nk + 1$ states, which is always halting. Applied to special cases of GWA, such as TWA or multi-head automata, these generic constructions produce fewer states than the earlier specialized constructions.

The goal of this paper is to obtain lower bounds on the complexity of these transformations. To begin with, the constructions by Kunc and Okhotin [9] are revisited in Section 3, and it turns out that the elements they are built of can be recombined more efficiently, resulting in improved upper bounds based on the existing methods. This way, the transformation to a returning GWA is improved to use $2nk + n$ states, the transformation to halting can use $2nk + 1$ states, and constructing a reversible GWA (which is both returning and halting) requires at most $4nk+1$ states. The main result of the paper is that, with these improvements, each of these constructions is asymptotically optimal.

The lower bounds are proved according to the following plan. For each n and k, one should construct an n-state automaton operating on graphs with k direction labels, so that any returning, halting or reversible automaton recognizing the same language would require many states. The n-state automaton follows a particular path in an input graph in search for a special node. The node is always on that path, so that the automaton naturally encounters it if it exists. On the other hand, the graph is constructed, so that getting back is more challenging.

The graph is made of elements called *diodes*, which are easy to traverse in one direction and hard to traverse backwards. Diodes are defined in Section 4, where it is shown that a GWA needs to employ extra states to traverse a diode backwards.

The graph used in all lower bound arguments, constructed in Section 5, has a main path made of diodes leading to a special node, which makes returning more complicated, so that a returning automaton needs at least $2(n-1)(k-3)$ states. A variant of this graph containing a cycle made of diodes, presented in Section 6, poses a challenge to a halting automaton, which needs at least $2(n-1)(k-3)$ states. Section 7 combines the two arguments to establish a lower bound of $4(n-1)(k-3)$ on the number of states of an automaton that is returning and halting at the same time. This bound is adapted to reversible automata in Section 8: at least $4(n-1)(k-3) - 1$ states are required.

Overall, each transformation requires ca. $C \cdot nk$ states in the worst case, for a constant C. Each transformation has its own constant C, and these constants are determined precisely.

2 Graph-walking automata and their subclasses

This section provides a succinct introduction to graph-walking automata and to their halting and reversible subclasses. For more details, an interested reader is directed to the paper by Kunc and Okhotin [9], whereas some general explanations can be found in a recent survey [14].

The definition of graph-walking automata (GWA) is an intuitive extension of two-way finite automata (2DFA) and tree-walking automata (TWA). However, formalizing it requires extensive notation. First, there is a notion of a *signature*, which is a generalization of an alphabet to the case of graphs.

▶ **Definition 1** (Kunc and Okhotin [9]). A signature S *consists of*

- *A finite set D of directions, that is, labels attached to edge end-points;*
- *A bijection $-\colon D \to D$ providing an opposite direction, with $-(-d) = d$ for all $d \in D$;*
- *A finite set Σ of node labels;*
- *A non-empty subset $\Sigma_0 \subseteq \Sigma$ of possible labels of the initial node;*
- *A set of directions $D_a \subseteq D$ for every label $a \in \Sigma$. Every node labelled with a must be of degree $|D_a|$, with the incident edges corresponding to the elements of D_a.*

Graphs are defined over a signature, like strings over an alphabet.

▶ **Definition 2.** A graph over a signature $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a \in \Sigma})$ *is a quadruple* $(V, v_0, +, \lambda)$*, where*

- *V is a finite set of nodes;*
- *$v_0 \in V$ is the initial node;*
- *$+\colon V \times D \to V$ is a partial function, such that if $v + d$ is defined, then $(v + d) + (-d)$ is defined and equals v;*
- *a total mapping $\lambda\colon V \to \Sigma$, such that $v + d$ is defined if and only if $d \in D_{\lambda(v)}$, and $\lambda(v) \in \Sigma_0$ if and only if $v = v_0$.*

Once graphs are formally defined, a graph-walking automaton is defined similarly to a 2DFA.

▶ **Definition 3.** A (deterministic) graph-walking automaton (GWA) over a signature $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a \in \Sigma})$ *is a quadruple* $A = (Q, q_0, F, \delta)$*, where*

- *Q is a finite set of states;*
- *$q_0 \in Q$ is the initial state;*
- *$F \subseteq Q \times \Sigma$ is a set of acceptance conditions;*
- *$\delta\colon (Q \times \Sigma) \setminus F \to Q \times D$ is a partial transition function, with $\delta(q, a) \in Q \times D_a$ for all a and q where δ is defined.*

A computation of a GWA on a graph $(V, v_0, +, \lambda)$ is a uniquely defined sequence of configurations (q, v), with $q \in Q$ and $v \in V$. It begins with (q_0, v_0) and proceeds from (q, v) to $(q', v + d)$, where $\delta(q, \lambda(v)) = (q', d)$. The automaton accepts by reaching (q, v) with $(q, \lambda(v)) \in F$.

On each input graph, a GWA can accept, reject or loop. There is a natural subclass of GWA that never loop.

▶ **Definition 4.** *A graph-walking automaton is said to be halting, if its computation on every input graph is finite.*

Another property is getting back to the initial node before acceptance: if a GWA is regarded as a robot, it returns to its hangar, and for a generic model of computation, this property means cleaning up the memory.

▶ **Definition 5.** *A graph-walking automaton $A = (Q, q_0, F, \delta)$ over a signature $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a \in \Sigma})$ is called returning, if $F \subseteq Q \times \Sigma_0$, which means that it can accept only in the initial node.*

A returning automaton is free to reject in any node, and it may also loop, that is, it need not be halting.

The next, more sophisticated property is *reversibility*, meaning that, for every configuration, the configuration at the previous step can be uniquely reconstructed. This property is essential in quantum computing, whereas irreversibility in classical computers causes energy dissipation, which is known as *Landauer's principle* [10].

The definition of reversibility begins with the property that every state is reachable from only one direction.

▶ **Definition 6.** *A graph-walking automaton $A = (Q, q_0, F, \delta)$ over a signature $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a\in\Sigma})$ is called direction-determinate, if there is a function $d \colon Q \to D$, such that, for all $p \in Q$ and $a \in \Sigma$, if $\delta(p, a)$ is defined, then $\delta(p, a) = (q, d(q))$ for some $q \in Q$.*

▶ **Definition 7.** *A graph-walking automaton $A = (Q, q_0, F, \delta)$ over a signature $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a\in\Sigma})$ is called reversible, if*

- *A is direction-determinate;*
- *for all $a \in \Sigma$ and $q \in Q$, there is at most one state p, such that $\delta(p, a) = (q, d(q))$; in other words, knowing a state and a previous label, one can determine the previous state;*
- *The automaton is returning, and for each $a_0 \in \Sigma_0$ there exists at most one such state q, that $(q, a_0) \in F$.*

In theory, a reversible automaton may loop, but only through the initial configuration. In this case, it can be made halting by introducing an extra initial state.

Every GWA can be transformed to each of the above subclasses [9]. In the next section, the known transformations will be explained and slightly improved.

3 Upper bounds revisited

Before establishing the lower bounds on all transformations, the existing constructions of Kunc and Okhotin [9] will be somewhat improved by using fewer states. This is achieved by recombining the elements of the original construction, and, with these improvements, the constructions shall be proved asymptotically optimal.

All transformations are based on the following lemma.

▶ **Lemma 8** (Kunc and Okhotin [9, Lemma 4]). *For every direction-determinate GWA A, one can construct a reversible GWA B with twice as many states, so that for every accepting configuration (q, v) of A on a graph G, if B starts on G in $(q, v - d(q))$, then B reversibly traverses the tree of all computations of A that lead to the configuration (q, v). If B ever finds the initial configuration, it accepts, and otherwise it rejects in a copy of the accepting configuration of A.*

Note that the computation of A starting from the initial configuration can reach at most one accepting configuration (q, v), whereas for any other accepting configuration (q, v), the automaton B will not find the initial configuration and will reject as stated in the lemma.

To transform a given n-state GWA $\widehat{A}$ over a signature with k directions to a returning automaton, Kunc and Okhotin [9] first transform it to a direction-determinate automaton A with nk states; let B be the $2nk$-state automaton obtained from A by Lemma 8. Then they construct an automaton that first operates as A, and then, after reaching an accepting configuration, works as B to return to the initial node. This results in a returning direction-determinate automaton with $3nk$ states.

If the goal is just to return, and remembering the direction is not necessary, then $2nk + n$ states are actually enough.

▶ **Theorem 9.** *For every n-state GWA over a signature with k directions, there exists a returning automaton with $2nk + n$ states recognizing the same set of graphs.*

Indeed, the original automaton $\widehat{A}$ can be first simulated as it is, and once it reaches an accepting configuration, one can use the same automaton B as in the original construction to return to the initial node. There is a small complication in the transition from $\widehat{A}$ to B, because in the accepting configuration, the direction last used is unknown. This is handled by cycling through all possible previous configurations of A at this last step, and executing B from each of them. If the direction is guessed correctly, then B finds the initial configuration and accepts. Otherwise, if the direction is wrongly chosen, B returns back, and then, instead of rejecting, it is executed again starting from the next direction. One of these directions leads it back to the initial node.

Kunc and Okhotin [9] did not consider halting automata separately. Instead, they first transform an n-state GWA to a $3nk$-state returning direction-determinate automaton, then use Lemma 8 to obtain a $6nk$-state reversible automaton, and add an extra initial state to start it. The resulting $(6nk+1)$-state automaton is always halting.

If only the halting property is needed, then the number of states can be reduced.

▶ **Theorem 10.** *For every n-state direction-determinate automaton, there exists a $(2n+1)$-state halting and direction-determinate automaton that recognizes the same set of graphs.*

First, an n-state automaton $\widehat{A}$ is transformed to a direction-determinate nk-state automaton A, and Lemma 8 is used to construct a $2nk$-state automaton B. Then, the automaton B is *reversed* by the method of Kunc and Okhotin [9], resulting in an automaton B^R with $2nk+1$ states that carries out the computation of B backwards. The automaton B^R is a halting automaton that recognizes the same set of graphs as $\widehat{A}$: it starts in the initial configuration, and if B accepts from an accepting configuration of A, then B^R finds this configuration and accepts; otherwise, B^R halts and rejects.

The construction of a reversible automaton with $6nk+1$ states can be improved to $4nk+1$ by merging the automata B and B^R. The new automaton first works as B^R to find the accepting configuration of A. If it finds it, then it continues as B to return to the initial node. In addition, this automaton halts on every input.

▶ **Theorem 11.** *For every n-state direction-determinate automaton there exists a $(4n+1)$-state reversible and halting automaton recognizing the same set of graphs.*

With the upper bounds improved, it is time to establish asymptotically matching lower bounds.

4 Construction of a “diode”

Lower bounds on the size of GWA obtained in this paper rely on quite involved constructions of graphs that are easy to traverse from the initial node to the moment of acceptance, whereas traversing the same path backwards is hard. An essential element of this construction is a subgraph called a *diode*; graphs in the lower bound proofs are made of such elements.

A diode is designed to replace an $(a,-a)$-edge. An automaton can traverse it in the direction a without changing its state. However, traversing it in the direction $-a$ requires at least $2(|D|-3)$ states, where D is the set of directions in the diode's signature.

If an automaton never moves in the direction $-a$, then it can be transformed to an automaton with the same number of states, operating on graphs in which every $(a,-a)$-edge is replaced with a diode.

Lower bound proofs for automata with n states over a signature with k directions use a diode designed for these particular values of n and k. This diode is denoted by $\Delta_{n,k}$.

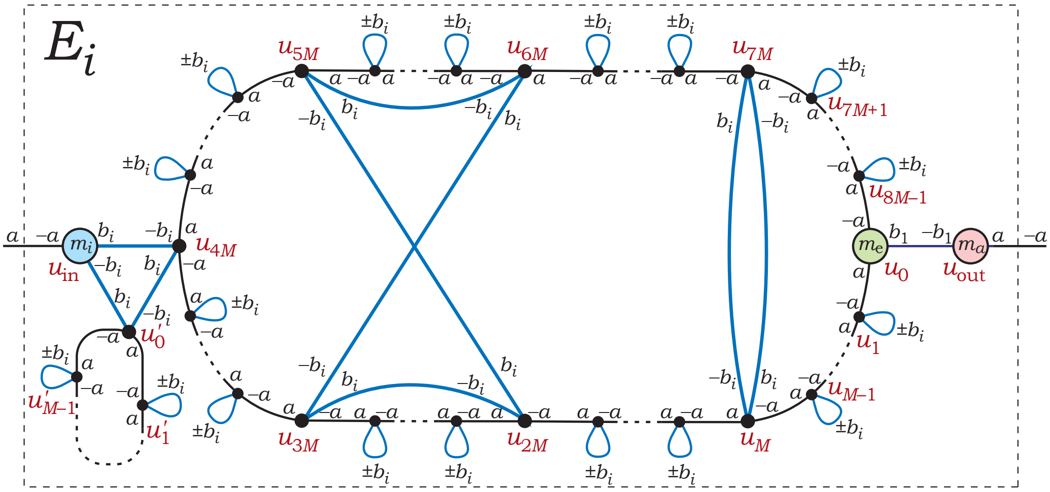


Figure 1 Element E_i. Filled circles are nodes labelled with m, each with $r-1$ loops in directions $\pm b_s$, with $s \neq i$.

For $n \geqslant 2$ and $k \geqslant 4$, let $M = (4nk)!$, and let $r = \lfloor \frac{k-2}{2} \rfloor$. A diode $\Delta_{n,k}$ is defined over a signature S_k that does not depend on n.

▶ **Definition 12.** *A signature $S_k = (D, -, \Sigma, \Sigma_0, (D_a)_{a \in \Sigma})$ consists of:*

- *the set of directions $D = \{a, -a\} \cup \{b_1, b_{-1}, \ldots, b_r, b_{-r}\}$;*
- *opposite directions $-(a) = (-a)$, $-b_i = b_{-i}$, for $1 \leqslant i \leqslant r$;*
- *the set of node labels $\Sigma = \{m_1, \ldots, m_r\} \cup \{m_{-1}, \ldots, m_{-r}\} \cup \{m, m_{\mathrm{e}}, m_a\}$, with no initial labels defined ($\Sigma_0 = \varnothing$) since the diode is inserted into graphs;*
- *sets of directions allowed at labels: $D_m = D$, $D_{m_i} = D_{m_{-i}} = \{-a, b_i, -b_i\}$, for $i = 1, \ldots, r$, $D_{m_{\mathrm{e}}} = \{b_1, a, -a\}$, $D_{m_a} = \{-b_1, a\}$.*

A diode is comprised of $2r$ elements E_i, E_{-i}, for $i \in \{1, \ldots, r\}$. Each element E_i and E_{-i} is a graph over the signature S_k, with two external edges, one with label a, the other with $-a$. By these edges, the elements are connected in a chain.

The form of an element E_i is illustrated in Figure 1. Its main part is a cycle of length $8M$ in directions $a, -a$; these are nodes $u_0, \ldots, u_{8M-1}$, where the arithmetic in the node numbers is modulo $8M$, e.g., $u_{-1} = u_{8M-1}$. The node numbers are incremented in direction a. Besides the main cycle, there are two extra nodes: the entry point u_{in} and the exit u_{out}, as well as a small circle of length M in directions $a, -a$ with the nodes $u'_0, \ldots, u'_{M-1}$. All nodes are labelled with m, except three: u_{in} with label m_i matching the index of the element, u_{out} with label m_a, and u_0 has label m_{e}.

An element E_i has specially defined edges in directions b_i and $-b_i$. Each node u_j with $j \not\equiv 0 \pmod{M}$ has a $(b_i, -b_i)$-loop. The nodes u_j with $j \in \{M, 2M, 3M, 5M, 6M, 7M\}$ are interconnected with edges, as shown in Figure 1; these edges serve as traps for an automaton traversing the element backwards. The node u_{4M} has a different kind of trap in the form of a cycle $u'_0, \ldots, u'_{M-1}$. For all $s \neq i$, each node labelled with m has a $(b_s, -b_s)$-loop.

The element E_{-i} is the same as E_i, with the directions b_i and $-b_i$ swapped.

The diode $\Delta_{n,k}$ is a chain of such elements, as illustrated in Figure 2. Each element can be traversed from the entrance to the exit without changing the state: at first, the automaton sees the label m_i, and accordingly moves in the direction b_i; then, on labels m, it proceeds in the direction a until it reaches u_0, labelled with m_{e}. Then the automaton leaves the element by following directions b_1 and a.

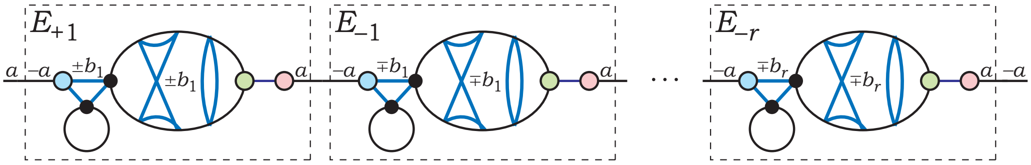


Figure 2 Diode $\Delta_{n,k}$: a chain of elements E_1, E_{-1}, E_2, E_{-2}, $\ldots$, E_r, E_{-r}.

The diode is hard to traverse backwards, because the node u_{4M} is not specifically labelled, and in order to locate it, the automaton needs to move in directions $\pm b_i$ from many nodes, and is accordingly prone to falling into traps.

The diode is used as a subgraph connecting two nodes of a graph as if an $(a, -a)$-edge. For a graph G over some signature $\widetilde{S}$, let G' be a graph obtained by replacing every $(a, -a)$-edge in G with the diode $\Delta_{n,k}$. Denote this graph operation by $h_{n,k} \colon G \mapsto G'$.

The following lemma states that if an automaton never traverses an $(a, -a)$-edge backwards, then its computations can be replicated on graphs with these edges substituted by diodes, with no extra states needed.

▶ **Lemma 13.** *Let $\widetilde{S}$ be any signature containing directions a, $-a$, which has no node labels from the signature S_k. Let $A = (Q, q_0, F, \delta)$ be a GWA over the signature $\widetilde{S}$, which never moves in the direction $-a$.*

Then, there exists a GWA $A' = (Q', q'_0, F', \delta')$ over a joint signature $\widetilde{S} \cup S_k$, with $|Q'| = |Q|$, so that A accepts a graph G if and only if A' accepts the graph $G' = h_{n,k}(G)$.

Lemma 13 shows that, under some conditions, a substitution of diodes can be implemented on GWA without increasing the number of states. The next lemma presents an *inverse substitution of diodes*: the set of pre-images under $h_{n,k}$ of graphs accepted by a GWA can be recognized by another GWA with the same number of states.

▶ **Lemma 14.** *Let $k \geqslant 4$ and $n \geqslant 2$, denote $h(G) = h_{n,k}(G)$ for brevity. Let $\widetilde{S}$ be a signature containing the directions $a, -a$ and no node labels from the diode's signature S_k. Let B be a GWA over the signature $\widetilde{S} \cup S_k$. Then there exists an automaton C over the signature $\widetilde{S}$, using the same set of states, with the following properties.*

- *For every graph G over $\widetilde{S}$, the automaton C accepts G if and only if B accepts $h(G)$.*
- *If C can enter a state q by a transition in direction $-a$, then B can enter the state q after traversing the diode backwards.*
- *If B is returning, then so is C.*
- *If B is halting, then C is halting as well.*

The automaton C is constructed by simulating B on small graphs, and using the outcomes of these computations to define the transition function and the set of acceptance conditions of C. Note that the signatures $\widetilde{S}$ and S_k may contain any further common directions besides $a, -a$: this does not cause any problems with the proof, because the node labels are disjoint, and thus B always knows whether it is inside or outside a diode.

▶ **Lemma 15.** *Let $A = (Q, q_0, F, \delta)$ be a GWA over a signature that includes the diode's signature S_k, with $|Q| \leqslant 4nk$. Assume that A, after traversing the diode $\Delta_{n,k}$ backwards, can leave the diode in any of h distinct states. Then A has at least $2h(k-3)$ states.*

Sketch of a proof. While moving through an element E_i backwards, the automaton sees labels m most of the time, and soon begins repeating a periodic sequence of states. Without loss of generality, assume that this periodic sequence contains more transitions in the direction

a than in $-a$. Then the automaton reaches the node u_M, and at this point it may teleport between u_M and u_{-M} several times. Let $w \in \{b_i, -b_i\}^*$ be the sequence of these teleportation moves, and let x be the corresponding sequence of states. Depending on the sequence w, the automaton may eventually exit the cycle to the node u_{in}, or fall into one of the traps and get back to u_0. It is proved that for the automaton to reach u_{in}, the string w must be non-empty and of even length; furthermore, if $|w| = 2$, then $w = (-b_i)b_i$.

Now consider the h backward traversals of the diode ending in some states $p_1, \ldots, p_h$. When the traversal ending in p_j proceeds through the element E_i, the strings $w_{i,j} \in \{b_i, -b_i\}^*$ and $x_{i,j}$ are defined as above. Then, as the last step of the argument, it is proved that whenever $|w_{i,j}| = 2$, the states in $x_{i,j}$ cannot occur in any other string $x_{i',j'}$. For $w_{i,j}$ of length 4 or more, the states in $x_{i,j}$ can repeat in other strings $x_{i',j'}$, but only once. It follows that there are at least $2h(k-3)$ distinct states in these strings. ◀

5 Lower bound on the size of returning automata

By the construction of Kunc and Okhotin [9], as improved in Section 3, an n-state GWA over a signature with k directions can be transformed to a returning GWA with $2nk + n$ states. A closely matching lower bound will now be proved by constructing an automaton with n states over a signature with k directions, such that every returning automaton that recognizes the same set of graphs must have at least $2(n-1)(k-3)$ states.

The first step is a construction of a simple automaton over a signature $\widetilde{S}$ with four directions $a, -a, b, -b$ and two graphs over this signature, so that the automaton accepts one of them and rejects the other. The automaton will have n states, it will never move in the direction $-a$, and every returning automaton recognizing the same set of graphs can enter $n-1$ distinct states after transitions in the direction $-a$. Then, Lemma 15 shall assert that every returning automaton recognizing the same graphs with diodes substituted must have the claimed number of states.

▶ **Definition 16.** *The signature $\widetilde{S} = (D, -, \Sigma, \Sigma_0, (D_a)_{a\in\Sigma})$ uses the set of directions $D = \{a, -a, b, -b\}$, with $-(a) = (-a)$, $-(b) = (-b)$. The set of node labels is $\Sigma = \{c_0, c, c_l, c_r, c_{acc}\}$, with initial labels $\Sigma_0 = \{c_0\}$. The allowed directions are $D_c = D$, $D_{c_0} = D_{c_l} = \{a\}$, and $D_{c_r} = D_{c_{acc}} = \{-a\}$.*

For $n \geqslant 2$ and $k \geqslant 4$, let $M = (4nk)!$ be as in the definition of the diode $\Delta_{n,k}$. Let $G^{accept}_{n,k}$ and $G^{reject}_{n,k}$ be two graphs over the signature $\widetilde{S}$, defined as follows. The graph $G^{accept}_{n,k}$ is illustrated in Figure 3; the other graph $G^{reject}_{n,k}$ is almost identical, but the node that determines acceptance is differently labelled.

Both graphs consist of two horizontal chains of nodes, connected by bridges at two places. Nodes are pairs (x, y), where $y \in \{-1, 1\}$ is the number of the chain, and x is the horizontal coordinate, with $-(n-1) \leqslant x \leqslant M + 8nk$ for the lower chain ($y = -1$) and $-8nk \leqslant x \leqslant M + 8nk$ for the upper chain ($y = 1$).

All nodes except the ends of chains have labels c. The node $(-(n-1), -1)$ is the initial node, with label c_0. The other left end $(-8nk, 1)$ is labelled with c_l. The node $(M + 8nk, -1)$ has label c_r. The node $(M + 8nk, 1)$ is labelled with c_{acc} in $G^{accept}_{n,k}$ and with c_r in $G^{reject}_{n,k}$; this is the only difference between the two graphs.

The horizontal chains are formed of $(a, -a)$-edges, with a incrementing x and $-a$ decrementing it. Edges with labels $(b, -b)$ are loops at all nodes except for $(0, 1)$, $(0, -1)$, $(M, 1)$ and $(M, -1)$. The latter four nodes form two pairs connected with bridges in directions $(b, -b)$.

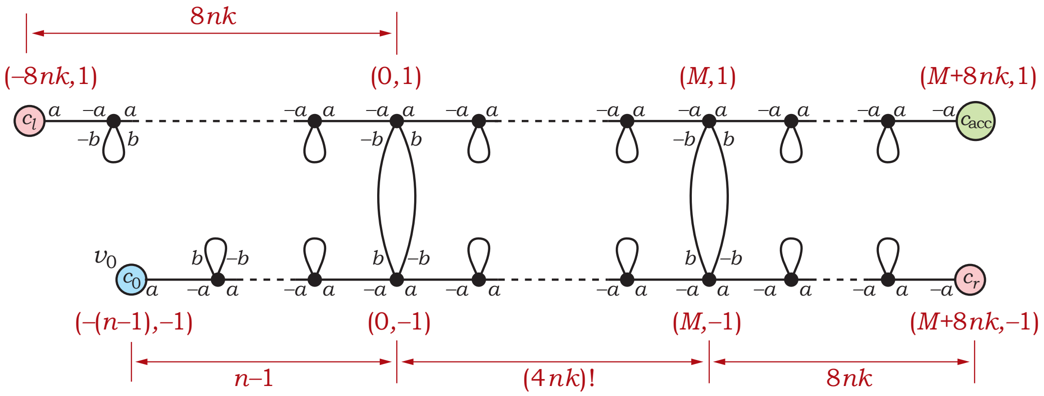


Figure 3 The graph $G^{accept}_{n,k}$.

An n-state automaton A, that accepts the graph $G^{accept}_{n,k}$, does not accept any graphs without labels c_{acc} and never moves in the direction $-a$, is defined as follows. In the beginning, it moves in the direction a in the same state q_0, then makes $n-2$ further steps in the direction a, incrementing the number of state. Next, it crosses the bridge in the direction b and enters the last, n-th state, in which it moves in the direction a until it sees the label c_{acc}.

▶ **Lemma 17.** *Every returning automaton that accepts the same set of graphs as A, and has at most $4nk$ states, may enter at least $n-1$ distinct states after transitions in the direction $-a$.*

Sketch of a proof. On the graph $G^{accept}_{n,k}$, a returning automaton, after seeing the node $(M+8nk, 1)$, must find its way back to the initial node. At some point, it leaves one of the ends of the upper chain, $(-8nk, 1)$ or $(M+8nk, 1)$, and then arrives at one of the ends of the lower chain, $(-(n-1), -1) = v_0$ or $(M+8nk, -1)$. On the way, it passes through nodes labelled with c, and eventually starts behaving periodically. It is claimed that its periodic sequence of directions contains at least $n-1$ moves in the direction $-a$.

First assume that the automaton leaves the node $(M+8nk, 1)$. Let s be the difference between the number of $-a$ and a in the periodic sequence. Then $s > 0$, and each period the automaton shifts by s edges to the left. As the automaton passes through the nodes $(M, \pm 1)$, it may move to the lower chain without noticing that; but if it does so, then later at the nodes $(0, \pm 1)$, the sequence of transitions will move it back to the upper chain. If it stays on the same chain at $(M, \pm 1)$, then it will stay on it at the second time as well. If there are too few directions $-a$, then, on the way through $(0, \pm 1)$, it would not reach the node $(-(n-1), -1) = v_0$, and will end up at the node $(-8nk, 1)$. This contradicts that assumption that the automaton has left both ends of the upper chain for good.

If the automaton leaves the node $(-8nk, 1)$, the argument is similar. The number s is defined, and now it must be negative. As the automaton passes through $(0, \pm 1)$, if it has too few directions $-a$, then it cannot reach v_0, and it eventually proceeds to $(M+8nk, 1)$, which is again a contradiction. ◀

It remains to combine this lemma with the properties of the diode to obtain the desired theorem.

▶ **Theorem 18.** *For every $k \geqslant 4$, there exists a signature with k directions, such that, for every $n \geqslant 2$, there is an n-state graph-walking automaton, such that every returning automaton recognizing the same set of graphs must have at least $2(n-1)(k-3)$ states.*

Proof. The proof uses the automaton A defined above. By Lemma 13, the n-state automaton A over the signature $\widetilde{S}$, is transformed to n-state automaton A' over the signature $\widetilde{S} \cup S_k$. The directions $\pm a$ are the same for $\widetilde{S}$ and S_k, and $\pm b$ in $\widetilde{S}$ are merged with $\pm b_1$ in S_k, so there are k directions in total.

For every graph G, the automaton A' accepts a graph $h_{n,k}(G)$ with $(a, -a)$-edges replaced by diodes, if and only if A accepts G. The automaton A' is the desired example: it is claimed that every returning automaton B recognizing the same set of graphs as A' has at least $2(n-1)(k-3)$ states.

Let B be any returning automaton with at most $4nk$ states recognizing these graphs. By Lemma 14, there is an automaton C over the signature $\widetilde{S}$ and with the same number of states, which accepts a graph G if and only if B accepts $h(G)$. This is equivalent to A accepting G, and so C and A accept the same set of graphs. Since B is returning, by Lemma 14, C is returning too. Then, Lemma 17 asserts that the automaton C may enter $n-1$ distinct states after moving in the direction $-a$.

Then, according to Lemma 14, the automaton B enters at least $n-1$ distinct states after traversing the diode backwards. Therefore, by Lemma 15, this automaton should have at least $2(k-3)(n-1)$ states. ◀

6 Lower bound on the size of halting automata

Every n-state GWA with k directions can be transformed to a halting GWA with $2nk+1$ states, as shown in Section 3. In this section, the following lower bound for this construction is established.

▶ **Theorem 19.** *For every $k \geqslant 4$, there is a signature with k directions, such that for every $n \geqslant 2$ there is an n-state GWA, such that every halting automaton accepting the same set of graphs has at least $2(n-1)(k-3)$ states.*

The argument shares some ideas with the earlier proof for the case of returning automata: the signature $\widetilde{S}$, the graphs $G_{n,k}^{accept}$ and $G_{n,k}^{reject}$, and the automaton A are the same as constructed in Section 5. The proof of Theorem 19 uses the following lemma, stated similarly to Lemma 17 for returning automata.

▶ **Lemma 20.** *Every halting automaton A', accepting the same set of graphs as A and using at most $4nk$ states, must be able to enter at least $n-1$ distinct states after transitions in the direction $-a$.*

Sketch of a proof. Consider the computation of A' on the graph G, defined by merging the nodes $(M + 8nk, 1)$ and $(-8nk, 1)$ in $G_{n,k}^{reject}$, into a single node v_{joint}, with label c. The automaton A' must visit this node, because it is the only difference between G and $G_{n,k}^{accept}$. By the time the automaton reaches v_{joint}, it already behaves periodically, and in order to stop, it needs to visit any label other than c, that is, return to one of the end-points of the lower chain. As in Lemma 17, the automaton can reach either end-point only if the periodic sequence contains at least $n-1$ moves in the direction $-a$. ◀

The proof of Theorem 19 is completed via Lemmata 13, 14 and 15, in the same way as for returning automata, only using Lemma 20 instead of Lemma 17.

7 Lower bound on the size of returning and halting automata

An n-state GWA over a signature with k directions can be transformed to an automaton that *both* halts on every input *and* accepts only in the initial node: a reversible automaton with $4nk + 1$ states, described in Section 3, will do.

This section establishes a close lower bound on this transformation. The witness n-state automaton is the same as in Sections 5–6, for which Theorem 18 asserts that a returning automaton needs at least $2(n-1)(k-3)$ states, whereas Theorem 19 proves that a halting automaton needs at least $2(n-1)(k-3)$ states. The goal is to prove that these two sets of states must be disjoint, leading to the following lower bound.

▶ **Theorem 21.** *For every $k \geqslant 4$, there exists a signature with k directions, such that for every $n \geqslant 2$, there is an n-state graph-walking automaton, such that every returning and halting automaton recognizing the same set of graphs must have at least $4(n-1)(k-3)$ states.*

As before, the automaton is obtained from A by Lemma 13. For the argument to proceed, the following property needs to be established.

▶ **Lemma 22** (cf. Lemma 17)**.** *Every returning and halting automaton that recognizes the same set of graphs as A, and has at most $4nk$ states, enters at least $2(n-1)$ distinct states after transitions in the direction $-a$.*

Sketch of a proof. Consider any such returning and halting automaton. Since it is returning, as shown in Lemma 17, on the graph $G^{accept}_{n,k}$, the automaton uses a periodic sequence of states to return from $(M + 8nk, 1)$ to v_0. Since it is at the same time halting, Lemma 20 asserts that on the graph G it uses another periodic sequence of states to escape the cycle after visiting v_{joint}. Each of these two sequences makes transitions in the direction $-a$ in at least $n-1$ distinct states. It remains to prove that these sequences are disjoint.

Suppose the sequences have a common element, then they coincide up to a cyclic shift. Then it is possible to modify G so that the computation coming to v_{joint} later continued as the computation on $G^{accept}_{n,k}$, and led to acceptance. ◀

The proof of the theorem is inferred from Lemmata 13, 14, 15 and 22, as in the earlier arguments.

8 Lower bound on the size of reversible automata

For the transformation of a GWA with n states and k directions to a reversible automaton, $4nk + 1$ states are sufficient. A close lower bound shall now be established.

▶ **Theorem 23.** *For every $k \geqslant 4$, there exists a signature with k directions, such that for every $n \geqslant 2$, there is an n-state GWA, such that every reversible GWA recognizing the same set of graphs has at least $4(n-1)(k-3) - 1$ states.*

Proof. By Theorem 21, there is such an n-state automaton A' that every returning and halting automaton recognizing the same set of graphs has at least $4(n-1)(k-3)$ states. Suppose that there is a reversible automaton with fewer than $4(n-1)(k-3) - 1$ states that accepts the same graphs as A'. Let m be the number of states in it. Then, by the construction of reversing a reversible automaton given by Kunc and Okhotin [9], there is a returning and halting automaton with $m + 1$ states, that is, with fewer than $4(n-1)(k-3)$ states. This contradicts Theorem 21. ◀

9 Conclusion

The new bounds on the complexity of transforming graph-walking automata to automata with returning, halting and reversibility properties are fairly tight. However, for their important special cases, such as two-way finite automata (2DFA) and tree-walking automata (TWA), the gaps between lower bounds and upper bounds are still substantial.

For an n-state 2DFA, the upper bound for making it halting is $4n + \text{const}$ states [6]. No lower bound is known, and any lower bound would be interesting to obtain. A 2DFA can be made reversible using $4n + 3$ states [9], with a lower bound of $2n - 2$ states [8]; it would be interesting to improve these bounds.

The same question applies to tree-walking automata: they can be made halting [13], and, for k-ary trees, it is sufficient to use $4kn + 2k + 1$ states to obtain a reversible automaton [9]. No lower bounds are known, and this subject is suggested for further research.

Furthermore, it would be interesting to try to apply the lower bound methods for GWA to limited memory algorithms for navigation in graphs.

References

1 Mikolaj Bojanczyk and Thomas Colcombet. Tree-walking automata cannot be determinized. *Theor. Comput. Sci.*, 350(2-3):164–173, 2006. doi:10.1016/j.tcs.2005.10.031.

2 Lothar Budach. Automata and labyrinths. *Mathematische Nachrichten*, 86(1):195–282, 1978. doi:10.1002/mana.19780860120.

3 Yann Disser, Jan Hackfeld, and Max Klimm. Undirected graph exploration with $\Theta(\log\log n)$ pebbles. 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 25–39. SIAM, 2016. doi:10.1137/1.9781611974331.ch3.

4 Amr Elmasry, Torben Hagerup, and Frank Kammer. Space-efficient basic graph algorithms. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 288–301. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.288.

5 Pierre Fraigniaud, David Ilcinkas, Guy Peer, Andrzej Pelc, and David Peleg. Graph exploration by a finite automaton. *Theor. Comput. Sci.*, 345(2-3):331–344, 2005. doi:10.1016/j.tcs.2005.07.014.

6 Viliam Geffert, Carlo Mereghetti, and Giovanni Pighizzini. Complementing two-way finite automata. *Inf. Comput.*, 205(8):1173–1187, 2007. doi:10.1016/j.ic.2007.01.008.

7 Attila Kondacs and John Watrous. On the power of quantum finite state automata. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 66–75. IEEE Computer Society, 1997. doi:10.1109/SFCS.1997.646094.

8 Michal Kunc and Alexander Okhotin. Reversible two-way finite automata over a unary alphabet. Technical Report 1024, Turku Centre for Computer Science, 2011.

9 Michal Kunc and Alexander Okhotin. Reversibility of computations in graph-walking automata. *Inf. Comput.*, 275:104631, 2020. doi:10.1016/j.ic.2020.104631.

10 Rolf Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.*, 5(3):183–191, 1961. doi:10.1147/rd.53.0183.

11 Klaus-Jörn Lange, Pierre McKenzie, and Alain Tapp. Reversible space equals deterministic space. *J. Comput. Syst. Sci.*, 60(2):354–367, 2000. doi:10.1006/jcss.1999.1672.

12 Kenichi Morita. A deterministic two-way multi-head finite automaton can be converted into a reversible one with the same number of heads. In Robert Glück and Tetsuo Yokoyama, editors, *Reversible Computation, 4th International Workshop, RC 2012, Copenhagen, Denmark, July 2-3, 2012. Revised Papers*, volume 7581 of *Lecture Notes in Computer Science*, pages 29–43. Springer, 2012. doi:10.1007/978-3-642-36315-3_3.

13 Anca Muscholl, Mathias Samuelides, and Luc Segoufin. Complementing deterministic tree-walking automata. *Inf. Process. Lett.*, 99(1):33–39, 2006. doi:10.1016/j.ipl.2005.09.017.

14 Alexander Okhotin. Graph-walking automata: From whence they come, and whither they are bound. In Michal Hospodár and Galina Jirásková, editors, *Implementation and Application of Automata - 24th International Conference, CIAA 2019, Košice, Slovakia, July 22-25, 2019, Proceedings*, volume 11601 of *Lecture Notes in Computer Science*, pages 10–29. Springer, 2019. doi:10.1007/978-3-030-23679-3_2.

15 Michael Sipser. Lower bounds on the size of sweeping automata. *J. Comput. Syst. Sci.*, 21(2):195–202, 1980. doi:10.1016/0022-0000(80)90034-3.

An Improved Approximation Algorithm for the Maximum Weight Independent Set Problem in d-Claw Free Graphs

Meike Neuwohner
Research Institute for Discrete Mathematics, Universität Bonn, Germany

Abstract

In this paper, we consider the task of computing an independent set of maximum weight in a given d-claw free graph $G = (V, E)$ equipped with a positive weight function $w : V \to \mathbb{R}^+$. Thereby, $d \geq 2$ is considered a constant. The previously best known approximation algorithm for this problem is the local improvement algorithm *SquareImp* proposed by Berman [2]. It achieves a performance ratio of $\frac{d}{2} + \epsilon$ in time $\mathcal{O}(|V(G)|^{d+1} \cdot (|V(G)| + |E(G)|) \cdot (d-1)^2 \cdot \left(\frac{d}{2\epsilon} + 1\right)^2)$ for any $\epsilon > 0$, which has remained unimproved for the last twenty years. By considering a broader class of local improvements, we obtain an approximation ratio of $\frac{d}{2} - \frac{1}{63{,}700{,}992} + \epsilon$ for any $\epsilon > 0$ at the cost of an additional factor of $\mathcal{O}(|V(G)|^{(d-1)^2})$ in the running time. In particular, our result implies a polynomial time $\frac{d}{2}$-approximation algorithm. Furthermore, the well-known reduction from the weighted k-Set Packing Problem to the Maximum Weight Independent Set Problem in $k+1$-claw free graphs provides a $\frac{k+1}{2} - \frac{1}{63{,}700{,}992} + \epsilon$-approximation algorithm for the weighted k-Set Packing Problem for any $\epsilon > 0$. This improves on the previously best known approximation guarantee of $\frac{k+1}{2} + \epsilon$ originating from the result of Berman [2].

2012 ACM Subject Classification Theory of computation → Packing and covering problems

Keywords and phrases d-Claw free Graphs, independent Set, local Improvement, k-Set Packing, weighted

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.53

Related Version
Full Version: https://www.or.uni-bonn.de/home/neuwohner/set_packing_full_version.pdf

1 Introduction

For $d \geq 1$, a *d-claw* C [2] is defined to be a star consisting of one center node and a set T_C of d additional vertices connected to it, which are called the *talons* of the claw (see Figure 1). Moreover, similar to [2], we define a 0-claw to be a graph consisting only of a single vertex v, which is regarded as the unique element of T_C in this case. An undirected graph $G = (V, E)$ is said to be *d-claw free* if none of its induced subgraphs forms a d-claw. For example, 1-claw free graphs do not possess any edges, while 2-claw free graphs are disjoint unions of cliques. For natural numbers $k \geq 3$, the Maximum Weight Independent Set Problem (MWIS) in $k+1$-claw free graphs is often studied as a generalization of the weighted k-Set Packing Problem, which is defined as follows: Given a family $\mathcal{S}$ of sets each of size at most k together with a positive weight function $w : \mathcal{S} \to \mathbb{R}^+$, the task is to find a disjoint sub-collection of $\mathcal{S}$ of maximum weight. By considering the *conflict graph* $G_{\mathcal{S}}$ associated with an instance of the weighted k-Set Packing Problem, the vertices of which are given by the sets in $\mathcal{S}$ and the edges of which represent non-empty set intersections, one obtains a weight preserving one-to-one correspondence between feasible solutions to the k-Set Packing Problem and independent sets in $G_{\mathcal{S}}$, which can be shown to be $k+1$-claw free.

While as far as the weighted version of the k-Set Packing Problem is concerned, the algorithm devised by Berman in 2000 [2] to deal with the MWIS in $k+1$-claw free graphs remains unchallenged so far, considerable progress has been made for the cardinality variant during the last decade. The first improvement over the approximation guarantee of k achieved

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 53; pp. 53:1–53:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

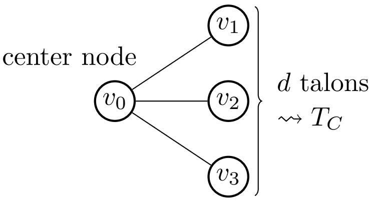
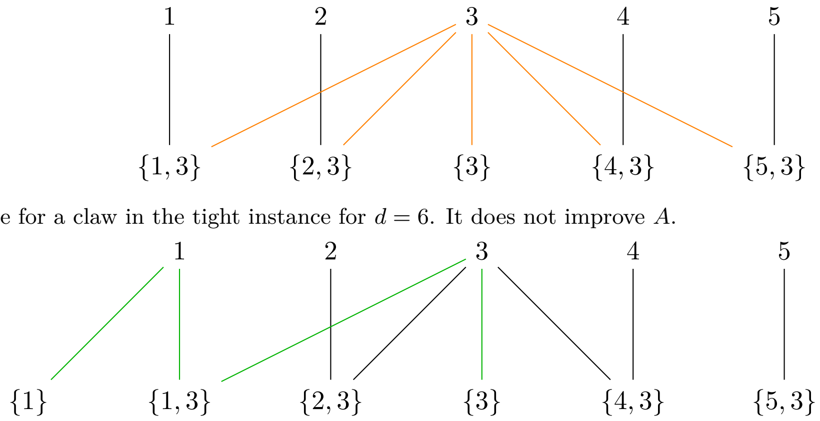


Figure 1 a d-claw C for $d = 3$.

by a simple greedy approach was obtained by Hurkens and Schrijver in 1989 [9], who showed that for any $\epsilon > 0$, there exists a constant p_ϵ for which a local improvement algorithm that first computes a maximal collection of disjoint sets and then repeatedly applies local improvements of constant size at most p_ϵ, until no more exist, yields an approximation guarantee of $\frac{k}{2} + \epsilon$. In this context, a disjoint collection X of sets contained in the complement of the current solution A is considered a *local improvement of size* $|X|$ if the sets in X intersect at most $|X| - 1$ sets from A, which are then replaced by the sets in X, increasing the cardinality of the found solution. Hurkens and Schrijver also proved that a performance guarantee of $\frac{k}{2}$ is best possible for a local search algorithm only considering improvements of constant size, while Hazan, Safra and Schwartz [8] established in 2006 that no $o(\frac{k}{\log k})$-approximation algorithm is possible in general unless $P = NP$. At the cost of a quasi-polynomial runtime, Halldórsson [7] could prove an approximation factor of $\frac{k+2}{3}$ by applying local improvements of size logarithmic in the total number of sets. Cygan, Grandoni and Mastrolilli [5] managed to get down to an approximation factor of $\frac{k+1}{3} + \epsilon$, still with a quasi-polynomial runtime. The first polynomial time algorithm improving on the result by Hurkens and Schrijver was obtained by Sviridenko and Ward [13] in 2013. By combining means of color coding with the algorithm presented in [7], they achieved an approximation ratio of $\frac{k+2}{3}$. This result was further improved to $\frac{k+1}{3} + \epsilon$ for any fixed $\epsilon > 0$ by Cygan [4], obtaining a polynomial runtime doubly exponential in $\frac{1}{\epsilon}$. The best approximation algorithm for the unweighted k-Set Packing Problem in terms of performance ratio and running time is due to Fürer and Yu from 2014 [6], who achieved the same approximation guarantee as Cygan, but a runtime that is only singly exponential in $\frac{1}{\epsilon}$.

Concerning the unweighted version of the MWIS in d-claw free graphs, as remarked in [13], both the result of Hurkens and Schrijver as well as the quasi-polynomial time algorithms by Halldórsson and Cygan, Grandoni and Mastrolilli translate to this more general context, yielding approximation guarantees of $\frac{d-1}{2} + \epsilon$, $\frac{d+1}{3}$ and $\frac{d}{3} + \epsilon$, respectively. However, it is not clear how to extend the color coding approach relying on coloring the underlying universe to the setting of d-claw free graphs [13].

When it comes to the weighted variant of the problem, even less is known. For $d \leq 3$, it is solvable in polynomial time (see [10] and [12] for the unweighted, [11] for the weighted variant), while for $d \geq 4$, again no $o(\frac{d}{\log d})$-approximation algorithm is possible unless $P = NP$ [8]. Moreover, in contrast to the unit weight case, considering local improvements the size of which is bounded by a constant can only slightly improve on the performance ratio of $d - 1$ obtained by the greedy algorithm since Arkin and Hassin have shown that such an approach yields an approximation ratio no better than $d - 2$ in general [1]. Thereby, analogously to the unweighted case, given an independent set A, an independent set X is called a *local improvement of* A if it is disjoint from A and the total weight of the neighbors of X in A is strictly smaller than the weight of X. Despite the negative result in [1], Chandra and Halldórsson [3] have found that if one does not perform the local improvements in an arbitrary order, but in each step augments the current solution A by an improvement X that maximizes the ratio between the total weight of the vertices added to and removed



(a) Example for a claw in the tight instance for $d = 6$. It does not improve A.

(b) $\{\{1\},\{1,3\},\{3\}\}$ constitutes a local improvement of constant size.

Figure 2 (Part of) the tight instance provided in [2].

from A (if exists), the resulting algorithm, which the authors call *BestImp*, approximates the optimum solution within a factor of $\frac{2d}{3}$. By scaling and truncating the weight function to ensure a polynomial number of iterations, they obtain a $\frac{2d}{3}+\epsilon$-approximation algorithm for the MWIS in d-claw free graphs for any $\epsilon > 0$.

As already mentioned, the currently best known approximation guarantee for the MWIS in d-claw free graphs is due to Berman [2], who suggested the algorithm *SquareImp*, which iteratively applies local improvements of the squared weight function that arise as sets of talons of claws in G, until no more exist. An induced subgraph C of G is thereby called a *claw in G* if there is some $t \geq 0$ such that C constitutes a t-claw. The algorithm SquareImp achieves an approximation ratio of $\frac{d}{2}$, leading to a polynomial time $\frac{d}{2}+\epsilon$-approximation algorithm for any $\epsilon > 0$. Its running time can be bounded by $\mathcal{O}(|V(G)|^{d+1}\cdot(|V(G)|+|E(G)|)\cdot(d-1)^2\cdot\left(\frac{d}{2\epsilon}+1\right)^2)$.

Berman also provides an example for $w \equiv 1$ showing that his analysis is tight. It consists of a bipartite graph $G = (V, E)$ the vertex set of which splits into a maximal independent set $A = \{1, \ldots, d-1\}$ such that no claw improves $|A|$, and an optimum solution $B = \binom{A}{1} \cup \binom{A}{2}$, whereby the set of edges is given by $E = \{\{a, b\} : a \in A, b \in B, a \in b\}$. As the example uses unit weights, he also concludes that applying the same type of local improvement algorithm for a different power of the weight function does not provide further improvements.
However, as also implied by the result in [9], while no small improvements *forming the set of talons of a claw* in the input graph exist in the tight example given by Berman, once this additional condition is dropped, improvements of small constant size can be found quite easily (see Figure 2). This in turn indicates that considering a less restricted class of local improvements may result in a better approximation guarantee.

In this paper, we revisit the analysis of the algorithm SquareImp proposed by Berman and show that whenever it is close to being tight, the instance actually bears a similar structure to the tight example given in [2] in a certain sense. By further observing that if this is the case, there must exist a local improvement (with respect to the squared weight function) of size at most $d-1+(d-1)^2$, we can conclude that a local improvement algorithm looking for improvements of w^2 obeying the aforementioned size bound achieves an improved approximation ratio at the cost of an additional $\mathcal{O}(|V(G)|^{(d-1)^2})$ factor in the running time.

The rest of this paper is organized as follows: In Section 2, we review the algorithm SquareImp by Berman and give a short overview of the analysis pointing out the results we reuse in the analysis of our algorithm. The latter is presented in Section 3, which also

Algorithm 1 SquareImp [2].

```
Input: an undirected $d$-claw free graph $G = (V, E)$ and a positive weight function
       $w : V \to \mathbb{R}^+$
Output: an independent set $A \subseteq V$
$A \leftarrow \emptyset$
while there exists a claw $C$ in $G$ that improves $w^2(A)$ do
    $A \leftarrow A \backslash N(T_C, A) \cup T_C$
return $A$
```

provides a detailed analysis proving an approximation guarantee of $\frac{d}{2} - \frac{1}{63{,}700{,}992} + \epsilon$ for any $\epsilon > 0$. Finally, Section 4 concludes the paper with some remarks on possibilities to improve on the given result, but also difficulties that one might face along the way.

2 Preliminaries

In this section, we shortly recap the definitions and main results from [2] that we will employ in the analysis of our local improvement algorithm. We first introduce some basic notation that is needed for its formal description.

▶ **Definition 1** (neighborhood [2]). *Given an undirected graph $G = (V, E)$ and subsets $U, W \subseteq V$ of vertices, we define the* neighborhood $N(U, W)$ *of U in W as*

$$N(U, W) := \{w \in W : \exists u \in U : \{u, w\} \in E \vee u = w\}.$$

In order to simplify notation, for $u \in V$ and $W \subseteq V$, we write $N(u, W)$ instead of $N(\{u\}, W)$.

▶ **Notation 2.** *Given a weight function $w : V \to \mathbb{R}$ and some $U \subseteq V$, we write $w^2(U) := \sum_{u \in U} w^2(u)$. Observe that in general, $w^2(U) \neq (w(U))^2$.*

▶ **Definition 3** ([2]). *Given an undirected graph $G = (V, E)$, a positive weight function $w : V \to \mathbb{R}^+$ and an independent set $A \subseteq V$, we say that a vertex set $B \subseteq V$* improves $w^2(A)$ *if B is independent in G and $w^2(A \backslash N(B, A) \cup B) > w^2(A)$ holds. For a claw C in G, we say that C improves $w^2(A)$ if its set of talons T_C does.*

Observe that an independent set B improves A if and only if we have $w^2(B) > w^2(N(B, A))$ (see Proposition 12). Further note that we do not require B to be disjoint from A.
Using the notation introduced above, Berman's algorithm SquareImp [2] can now be formulated as in Algorithm 1. Observe that by positivity of the weight function, every $v \notin A$ such that $A \cup \{v\}$ is independent constitutes the talon of a 0-claw improving $w^2(A)$, so the algorithm returns a maximal independent set.

The main idea of the analysis of SquareImp presented in [2] is to charge the vertices in A for preventing adjacent vertices in an optimum solution A^* from being included into A. The latter is done by spreading the weight of the vertices in A^* among their neighbors in the maximal independent set A in such a way that no vertex in A receives more than $\frac{d}{2}$ times its own weight. The suggested distribution of weights thereby proceeds in two steps:

First, each vertex $u \in A^*$ invokes costs of $\frac{w(v)}{2}$ at each $v \in N(u, A)$, leaving a remaining weight of $w(u) - \frac{w(N(u,A))}{2}$ to be distributed. (Note that this term can be negative.)
In a second step, each vertex in u therefore sends an amount of $w(u) - \frac{w(N(u,A))}{2}$ to a heaviest neighbor it possesses in A, which is captured by the following definition of *charges*:

▶ **Definition 4** (charges [2]). *Let $G = (V, E)$ be an undirected graph and let $w : V \to \mathbb{R}^+$ be a positive weight function. Further assume that an independent set $A^* \subseteq V$ and a maximal independent set $A \subseteq V$ are given. We define a map* $\text{charge} : A^* \times A \to \mathbb{R}$ *as follows:*

For each $u \in A^*$, pick a vertex $v \in N(u, A)$ of maximum weight and call it $n(u)$. Observe that this is possible, because A is a maximal independent set in G, implying that $N(u, A) \neq \emptyset$ since either $u \in A$ itself or u possesses a neighbor in A.
Next, for $u \in A^*$ and $v \in A$, define

$$\text{charge}(u, v) := \begin{cases} w(u) - \frac{1}{2}w(N(u, A)) & , \text{ if } v = n(u) \\ 0 & , \text{ otherwise} \end{cases}.$$

The definition of charges directly implies the subsequent statement:

▶ **Corollary 5** ([2]). *In the situation of Definition 4, we have*

$$\begin{aligned} w(A^*) &= \sum_{u \in A^*} \frac{w(N(u, A))}{2} + \sum_{u \in A^*} \text{charge}(u, n(u)) \\ &\leq \sum_{u \in A^*} \frac{w(N(u, A))}{2} + \sum_{u \in A^* : \text{charge}(u, n(u)) > 0} \text{charge}(u, n(u)). \end{aligned}$$

The analysis proposed by Berman now proceeds by bounding the total weight sent to the vertices in A during the two steps of the cost distribution separately. Lemma 6 thereby bounds the weight received in the first step, while Lemma 7 and Lemma 8 take care of the total charges invoked. (Note that although we have slightly changed the formulation of the subsequent results to suit our purposes, they either appear in [2] in an equivalent form or are directly implied by the proofs presented there.)

▶ **Lemma 6** ([2]). *In the situation of Definition 4, if the graph G is d-claw free for some $d \geq 2$, then*

$$\sum_{u \in A^*} \frac{w(N(u, A))}{2} \leq \frac{d-1}{2} \cdot w(A).$$

▶ **Lemma 7** ([2]). *In the situation of Definition 4, for $u \in A^*$ and $v \in A$ with* $\text{charge}(u, v) > 0$, *we have*

$$w^2(u) - w^2(N(u, A) \backslash \{v\}) \geq 2 \cdot \text{charge}(u, v) \cdot w(v).$$

▶ **Lemma 8** ([2]). *Let $G = (V, E)$ be d-claw free, $d \geq 2$, and $w : V \to \mathbb{R}^+$. Let further A^* be an independent set in G of maximum weight and let A be independent in G with the property that no claw improves $w^2(A)$. Then for each $v \in A$, we have*

$$\sum_{u \in A^* : \text{charge}(u, v) > 0} \text{charge}(u, v) \leq \frac{w(v)}{2}.$$

Algorithm 2 Local improvement algorithm.

```
Input: an undirected $d$-claw free graph $G = (V, E)$ and a positive weight function
       $w : V \to \mathbb{R}^+$
Output: an independent set $A \subseteq V$
$A \leftarrow \emptyset$
while there exists a local improvement $X$ of $w^2(A)$ do
    $A \leftarrow A \backslash N(X, A) \cup X$
return A
```

The proofs are omitted due to page limit.

By combining Corollary 5 with the previous lemmata, one obtains Theorem 9, stating an approximation guarantee of $\frac{d}{2}$:

▶ **Theorem 9** ([2]). *Let $G = (V, E)$ be d-claw free, $d \geq 2$, and $w : V \to \mathbb{R}^+$. Let further A^* be an independent set in G of maximum weight and let A be independent in G with the property that no claw improves $w^2(A)$. Then*

$$w(A^*) \leq \sum_{u \in A^*} \frac{w(N(u, A))}{2} + \sum_{u \in A^* : \text{charge}(u, n(u)) > 0} \text{charge}(u, n(u)) \leq \frac{d}{2} \cdot w(A).$$

After having recapitulated the results from [2] that we will reemploy in our analysis, we are now prepared to study our algorithm that takes into account a broader class of local improvements.

3 Improving the Approximation Factor

3.1 The Local Improvement Algorithm

▶ **Definition 10** (Local improvement). *Given a d-claw free graph $G = (V, E)$, a strictly positive weight function $w : V \to \mathbb{R}^+$ and an independent set $A \subseteq V$, we call an independent set $X \subseteq V$ a* local improvement *of $w^2(A)$ if $|X| \leq (d-1)^2 + (d-1)$ and $w^2(A \backslash N(X, A) \cup X) > w^2(A)$.*

▶ **Proposition 11.** *Let G, w and A be as in Definition 10. If X is a local improvement of $w^2(A)$, then $A \backslash N(X, A) \cup X$ is independent in G.*

▶ **Proposition 12.** *Let G, w and A be as in Definition 10. Then an independent set X of size at most $(d-1)^2 + (d-1)$ constitutes a local improvement of A if and only if we have $w^2(N(X, A)) < w^2(X)$.*

The remainder of Section 3 is now dedicated to the analysis of Algorithm 2 for the Maximum Weight Independent Set Problem in d-claw free graphs for $d \geq 2$. Thereby, the main result of this paper is given by the following theorem:

▶ **Theorem 13.** *If A^* is an optimum solution to the MWIS in a d-claw free graph G for some $d \geq 2$ and A denotes the solution returned by Algorithm 2, then we have*

$$w(A^*) \leq \left(\frac{d}{2} - \frac{1}{63,700,992}\right) \cdot w(A).$$

First, note that Algorithm 2 is correct in the sense that it returns an independent set. This follows immediately from the fact that we maintain the property that A is independent throughout the algorithm, because $\emptyset$ is independent and Proposition 11 tells us that none of our update steps can harm this invariant.

Next, observe that Algorithm 2 is guaranteed to terminate since no set A can be attained twice, given that $w^2(A)$ strictly increases in each iteration of the while-loop, and there are only finitely many possibilities. Furthermore, each iteration runs in polynomial (considering d a constant) time $\mathcal{O}(|V|^{(d-1)^2+d-1} \cdot (|V|+|E|))$, because there are only $\mathcal{O}(|V|^{(d-1)^2+d-1})$ many possible choices for X and we can check in linear time $\mathcal{O}(|V|+|E|)$ whether a given one constitutes a local improvement.

In order to achieve a polynomial number of iterations, we scale and truncate the weight function as explained in [3] and [2]. Given a constant $N > 1$, we first compute a greedy solution A' and rescale the weight function w such that $w(A') = N \cdot |V|$ holds. Then, we delete vertices v of truncated weight $\lfloor w(v) \rfloor = 0$ and run Algorithm 2 with the integral weight function $\lfloor w \rfloor$. In doing so, we know that $\lfloor w \rfloor^2(A)$ equals zero initially and must increase by at least one in each iteration. On the other hand, at each point, we have

$$\lfloor w \rfloor^2(A) \leq w^2(A) \leq (w(A))^2 \leq (d-1)^2 w^2(A') = (d-1)^2 \cdot N^2 \cdot |V|^2,$$

which bounds the total number of iterations by the latter term. Finally, if $r > 1$ specifies the approximation guarantee achieved by Algorithm 2, A denotes the solution it returns and A^* is an independent set of maximum weight with respect to the original respectively the scaled, but untruncated weight function w, we know that

$$r \cdot w(A) \geq r \cdot \lfloor w \rfloor(A) \geq \lfloor w \rfloor(A^*) \geq w(A^*) - |A^*| \geq w(A^*) - |V| \geq \frac{N-1}{N} \cdot w(A^*),$$

so the approximation ratio increases by a factor of at most $\frac{N}{N-1}$.

3.2 Analysis of the Performance Ratio

We now move on to the analysis of the approximation guarantee. Denote some optimum solution by A^* and denote the solution found by Algorithm 2 by A. Observe that by positivity of the weight function, A must be a maximal independent set, as adding a vertex would certainly yield a local improvement of $w^2(A)$.

We first show that for $d = 2$, our algorithm is actually optimal, so that we can restrict ourselves to the case $d \geq 3$ for the main analysis. As already remarked earlier, 2-claw free graphs are disjoint unions of cliques, so an optimum solution can be found by picking a vertex of maximum weight from each clique. But this is precisely what Algorithm 2 does:

First, we know that it returns a maximal independent set A, which must hence contain exactly one vertex per clique.

Second, if for some of the cliques, A contains a vertex v the weight of which is not maximum among all vertices in the clique, and $u \notin A$ belongs to the same clique and has maximum weight, then $\{u\}$ constitutes a local improvement of w^2 since we have $N(u, A) = \{v\}$ and $w^2(v) < w^2(u)$. This contradicts the termination criterion of our algorithm. Hence, Algorithm 2 is optimum for $d = 2$, and we can assume $d \geq 3$ in the following.

For the analysis, we define two constants, δ and ϵ, which we choose to be $\delta := \frac{1}{6}$ and $\epsilon := \frac{1}{5308416}$. These choices satisfy a bunch of inequalities that are used throughout the analysis and can be found in Appendix A.

Our goal is to show that Algorithm 2 produces a $\frac{d-\epsilon\delta}{2}$-approximation. We use some notation as well as most of the analysis of the algorithm SquareImp by Berman. In particular, we employ the same definition of neighborhoods and charges. Observe that this is well-defined as we have seen that the solution A returned by our algorithm must constitute a maximal independent set in the given graph.

For the remainder of this section, fix $d \geq 3$ and some instance of the MWIS in d-claw free graphs given by a (d-claw free) graph $G = (V, E)$ and a positive weight function $w : V \to \mathbb{R}^+$ and pick an optimum solution A^* for the given instance. Let further A denote the solution returned by Algorithm 2. We have to prove that $w(A^*) \leq \frac{d-\epsilon\delta}{2} \cdot w(A)$. In doing so, the first step of the analysis is to ensure that for almost all vertices $u \in A^*$, the total weight of their neighborhood in A is only by a small constant factor larger than the weight of u. For this purpose, we consider the set P of "payback vertices" $u \in A^*$ for which the total weight of $N(u, A)$ is at least three times as large as $w(u)$. For these vertices, the first step of the weight distribution employed in the analysis by Berman significantly overestimates their weight in that they invoke total costs that are by a factor of 1.5 larger. As a consequence, we can reduce the total weight sent to A by at least $\frac{w(P)}{2}$, making each of the vertices in P "pay back" the unnecessary costs they have created, and still obtain an upper bound on $w(A^*)$. But this means that the analysis of Berman, applied to our algorithm, can actually only be close to tight if the total weight of P is almost zero, which is the essential statement of the following lemma. The proof is omitted due to page limit.

▶ **Lemma 14.** *Let $P := \{u \in A^* : w(N(u, A)) \geq 3 \cdot w(u)\}$. Then for all $\gamma > 0$, if $w(P) \geq \gamma \cdot w(A)$, we have $w(A^*) \leq \frac{d-\gamma}{2} \cdot w(A)$.*

In order to prove an approximation factor of $\frac{d-\epsilon\delta}{2}$, we can hence restrict ourselves to the case where $w(P) < \epsilon\delta \cdot w(A)$ in the following.

Our next goal is to examine the structure of the neighborhoods $N(v, A^*)$ of vertices $v \in A$ that receive a total amount of charges that is close to $\frac{w(v)}{2}$, that is, for which the analysis of SquareImp, applied to Algorithm 2, is almost tight. More precisely, we only consider those neighbors of v sending positive charges to v and try to relate them to the vertices of the form $\{i\}$ respectively $\{i, j\}$ for $i \neq j$ (which actually invoke zero charges in the given instance) from the tight example. For this purpose, the following definitions are required:

▶ **Definition 15** (T_v). *For $v \in A$, we define $T_v := \{u \in A^* : \text{charge}(u, v) > 0\}$.*

▶ **Definition 16** (single vertex). *For $v \in A$, we call a vertex $u \in T_v$* single *if*

(i) $\frac{w(u)}{w(v)} \in [1 - \sqrt{\epsilon}, 1 + \sqrt{\epsilon}]$ *and*

(ii) $w(N(u, A)) \leq (1 + \sqrt{\epsilon}) \cdot w(v)$.

▶ **Definition 17** (double vertex). *For $v \in A$, we call a vertex $u \in T_v$* double *if $|N(u, A)| \geq 2$ and for $v_1 = v$ and v_2 a vertex of maximum weight in $N(u, A) \backslash \{v_1\}$, the following properties hold:*

(i) $\frac{w(u)}{w(v_1)} \in [1 - \sqrt{\epsilon}, 1 + \sqrt{\epsilon}]$

(ii) $\frac{w(v_2)}{w(v_1)} \in [1 - \sqrt{\epsilon}, 1]$ *and*

(iii) $(2 - \sqrt{\epsilon}) \cdot w(v_1) \leq w(N(u, A)) < 2 \cdot w(u)$.

Note that for v_1 and v_2 as in the previous definition, we have $w(v_2) \leq w(v_1)$ since we know that $v_1 = v = n(u)$ is an element of $N(u, A)$ of maximum weight by definition of T_v and charges. Further observe that no vertex can be both single and double since this would imply $(2 - \sqrt{\epsilon}) \cdot w(v) \leq w(N(u, A)) \leq (1 + \sqrt{\epsilon}) \cdot w(v)$ and therefore $2 - \sqrt{\epsilon} \leq 1 + \sqrt{\epsilon}$, as $w(v) > 0$, leading to $\epsilon \geq \frac{1}{4}$ contradicting (5).

The single vertices can be thought of as the vertices of the form $\{i\}$ from the tight example, while the double vertices are in correspondence with those vertices given by sets of size 2, although in the given example, these actually would not be considered double themselves since they send zero charges.

▶ **Lemma 18.** *For $v \in A$, we either have $\sum_{u\in T_v} \text{charge}(u,v) \leq \frac{1-\epsilon}{2} \cdot w(v)$, or for each $u \in T_v$, we have exactly one of the following:*

(i) *u is single or*

(ii) *u is double,*

and moreover, there exists at most one $u \in T_v$ that is single.

We would like to provide some motivation why we are actually interested in a statement of this type. To this end, first note that if the total weight of those vertices $v \in A$ satisfying $\sum_{u\in T_v} \text{charge}(u,v) \leq \frac{1-\epsilon}{2} \cdot w(v)$ constitutes some constant fraction of $w(A)$, we get an improved approximation factor since we gain an $\frac{\epsilon}{2}$-fraction of the weight of each such vertex when bounding the weight of A^*. On the other hand, if there are only few such vertices (in terms of weight), the vertices $v \in A$ for which the analysis of SquareImp is almost tight when it comes to charges, and for which all vertices in the set T_v can hence be classified as being either single or double, possess a large total weight. The set comprising these vertices v can be further split into the collection of those vertices that feature a neighbor that is single, and the set of those who do not. In order to gain some intuitive understanding of why Algorithm 2 achieves a better approximation guarantee than SquareImp, we have to see how both types of vertices can be helpful for our analysis.

For this purpose, let us first consider those vertices $v \in A$ all neighbors (in T_v) of which are double. Observe that for a double vertex $u_0 \in A^*$, its neighborhood $N(u_0, A)$ consists of two vertices $v_1 = n(u_0)$ and v_2 of roughly the same weight as u_0, plus maybe some additional vertices the total weight of which is by a factor in the order of $\sqrt{\epsilon}$ smaller. For simplicity, imagine that v_1 and v_2 have exactly the same weight and that there are no further neighbors of u_0 in A. In this situation, it is completely arbitrary whether v_1 or v_2 is chosen as $n(u_0)$. In particular, we can bound both of the terms $w^2(u_0) - w^2(N(u_0, A)\backslash\{v_1\})$ and $w^2(u_0) - w^2(N(u_0, A)\backslash\{v_2\})$ by $2 \cdot \text{charge}(u_0, n(u_0)) \cdot w(v_1) = 2 \cdot \text{charge}(u_0, n(u_0)) \cdot w(v_2)$ from below. Moreover, the proof of Lemma 8 tells us that for each $v \in A$, we actually get the stronger statement

$$\sum_{u\in N(v,A^*)} \max\{0, w^2(u) - w^2(N(u, A)\backslash\{v\})\} \leq w^2(v).$$

When summing over all $v \in A$, while every vertex $u \in A^*$ adds at least $2 \cdot \text{charge}(u, n(u))$ by Lemma 7, our "ideal" double vertex u_0 actually contributes twice as much since it adds an amount of at least $2 \cdot \text{charge}(u, n(u)) \cdot w(v_{1/2})$ for both v_1 and v_2.

Although for general double vertices, the situation is more complicated, one can still show that $w^2(u) - w^2(N(u, A)\backslash\{v_1\})$ amounts to almost $3 \cdot \text{charge}(u, v_1) \cdot w(v_1)$, or u adds approximately $\text{charge}(u, v_1) \cdot w(v_2)$ when it comes to v_2. As a consequence, for those vertices $v \in A$ receiving a total amount of charges of at least $\frac{1-\epsilon}{2} \cdot w(v)$ and all neighbors of which are double, the total charges sent to v can be counted almost three instead of only two times, resulting in an improved approximation factor provided the total weight of these vertices constitutes a constant fraction of $w(A)$.

We are therefore left with discussing the role of those $v \in A$ that possess at least one single neighbor. By Lemma 18, we further know that those v have exactly one single neighbor, which we denote by $t(v)$ in the following. Recall that by definition of single vertices, this neighbor bears roughly the same weight as v, and v makes up almost all of $N(t(v), A)$ in terms of weight. Imagine removing each such vertex v with a single neighbor from A and its neighbor $t(v) \in T_v$ from A^*. Then the sets of vertices removed from A and A^*, respectively, have roughly the same weight. It further constitutes a large fraction of $w(A)$, provided that $w(P)$, as well as the total weight of vertices for which the analysis of SquareImp is not close to being tight and the total weight of vertices with only double neighbors are small. (Remember that we obtain a better approximation guarantee if this is not the case.) But now, given that the ratio between the weights of the sets of vertices we have removed from A and A^*, respectively, is close to 1, we must get an improved approximation guarantee unless the ratio between the weights of the sets of vertices A'^* and A' remaining from A^* and A is way larger than $\frac{d}{2}$. But then, we know that we can find a local improvement X of $w^2(A')$ in the resulting instance, which can be extended to a local improvement in the original one by adding vertices that were removed from A^* to make up for the additional weight of neighbors of X that were removed from A. The existence of this local improvement contradicts the termination criterion of Algorithm 2.

We have therefore outlined the key ideas of the analysis of Algorithm 2 and in particular convinced ourselves of the benefit of the lemma. Its proof can be found in the appendix. After having seen that all neighbors of vertices v for which the analysis of SquareImp, applied to our algorithm, is almost tight, are either double or single, we continue by establishing the “usefulness“ of double vertices. As already outlined before, we show that the charges invoked by these can be counted almost three instead of only two times, which is captured by the next lemma.

▶ **Lemma 19.** *Let $u \in T_v$ be double, let $v = v_1$ and let v_2 be a vertex of maximum weight in $N(u, A)\backslash\{v_1\}$. Then at least one of the following inequalities holds:*

(i) $w^2(u) - w^2(N(u, A)\backslash\{v_1\}) \geq \frac{149}{50} \cdot \text{charge}(u, v_1) \cdot w(v_1)$ *or*

(ii) $w^2(u) - w^2(N(u, A)\backslash\{v_2\}) \geq \frac{49}{50} \cdot \text{charge}(u, v_1) \cdot w(v_2)$*.*

When motivating Lemma 18, we proposed to add charges invoked by vertices in A^* to a certain extent for vertices in A. This rather vague idea is clarified by the next definition as well as the two propositions and the lemma it is followed by.

While Proposition 21 bounds the total amount the neighborhood of each $v \in A$ can contribute to v in a locally optimal solution, Proposition 22 and Lemma 23 give lower bounds on the fraction of the invoked charges non-double and double vertices contribute in total.

▶ **Definition 20** (contribution)**.** *Define a contribution map*
$\text{contr} : A^* \times A \to \mathbb{R}_{\geq 0}$ *by setting*

$$\text{contr}(u, v) := \begin{cases} \max\left\{0, \frac{w^2(u) - w^2(N(u,A)\backslash\{v\})}{w(v)}\right\} & , \textit{ if } v \in N(u, A) \\ 0 & , \textit{ else} \end{cases}.$$

▶ **Proposition 21.** *For each $v \in A$, we have $\sum_{u \in A^*} \text{contr}(u, v) \leq w(v)$.*

This is a straightforward consequence of the fact that no local improvement of $w^2(A)$ exists.

▶ **Proposition 22.** *For each $u \in A^*$, we have*

$$\sum_{v \in A} \text{contr}(u, v) \geq \text{contr}(u, n(u)) \geq 2 \cdot \text{charge}(u, n(u)).$$

The statement follows by nonnegativity of the contribution and Lemma 7. Combining Lemma 7 and Lemma 19 yields the following result:

▶ **Lemma 23.** *For each double vertex u, we have* $\sum_{v\in A}\operatorname{contr}(u,v) \geq \frac{149}{50}\cdot\operatorname{charge}(u,n(u))$.

▶ **Definition 24** (C and D)**.** *Let C denote the set of all $v \in A$ for which*

(i) $\sum_{u\in T_v}\operatorname{charge}(u,v) > \frac{1-\epsilon}{2}\cdot w(v)$ *and*

(ii) *all vertices in T_v are double.*

Let further $D := \bigcup_{v\in C} T_v$.

Note that all vertices in D are double by definition. The following proposition tells us that the total charges invoked by vertices in D constitute a considerable fraction of the weight of C. It is a direct consequence of the definitions of C and D.

▶ **Proposition 25.** $\sum_{u\in D}\operatorname{charge}(u,n(u)) \geq \frac{1-\epsilon}{2}\cdot w(C)$.

As we have seen that double vertices contribute a factor of at least $\frac{149}{50}$ times the charges they send, we can finally conclude that we obtain an improved approximation factor unless the weight of C is extremely small compared to $w(A)$, which is the statement of the next lemma. It follows by combining Corollary 5, Lemma 6, Proposition 21, Proposition 22, Lemma 23 and Proposition 25.

▶ **Lemma 26.** *If* $w(C) \geq \frac{25}{12}\cdot\epsilon\delta\cdot w(A)$*, then* $w(A^*) \leq \frac{d-\epsilon\delta}{2}\cdot w(A)$.

By the previous lemma, we know that we can assume $w(C) < \frac{25}{12}\cdot\epsilon\delta\cdot w(A)$ in the following. As outlined before, we continue by proving that we get the desired approximation guarantee if the set of vertices for which the analysis of SquareImp is not almost tight constitutes at least a δ fraction of the weight of A. Let therefore

$$\bar{B} := \left\{v \in A : \sum_{u\in T_v}\operatorname{charge}(u,v) > \frac{1-\epsilon}{2}\cdot w(v)\right\}$$

denote the set of vertices for which the analysis of SquareImp is close to being tight. The proof of the following lemma is omitted due to page limit.

▶ **Lemma 27.** *If* $w(\bar{B}) \leq (1-\delta)\cdot w(A)$*, then* $\frac{d-\epsilon\delta}{2}\cdot w(A) \geq w(A^*)$.

If we have $w(\bar{B}) \leq (1-\delta)\cdot w(A)$, we achieve the claimed approximation factor of $\frac{d-\epsilon\delta}{2}$, so assume $w(\bar{B}) > (1-\delta)\cdot w(A)$ in the following. Let further $B := \bar{B}\backslash C$. Then we have $w(B) = w(\bar{B}) - w(C) > (1-\delta-\frac{25}{12}\cdot\epsilon\delta)\cdot w(A)$. By Lemma 18, each vertex $v \in B$ has a unique neighbor in T_v which is single. Call this neighbor $t(v)$ and let $B^* := \{t(v), v \in B\}$. We proceed by stating two lemmata that will later help us to transform local improvements in the instance arising by deleting the vertices in B, B^* and P into local improvements in the original one. Lemma 28 thereby tells us that for each $v \in B$, the total weight of the neighbors of $t(v)$ in A other than v is extremely small, while Lemma 29 establishes a relation between the squared weights of v and $t(v)$. The proofs are omitted due to page limit.

▶ **Lemma 28.** *For $v \in B$, we have* $w(N(t(v),A)\backslash\{v\}) \leq \sqrt{\epsilon}\cdot w(v)$.

▶ **Lemma 29.** *For $v \in B$, we have* $w(v)^2 \leq w(t(v))^2 + (4\sqrt{\epsilon}+4\epsilon)\cdot w^2(v)$.

Consider the sets $A' := A\backslash B$ and $A'^* := A^*\backslash(B^*\cup P)$ that arise from deleting all vertices in B and $B^*\cup P$. As outlined before, we would like to apply the analysis of SquareImp to bound the weight of A'^* in terms of the weight of A'. However, in order to employ the definition of charges, we have to make sure that A' constitutes a maximal independent set in $G[A'\cup A'^*]$. Showing this property is the purpose of the following lemma. As its proof is similar to the one of Lemma 32, we omit it due to page limit.

▶ **Lemma 30.** *If there exists a vertex $u \in A'^*$ such that $N(u, A') = \emptyset$, then there exist a local improvement of $w^2(A)$ in the original instance.*

Due to the termination criterion of our algorithm, we know that there is no local improvement in the original instance, so the previous lemma tells us that every vertex in A'^* must possess a neighbor in A' (considering vertices as adjacent to themselves), showing that A' is a maximal independent set in $G[A' \cup A'^*]$. We can hence apply the same strategy as in the analysis of SquareImp to bound the weight of A'^* by the weight of A', letting each vertex send charges to its heaviest neighbor in A', which must exist by the previous arguments. More precisely, we apply the definition of charges, Definition 4, to the sub-instance induced by $A' \cup A'^*$, in which A'^* is independent and A' is a maximal independent set. Call the resulting charge map charge$'$ and recall that it is constructed as follows:

For each $u \in A'^*$, we pick a heaviest neighbor $v \in N(u, A')$ and call it $n'(u)$. Then, for $u \in A'^*$ and $v \in A'$, we define

$$\text{charge}'(u, v) := \begin{cases} w(u) - \frac{w(N(u,A'))}{2} & \text{if } v = n'(u) \\ 0 & \text{otherwise} \end{cases}.$$

For $v \in A'$, let $T'_v := \{u \in A'^* : \text{charge}'(u, v) > 0\}$ denote the set of vertices in A'^* that now send positive charges to v.

We show that we obtain the desired approximation ratio, provided

$$\sum_{u \in T'_v} \text{charge}'(u, v) \leq \frac{d+2}{4} \cdot w(v)$$

holds for all $v \in A'$, and that we can find a local improvement of $w^2(A)$ in the original instance if this is not the case, contradicting the fact that our algorithm did terminate.

▶ **Lemma 31.** *If $\sum_{u \in T'_v} \text{charge}'(u, v) \leq \frac{d+2}{4} \cdot w(v)$ holds for all $v \in A'$, then we have $w(A^*) \leq \frac{d-\epsilon\delta}{2} \cdot w(A)$.*

We are left with proving the following lemma:

▶ **Lemma 32.** *For all $v \in A'$, we have*

$$\sum_{u \in T'_v} \text{charge}'(u, v) \leq \frac{d+2}{4} \cdot w(v).$$

This concludes the proof that Algorithm 2 achieves approximation factor of at most

$$\frac{d - \epsilon\delta}{2} = \frac{d - \frac{1}{31850496}}{2} = \frac{d}{2} - \frac{1}{63700992}.$$

By scaling and truncating the weight function , we obtain a polynomial time $\frac{d}{2} - \frac{1}{63700992} + \epsilon'$-approximation algorithm for any $\epsilon' > 0$, whereby the running time depends polynomially on $\frac{1}{\epsilon'}$. In particular, setting $\epsilon' := \frac{1}{63700992}$, we get a polynomial time $\frac{d}{2}$-approximation algorithm. However, given the fact that the running time of (at least a straightforward implementation of) Algorithm 2 is in $\Omega(|V|^{(d-1)^2+(d-1)})$, this result remains of only theoretical interest for the time being.

4 Further Remarks

The proven result indicates that an approximation ratio of $\frac{d}{2}$ is not the end of the story of local improvement algorithms for the Maximum Weight Independent Set Problem in d-claw free graphs. This observation is inevitably followed by the question of how far one can still get with this approach. Concerning algorithms that only consider local improvements of some fixed constant size (possibly dependent on d), the result of Hurkens and Schrijver [9] implies a lower bound of $\frac{d-1}{2}$ for $d \geq 4$. This raises the question of whether and how the gap between our result, providing an approximation guarantee of $\frac{d}{2} - \frac{1}{63700992} + \epsilon'$ for any $\epsilon' > 0$, and the lower bound of $\frac{d-1}{2}$ can be closed. Although the choice of our constants ϵ and δ still permits some room for optimization, as the rather rough estimates in the proof of the properties (1) to (11) indicate, the more critical ones among them still seem to be "tight enough" to limit hope for an improvement in an entirely different order of magnitude. Therefore, we also picked our constants in a way keeping the proof of (1)-(11) as short as possible. Some further ideas might be required to get substantially closer to an approximation factor of $\frac{d-1}{2}$. Whether or not the latter is possible could be regarded as a worthwhile subject for further research.

References

1 Esther M. Arkin and Refael Hassin. On local search for weighted k-set packing. *Mathematics of Operations Research*, 23(3):640–648, 1998. doi:10.1287/moor.23.3.640.

2 Piotr Berman. A d/2 Approximation for Maximum Weight Independent Set in d-Claw Free Graphs. In *Scandinavian Workshop on Algorithm Theory*, pages 214–219. Springer, 2000. doi:10.1007/3-540-44985-X_19.

3 Barun Chandra and Magnús M. Halldórsson. Greedy Local Improvement and Weighted Set Packing Approximation. *Journal of Algorithms*, 39(2):223–240, 2001. doi:10.1006/jagm.2000.1155.

4 Marek Cygan. Improved Approximation for 3-Dimensional Matching via Bounded Pathwidth Local Search. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 509–518. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.61.

5 Marek Cygan, Fabrizio Grandoni, and Monaldo Mastrolilli. How to Sell Hyperedges: The Hypermatching Assignment Problem. In *Proceedings of the 2013 Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 342–351. SIAM, 2013. doi:10.1137/1.9781611973105.25.

6 Martin Fürer and Huiwen Yu. Approximating the k-Set Packing Problem by Local Improvements. In *International Symposium on Combinatorial Optimization*, pages 408–420. Springer, 2014. doi:10.1007/978-3-319-09174-7_35.

7 Magnús M. Halldórsson. Approximating Discrete Collections via Local Improvements. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, page 160–169, USA, 1995. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=313651.313687.

8 Elad Hazan, Shmuel Safra, and Oded Schwartz. On the complexity of approximating k-Set Packing. *Computational Complexity*, 15:20–39, 2006. doi:10.1007/s00037-006-0205-6.

9 Cor A. J. Hurkens and Alexander Schrijver. On the size of systems of sets every t of which have an SDR, with an application to the worst-case ratio of heuristics for packing problems. *SIAM Journal on Discrete Mathematics*, 2(1):68–72, 1989. doi:10.1137/0402008.

10 George J. Minty. On Maximal Independent Sets of Vertices in Claw-Free Graphs. *Journal of Combinatorial Theory, Series B*, 28(3):284–304, 1980. doi:10.1016/0095-8956(80)90074-X.

11 Daishin Nakamura and Akihisa Tamura. A revision of Minty's algorithm for finding a maximum weight stable set of a claw-free graph. *Journal of the Operations Research Society of Japan*, 44(2):194–204, 2001. doi:10.15807/jorsj.44.194.

12 Najiba Sbihi. Algorithme de recherche d'un stable de cardinalité maximum dans un graphe sans étoile. *Discrete Mathematics*, 29(1):53–76, 1980. doi:10.1016/0012-365X(90)90287-R.

13 Maxim Sviridenko and Justin Ward. Large Neighborhood Local Search for the Maximum Set Packing Problem. In *International Colloquium on Automata, Languages, and Programming*, pages 792–803. Springer, 2013. doi:10.1007/978-3-642-39206-1_67.

A Inequalities Satisfied by Our Choice of ϵ and δ

- $4 - 2 \cdot \frac{6-9\sqrt{\epsilon}}{4-10\sqrt{\epsilon}} - 9\sqrt{\epsilon} \geq \frac{49}{50}$ (1)
- $9 \cdot (4\sqrt{\epsilon} + 5\epsilon) < 1$ (2)
- $(1+\sqrt{\epsilon}) \cdot \left(1 - \delta - \frac{25}{12} \cdot \epsilon\delta\right) + \frac{3d}{4} \cdot \left(\delta + \frac{25}{12} \cdot \epsilon\delta\right) + \epsilon\delta \leq \frac{d-\epsilon\delta}{2}$ (3)
- $36\sqrt{\epsilon} + 45\epsilon \leq \frac{1}{32}$ (4)
- $0 < \epsilon < \frac{16}{100} < \frac{1}{4}$ (5)
- $1 - 3\sqrt{\epsilon} > \frac{1}{2}$ (6)
- $1 + \sqrt{\epsilon} < \frac{3d}{4}$ (7)
- $4 \cdot \left(1 - \frac{3}{2} \cdot \sqrt{\epsilon}\right) \cdot (1 - \sqrt{\epsilon}) \geq 3 > \frac{149}{50}$ (8)
- $\frac{49 \cdot (1-\epsilon)}{100} \geq \frac{12}{25}$ (9)
- $(2 - 10\sqrt{\epsilon}) \cdot \frac{6-9\sqrt{\epsilon}}{4-10\sqrt{\epsilon}} \geq \frac{149}{50}$ (10)
- $\min\{2 - 10\sqrt{\epsilon}, 6 - 9\sqrt{\epsilon}, 4 - 10\sqrt{\epsilon}\} = 2 - 10\sqrt{\epsilon} > 0$ (11)

The proofs of these inequalities are omitted due to page limit.

B Some Propositions and Proofs Omitted in the Main Body

The following proposition is helpful to bound the sizes of candidate local improvements we consider during the analysis. It is a direct consequence of d-claw freeness.

▶ **Proposition 33.** *For any $v \in A$, we have $|N(v, A^*)| \leq d-1$ and for any $u \in A^*$, $|N(u, A)| \leq d-1$.*

Proof of Lemma 18. If $\sum_{u \in T_v} \text{charge}(u, v) \leq \frac{1-\epsilon}{2} \cdot w(v)$, we are done, so assume the contrary, i.e.

$$\sum_{u \in T_v} \text{charge}(u, v) > \frac{1-\epsilon}{2} \cdot w(v). \tag{12}$$

We have $|T_v| \subseteq N(v, A^*)$ by definition, so $|T_v| \leq d-1$ by Proposition 33. As Algorithm 2 has terminated, T_v does not yield a local improvement of w^2 and we know that

$$\sum_{u \in T_v} w^2(u) = w^2(T_v) \leq w^2(N(T_v, A)) \leq w^2(v) + \sum_{u \in T_v} w^2(N(u, A) \backslash \{v\}),$$

and the outer inequality is equivalent to

$$\sum_{u \in T_v} w^2(u) - w^2(N(u, A) \backslash \{v\}) \leq w^2(v). \tag{13}$$

By Lemma 7, we know that if $\text{charge}(u, v) > 0$ (which is the case for all $u \in T_v$ by definition), we have

$$w^2(u) - w^2(N(u, A) \backslash \{v\}) \geq 2 \cdot \text{charge}(u, v) \cdot w(v). \tag{14}$$

As $w(v) > 0$, for $u \in T_v$, let $\epsilon_u \geq 0$ such that

$$w^2(u) - w^2(N(u,A)\backslash\{v\}) = 2 \cdot \text{charge}(u,v) \cdot w(v) + \epsilon_u \cdot w^2(v). \tag{15}$$

Then (12) and (13) imply

$$\begin{aligned} w^2(v) &\geq \sum_{u \in T_v} w^2(u) - w^2(N(u,A)\backslash\{v\}) \\ &= \sum_{u \in T_v} 2 \cdot \text{charge}(u,v) \cdot w(v) + \epsilon_u \cdot w(v)^2 \\ &> \quad 2 \cdot \frac{1-\epsilon}{2} \cdot w^2(v) + \sum_{u \in T_v} \epsilon_u \cdot w^2(v) \\ &= w^2(v) \cdot \left(1 - \epsilon + \sum_{u \in T_v} \epsilon_u\right), \end{aligned}$$

and $w(v) > 0$ yields

$$\sum_{u \in T_v} \epsilon_u \leq \epsilon. \tag{16}$$

We now show that for each $u \in T_v$, one of the conditions listed in the lemma applies: Pick $u \in T_v$. By definition of charges, we know that $v = n(u)$ is a neighbor of u in A of maximum weight, implying

$$\begin{aligned} w^2(N(u,A)\backslash\{v\}) &= \sum_{x \in N(u,A)\backslash\{v\}} w^2(x) \\ &\leq \sum_{x \in N(u,A)\backslash\{v\}} w(x) \cdot \max\{0, \max_{y \in N(u,A)\backslash\{v\}} w(y)\} \\ &= (w(N(u,A)) - w(v)) \cdot \max\{0, \max_{y \in N(u,A)\backslash\{v\}} w(y)\}, \end{aligned} \tag{17}$$

whereby $\max \emptyset := -\infty$. By (15), we therefore obtain

$$\begin{aligned} & w^2(u) - w^2(N(u,A)\backslash\{v\}) &&= 2 \cdot \text{charge}(u,v) \cdot w(v) + \epsilon_u \cdot w^2(v) \\ \Leftrightarrow\ & w^2(u) - w^2(N(u,A)\backslash\{v\}) &&= (2 \cdot w(u) - w(N(u,A))) \cdot w(v) \\ & && \quad + \epsilon_u \cdot w^2(v) \\ \Leftrightarrow\ & w^2(u) + w^2(v) - w^2(N(u,A)\backslash\{v\}) &&= (2 \cdot w(u) + w(v) - w(N(u,A))) \cdot w(v) \\ & && \quad + \epsilon_u \cdot w^2(v), \end{aligned}$$

which results in

$$(w(u) - w(v))^2 - w^2(N(u,A)\backslash\{v\}) + (w(N(u,A)) - w(v)) \cdot w(v) = \epsilon_u \cdot w^2(v).$$

Applying (17) yields

$$(w(u) - w(v))^2 + (w(N(u,A)) - w(v)) \cdot (w(v) - \max\{0, \max_{y \in N(u,A)\backslash\{v\}} w(y)\}) \leq \epsilon_u \cdot w^2(v). \tag{18}$$

As both summands in (18) are nonnegative since real squares are nonnegative, $v \in N(u,A)$ is of maximum weight and $w > 0$, (18) in particular implies that both

$$\epsilon_u \cdot w^2(v) \geq (w(u) - w(v))^2 \text{ and} \tag{19}$$

$$\epsilon_u \cdot w^2(v) \geq (w(N(u,A)) - w(v)) \cdot (w(v) - \max\{0, \max_{y \in N(u,A)\backslash\{v\}} w(y)\}). \tag{20}$$

From (19), we can infer that $|w(u) - w(v)| \leq \sqrt{\epsilon_u} \cdot w(v)$, which in turn implies that

$$w(u) \leq w(v) + |w(u) - w(v)| \leq (1 + \sqrt{\epsilon_u}) \cdot w(v) \text{ as well as}$$
$$w(v) \leq w(u) + |w(v) - w(u)| \leq w(u) + \sqrt{\epsilon_u} \cdot w(v),$$

which yields $(1 - \sqrt{\epsilon_u}) \cdot w(v) \leq w(u)$. As a consequence, by (16), we obtain

$$\frac{w(u)}{w(v)} \in [1 - \sqrt{\epsilon_u}, 1 + \sqrt{\epsilon_u}] \subseteq [1 - \sqrt{\epsilon}, 1 + \sqrt{\epsilon}]. \tag{21}$$

In addition to that, (20) tells us that at least one of the two inequalities

$$\sqrt{\epsilon_u} \cdot w(v) \geq w(v) - \max\{0, \max_{y \in N(u,A) \setminus \{v\}} w(y)\} \text{ or} \tag{22}$$
$$\sqrt{\epsilon_u} \cdot w(v) \geq w(N(u,A)) - w(v) \tag{23}$$

must hold. If (22) applies, the fact that $\epsilon_u \leq \epsilon < 1$ by (5) and (16), together with $w(v) > 0$, implies that $N(u,A) \setminus \{v\} \neq \emptyset$, so let $v_2 \in N(u,A) \setminus \{v\}$ be of maximum weight. Then

$$w(v) - w(v_2) \leq \sqrt{\epsilon_u} \cdot w(v) \text{ and hence}$$
$$(1 - \sqrt{\epsilon}) \cdot w(v) \leq (1 - \sqrt{\epsilon_u}) \cdot w(v) \leq w(v_2) \leq w(v) \tag{24}$$

by maximality of $w(v)$ in $N(u,A)$. From this, we also get

$$(2 - \sqrt{\epsilon}) \cdot w(v) \leq w(v) + w(v_2) \leq w(N(u,A)) < 2 \cdot w(u),$$

whereby the last inequality follows from the fact that u sends positive charges to v. Hence, together with (21) and (24), all conditions for u being double are fulfilled. In case (23) holds true, we get

$$w(N(u,A)) \leq (1 + \sqrt{\epsilon_u}) \cdot w(v) \leq (1 + \sqrt{\epsilon}) \cdot w(v),$$

leaving us with a vertex that is single by (21).

In order to finally see that there can be at most one vertex $u \in T_v$ which is single, observe that for a single vertex u, we have

$$\begin{aligned}\text{charge}(u,v) = w(u) - \frac{w(N(u,A))}{2} &\geq (1 - \sqrt{\epsilon}) \cdot w(v) - \frac{1 + \sqrt{\epsilon}}{2} \cdot w(v) \\ &= \frac{1 - 3\sqrt{\epsilon}}{2} \cdot w(v).\end{aligned}$$

Hence, the existence of at least two single vertices in T_v and (6) would imply

$$\sum_{u \in T_v} \text{charge}(u,v) \geq (1 - 3\sqrt{\epsilon}) \cdot w(v) > \frac{w(v)}{2}$$

and (14), combined with the fact that $w(v) > 0$, would yield

$$\sum_{u \in T_v} w^2(u) - w^2(N(u,A) \setminus \{v\}) \geq \sum_{u \in T_v} 2 \cdot \text{charge}(u,v) \cdot w(v) > w^2(v),$$

a contradiction to (13). ◀

Proof of Lemma 19. We distinguish two cases, $w(v_1) \geq w(u)$ and $w(v_1) < w(u)$. Due to page limit, we only present the proof for the first, easier case.
Case 1: $w(v_1) \geq w(u)$. Then we have

$$\begin{aligned} 0 \leq w(N(u,A)) - w(v_1) &= 2 \cdot (w(u) - \text{charge}(u,v_1)) - w(v_1) \\ &= w(u) - 2 \cdot \text{charge}(u,v_1) + w(u) - w(v_1) \\ &\leq w(u) - 2 \cdot \text{charge}(u,v_1) \end{aligned}$$

and therefore

$$\begin{aligned} w^2(u) - w^2(N(u,A)\backslash\{v_1\}) &\geq w^2(u) - (w(N(u,A)) - w(v_1))^2 \\ &\geq w^2(u) - (w(u) - 2 \cdot \text{charge}(u,v_1))^2 \\ &= w^2(u) - w^2(u) + 4 \cdot w(u) \cdot \text{charge}(u,v_1) \\ &\quad - 4 \cdot \text{charge}(u,v_1)^2 \\ &= 4 \cdot \text{charge}(u,v_1) \cdot (w(u) - \text{charge}(u,v_1)). \end{aligned} \tag{25}$$

Given that for a double vertex, we have

$$\begin{aligned} \text{charge}(u,v_1) = w(u) - \frac{w(N(u,A))}{2} &\leq w(u) - \frac{2-\sqrt{\epsilon}}{2} \cdot w(v_1) \\ \leq w(u) - \frac{2-\sqrt{\epsilon}}{2(1+\sqrt{\epsilon})} \cdot w(u) &\leq w(u) - \frac{(2-\sqrt{\epsilon}) \cdot (1-\sqrt{\epsilon})}{2} \cdot w(u) \\ = w(u) \cdot \frac{2 - (2 - 3\sqrt{\epsilon} + \epsilon)}{2} &\leq \frac{3}{2} \cdot \sqrt{\epsilon} \cdot w(u) \end{aligned}$$

since $\frac{1}{1+\sqrt{\epsilon}} = 1 - \frac{\sqrt{\epsilon}}{1+\sqrt{\epsilon}} \geq 1 - \sqrt{\epsilon}$, (25) implies

$$w^2(u) - w^2(N(u,A)\backslash\{v_1\}) \geq 4 \cdot \left(1 - \frac{3}{2} \cdot \sqrt{\epsilon}\right) \cdot w(u) \cdot \text{charge}(u,v_1).$$

Further knowing that $w(u) \geq (1 - \sqrt{\epsilon}) \cdot w(v_1)$, we finally obtain

$$\begin{aligned} w^2(u) - w^2(N(u,A)\backslash\{v_1\}) &\geq 4 \cdot \left(1 - \frac{3}{2} \cdot \sqrt{\epsilon}\right) \cdot (1 - \sqrt{\epsilon}) \cdot w(v_1) \cdot \text{charge}(u,v_1) \\ &\geq \frac{149}{50} \cdot \text{charge}(u,v_1) \cdot w(v_1) \end{aligned}$$

by (8) as claimed. ◀

▶ **Proposition 34.** *$B \to B^*, v \mapsto t(v)$ is a bijection with inverse map $n \restriction B^*$.*

Proof of Lemma 31. Observing that $G[A' \cup A'^*]$ is d-claw free as an induced subgraph of G, Corollary 5 and Lemma 6 tell us that

$$\begin{aligned} w(A'^*) &\leq \sum_{u \in A'^*} \frac{w(N(u,A'))}{2} + \sum_{u \in A'^* : \text{charge}'(u,n'(u))>0} \text{charge}'(u,n'(u)) \\ &\leq \frac{d-1}{2} \cdot w(A') + \sum_{v \in A'} \sum_{u \in T'_v} \text{charge}'(u,v) \\ &\leq \frac{d-1}{2} \cdot w(A') + \sum_{v \in A'} \frac{d+2}{4} \cdot w(v) \\ &= \frac{d-1}{2} \cdot w(A') + \frac{d+2}{4} \cdot w(A') \\ &= \frac{3d}{4} \cdot w(A'). \end{aligned}$$

Moreover, by Lemma 18 and by definition of $t(v)$ for $v \in B$, we have

$$w(B^*) = w(\{t(v) : v \in B\}) \leq (1 + \sqrt{\epsilon}) \cdot w(B).$$

By assumption, we further know that $w(P) \leq \epsilon\delta \cdot w(A)$ as well as $w(B) \geq (1 - \delta - \frac{25}{12} \cdot \epsilon\delta) \cdot w(A)$ and $w(A') = w(A) - w(B)$. Putting everything together, we obtain

$$\begin{aligned} w(A^*) &= w(B^*) + w(A'^*) + w(P) \\ &\leq (1 + \sqrt{\epsilon}) \cdot w(B) + \frac{3d}{4} \cdot (w(A) - w(B)) + \epsilon\delta \cdot w(A) \\ &= \left(\frac{3d}{4} + \epsilon\delta\right) \cdot w(A) - \left(\frac{3d}{4} - (1 + \sqrt{\epsilon})\right) \cdot w(B) & \mid (7) \\ &\leq \left(\frac{3d}{4} + \epsilon\delta\right) \cdot w(A) - \left(\frac{3d}{4} - (1 + \sqrt{\epsilon})\right) \cdot \left(1 - \delta - \frac{25}{12} \cdot \epsilon\delta\right) \cdot w(A) \\ &= \left((1 + \sqrt{\epsilon}) \cdot \left(1 - \delta - \frac{25}{12} \cdot \epsilon\delta\right) + \frac{3d}{4} \cdot \left(\delta + \frac{25}{12} \cdot \epsilon\delta\right) + \epsilon\delta\right) \cdot w(A) & \mid (3) \\ &\leq \frac{d - \epsilon\delta}{2} \cdot w(A), \end{aligned}$$

which concludes the proof. ◀

Proof of Lemma 32. Assume that the assertion does not hold and pick $v_0 \in A'$ such that

$$\sum_{u \in T'_{v_0}} \text{charge}'(u, v_0) > \frac{d+2}{4} \cdot w(v_0).$$

Let $R := \{t(v) : v \in N(T'_{v_0}, B)\}$. We show that $T'_{v_0} \cup R$ yields a local improvement of $w^2(A)$, contradicting the termination criterion of our algorithm.

As $T'_{v_0} \subseteq N(v_0, A^*)$, Proposition 33 implies that $|T'_{v_0}| \leq d-1$. Given that for $u \in T'_{v_0} \subseteq A^*$, $N(u, B) \subseteq N(u, A)$ can contain at most $d - 1$ elements by Proposition 33, Proposition 34 implies that $|R| = |N(T'_{v_0}, B)| \leq (d-1)^2$. Hence, the total size of our improvement is at most $(d-1)^2 + (d-1)$.

As $\text{charge}'(u, v_0) > 0$ for all $u \in T'_{v_0}$, Lemma 7 shows that

$$w^2(u) - w^2(N(u, A') \backslash \{v_0\}) \geq 2 \cdot \text{charge}'(u, v_0) \cdot w(v_0)$$

for all $u \in T'_{v_0}$.

Additionally, for $u \in T'_{v_0}$ with $w(u) \geq 4 \cdot w(v_0)$, we get

$$2 \cdot w(u) - w(N(u, A')) = 2 \cdot \text{charge}'(u, v_0)$$

and therefore

$$w(N(u, A')) = 2 \cdot w(u) - 2 \cdot \text{charge}'(u, v_0).$$

As v_0 is the heaviest neighbor of u in A' by definition of charges, we further obtain

$$\begin{aligned} & w^2(N(u, A') \backslash \{v_0\}) \leq w^2(N(u, A')) \leq \sum_{v \in N(u, A')} w(v) \cdot w(v_0) \\ & = w(N(u, A')) \cdot w(v_0) = (2 \cdot w(u) - 2 \cdot \text{charge}'(u, v_0)) \cdot w(v_0) \\ & \leq 2 \cdot w(u) \cdot \frac{w(u)}{4} - 2 \cdot \text{charge}'(u, v_0) \cdot w(v_0) = \frac{w(u)^2}{2} - 2 \cdot \text{charge}'(u, v_0) \cdot w(v_0). \end{aligned}$$

As a consequence,

$$\frac{w(u)^2}{2} - w^2(N(u, A'\backslash\{v_0\})) \geq 2 \cdot \text{charge}'(u, v_0) \cdot w(v_0).$$

Let $S'_{v_0} := \{u \in T'_{v_0} : w(u) \geq 4 \cdot w(v_0)\}$. Then

$$\sum_{u \in T'_{v_0}} \text{charge}'(u, v_0) > \frac{d+2}{4} \cdot w(v_0),$$

together with the previous considerations and $w(v_0) > 0$, implies that

$$\begin{aligned}
&\sum_{u \in T'_{v_0}} w^2(u) - w^2(N(u, A')\backslash\{v_0\}) \\
&= \sum_{u \in S'_{v_0}} \frac{w^2(u)}{2} - w^2(N(u, A')\backslash\{v_0\}) + \sum_{u \in T'_{v_0}\backslash S'_{v_0}} w^2(u) - w^2(N(u, A')\backslash\{v_0\}) \\
&\quad + \sum_{u \in S'_{v_0}} \frac{w^2(u)}{2} \\
&\geq \sum_{u \in S'_{v_0}} 2 \cdot \text{charge}'(u, v_0) \cdot w(v_0) + \sum_{u \in T'_{v_0}\backslash S'_{v_0}} 2 \cdot \text{charge}'(u, v_0) \cdot w(v_0) \\
&\quad + \sum_{u \in S'_{v_0}} \frac{w^2(u)}{2} \\
&\quad = \sum_{u \in T'_{v_0}} 2 \cdot \text{charge}'(u, v_0) \cdot w(v_0) + \sum_{u \in S'_{v_0}} \frac{w^2(u)}{2} \\
&\quad > \left(1 + \frac{d}{2}\right) \cdot w^2(v_0) + \sum_{u \in S'_{v_0}} \frac{w^2(u)}{2}.
\end{aligned}$$

This implies

$$\sum_{u \in T'_{v_0}} w^2(u) > w^2(v_0) + \sum_{u \in T'_{v_0}} w^2(N(u, A')\backslash\{v_0\}) + \sum_{u \in S'_{v_0}} \frac{w^2(u)}{2} + \frac{d}{2} \cdot w^2(v_0)$$

and hence

$$\begin{aligned}
w^2(T'_{v_0}) &> w^2(N(T'_{v_0}, A')) + \sum_{u \in S'_{v_0}} \frac{w^2(u)}{2} + \frac{d}{2} \cdot w^2(v_0) \\
&\geq w^2(N(T'_{v_0}, A')) + \sum_{u \in S'_{v_0}} \frac{w^2(u)}{2} + \sum_{u \in T'_{v_0}\backslash S'_{v_0}} \frac{w^2(u)}{32} \\
&\geq w^2(N(T'_{v_0}, A')) + \sum_{u \in T'_{v_0}} \frac{w^2(u)}{32} \\
&= w^2(N(T'_{v_0}, A')) + \frac{1}{32} \cdot w^2(T'_{v_0}) \qquad (26)
\end{aligned}$$

since $|T'_{v_0}| \leq d - 1$ and $w(u) \leq 4 \cdot w(v_0)$ for $u \in T'_{v_0}\backslash S'_{v_0}$. We know that we can split the neighbors of $T'_{v_0} \cup R$ in A into the neighbors $N(T'_{v_0}, A')$ of T'_{v_0} in A', the neighbors $N(T'_{v_0}, B)$ of T'_{v_0} in B and the neighbors of R that we did not consider yet, i.e. $N(R, A)\backslash N(T'_{v_0}, A)$.

For $u \in R$ and $v := n(u) \in N(T'_{v_0}, B) \subseteq N(T'_{v_0}, A)$, we have $u = t(v)$ by Proposition 34 and $w(N(u, A)\backslash\{v\}) \leq \sqrt{\epsilon} \cdot w(v)$ by Lemma 28. This shows that

$$w^2(N(R, A)\backslash N(T'_{v_0}, A)) \leq \epsilon \cdot w^2(N(T'_{v_0}, B)).$$

As $T'_{v_0} \subseteq A'^* = A^*\backslash(B^* \cup P)$, we have

$$w^2(N(u, B)) \leq w^2(N(u, A)) \leq 9 \cdot w^2(u)$$

for all $u \in T'_{v_0}$, showing that

$$w^2(N(T'_{v_0}, B)) \leq w^2(N(T'_{v_0}, A)) \leq \sum_{u \in T'_{v_0}} w^2(N(u, A)) \leq 9 \sum_{u \in T'_{v_0}} w^2(u) = 9 \cdot w^2(T'_{v_0})$$

and hence

$$w^2(N(R, A)\backslash N(T'_{v_0}, A)) \leq \epsilon \cdot w^2(N(T'_{v_0}, B)) \leq 9\epsilon \cdot w^2(T'_{v_0}). \tag{27}$$

Finally, Lemma 29 and Proposition 34 yield

$$\begin{aligned} w^2(N(T'_{v_0}, B)) &\leq w^2(R) + (4\sqrt{\epsilon} + 4\epsilon) \cdot w^2(N(T'_{v_0}, B)) \\ &\leq w^2(R) + (4\sqrt{\epsilon} + 4\epsilon) \cdot 9 \cdot w^2(T'_{v_0}) \\ &= w^2(R) + (36\sqrt{\epsilon} + 36\epsilon) \cdot w^2(T'_{v_0}). \end{aligned} \tag{28}$$

Combining (26), (27) and (28), we get

$$\begin{aligned} w^2(N(T'_{v_0} \cup R, A)) = & \; w^2(N(T'_{v_0}, A')) + w^2(N(T'_{v_0}, B)) \\ & + w^2(N(R, A)\backslash N(T'_{v_0}, A)) \\ < & \; w^2(T'_{v_0}) - \frac{1}{32} \cdot w^2(T'_{v_0}) + w^2(R) \\ & + (36\sqrt{\epsilon} + 45\epsilon) \cdot w^2(T'_{v_0}) \\ \leq & \; w^2(T'_{v_0}) + w^2(R) - \left(\frac{1}{32} - (36\sqrt{\epsilon} + 45\epsilon)\right) w^2(T'_{v_0}) \\ \leq & \; w^2(T'_{v_0}) + w^2(R) \\ = & \; w^2(T'_{v_0} \cup R) \end{aligned}$$

by (4) and since $T'_{v_0} \subseteq A'^*$ and $R \subseteq B^*$ are disjoint. So we indeed get a local improvement of $w^2(A)$, a contradiction.

◀

Complexity of the List Homomorphism Problem in Hereditary Graph Classes

Karolina Okrasa

Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland
Institute of Informatics, University of Warsaw, Poland

Paweł Rzążewski

Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland
Institute of Informatics, University of Warsaw, Poland

Abstract

A homomorphism from a graph G to a graph H is an edge-preserving mapping from $V(G)$ to $V(H)$. For a fixed graph H, in the list homomorphism problem, denoted by LHom(H), we are given a graph G, whose every vertex v is equipped with a list $L(v) \subseteq V(H)$. We ask if there exists a homomorphism f from G to H, in which $f(v) \in L(v)$ for every $v \in V(G)$. Feder, Hell, and Huang [JGT 2003] proved that LHom(H) is polynomial time-solvable if H is a so-called bi-arc-graph, and NP-complete otherwise.

We are interested in the complexity of the LHom(H) problem in F-free graphs, i.e., graphs excluding a copy of some fixed graph F as an induced subgraph. It is known that if F is connected and is not a path nor a subdivided claw, then for every non-bi-arc graph the LHom(H) problem is NP-complete and cannot be solved in subexponential time, unless the ETH fails. We consider the remaining cases for connected graphs F.

If F is a path, we exhibit a full dichotomy. We define a class called predacious graphs and show that if H is not predacious, then for every fixed t the LHom(H) problem can be solved in quasi-polynomial time in P_t-free graphs. On the other hand, if H is predacious, then there exists t, such that the existence of a subexponential-time algorithm for LHom(H) in P_t-free graphs would violate the ETH.

If F is a subdivided claw, we show a full dichotomy in two important cases: for H being irreflexive (i.e., with no loops), and for H being reflexive (i.e., where every vertex has a loop). Unless the ETH fails, for irreflexive H the LHom(H) problem can be solved in subexponential time in graphs excluding a fixed subdivided claw if and only if H is non-predacious and triangle-free. On the other hand, if H is reflexive, then LHom(H) cannot be solved in subexponential time whenever H is not a bi-arc graph.

2012 ACM Subject Classification Mathematics of computing → Graph coloring; Theory of computation → Problems, reductions and completeness; Theory of computation → Graph algorithms analysis

Keywords and phrases list homomorphism, fine-grained complexity, hereditary graph classes

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.54

Related Version *Full Version*: `https://arxiv.org/abs/2010.03393` [30]

Funding *Karolina Okrasa*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme Grant Agreement no. 714704.
Paweł Rzążewski: Supported by Polish National Science Centre grant no. 2018/31/D/ST6/00062.

1 Introduction

Many natural graph-theoretic problems, including Independent Set, k-Coloring, Max Cut, Odd Cycle Transversal, etc., can be defined in a uniform way as the question of the existence of certain graph homomorphisms. For two graphs G and H, a function

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 54; pp. 54:1–54:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$f : V(G) \to V(H)$ is a *homomorphism from G to H* if for every $uv \in E(G)$ it holds that $f(u)f(v) \in E(H)$. If f is a homomorphism from G to H we denote it by $f : G \to H$. As an important special case, we observe that homomorphisms to K_k are precisely k-colorings of G. This is why homomorphisms to H are often called *H-colorings*. We will refer to the graph H as the *target* and to the vertices of H as *colors*. For fixed H, by $\textsc{Hom}(H)$ we denote the computational problem of deciding if an instance graph admits an H-coloring.

The complexity dichotomy for $\textsc{Hom}(H)$ was shown by Hell and Nešetřil [21]: If H is bipartite or has a vertex with a loop, then the problem is polynomial-time-solvable, and otherwise it is NP-complete. The study of variants of graph homomorphisms has attracted a significant attention [2, 26, 7, 8, 16, 15]. Arguably, the most natural generalization of the problem is the *list homomorphism problem*. For fixed H, an instance of the $\textsc{LHom}(H)$ problem is a pair (G, L), where G is a graph and L is a function that to every vertex $v \in V(G)$ assigns its *H-list* (or *list*) $L(v) \subseteq V(H)$. We ask if there exists a homomorphism $f : G \to H$, such that for every $v \in V(G)$ it holds that $f(v) \in L(v)$. We write $f : (G, L) \to H$ if f is a list homomorphism from G to H which respects the lists L, and we write $(G, L) \to H$ to indicate that some such f exists.

The complexity classification for $\textsc{LHom}(H)$ was proven in three steps. First, Feder and Hell [11] considered reflexive target graphs H, i.e., where every vertex has a loop. In this case $\textsc{LHom}(H)$ is polynomial-time solvable if H is an interval graph and NP-complete otherwise. Then, Feder et al. [12] showed the dichotomy in the case that H is irreflexive, i.e., has no loops. This problem appears to be polynomial-time solvable if H is bipartite and its complement is a circular-arc graph, and NP-complete otherwise. Finally, Feder et al. [13] defined a new class of graphs with possible loops, called *bi-arc graphs*, and showed that if H is a bi-arc graph, then $\textsc{LHom}(H)$ can be solved in polynomial time, and otherwise the problem is NP-complete. Reflexive bi-arc graphs coincide with interval graphs, and irreflexive bi-arc graphs are precisely bipartite graphs whose complement is a circular-arc graph. Let us point out that all mentioned hardness reductions for $\textsc{LHom}(H)$ also exclude the existence of a subexponential-time algorithm, unless the ETH fails.

An active line of research is to study the complexity of computational problems, when the instance is assumed to belong some specific graph class. We usually assume that the considered classes are *hereditary*, i.e., closed under vertex deletion. Each such a hereditary class can be characterized by a (possibly infinite) set of forbidden induced subgraphs. For a family $\mathcal{F}$ of graphs, a graph is *$\mathcal{F}$-free* if it does not contain any member of $\mathcal{F}$ as an induced subgraph. Most attention is put into considering classes with only one forbidden subgraph, i.e., for $\mathcal{F} = \{F\}$. In this case we write F-free, instead of $\{F\}$-free. We will always assume that F is connected.

Let us define two important families of graphs. For an integer $t \geqslant 1$, by P_t we denote the path with t vertices. For $a, b, c \geqslant 0$, by $S_{a,b,c}$ we denote the graph obtained by taking three disjoint paths P_{a+1}, P_{b+1}, and P_{c+1} and merging one of the endvertices of each path into one vertex. Note that if at least one of a, b, c is equal to 0, then $S_{a,b,c}$ is an induced path. The members of $\{S_{a,b,c} \mid a, b, c \geqslant 0\}$ are called *subdivided claws*.

Let us briefly discuss the complexity of k-$\textsc{Coloring}$ in F-free graphs. First, we observe that if F is not a path, then for every fixed $k \geqslant 3$, the k-$\textsc{Coloring}$ remains NP-complete in F-free graphs. Indeed, Emden-Weinert et al. [10] proved that the problem is hard for graphs with no cycles shorter than p, for any constant p. Setting $p = |V(F)| + 1$ yields the hardness for F-free graphs whenever F contains a cycle. On the other hand, k-$\textsc{Coloring}$ is NP-complete in line graphs [23, 27], which are in particular $S_{1,1,1}$-free. This implies the hardness for F-free graphs if F is a tree with maximum degree at least 3. Combining these, we conclude that the only connected graphs F, for which we might hope for a polynomial-time algorithm for k-Coloring in F-free graphs, are paths.

The complexity of k-Coloring in P_t-free graphs has been an active area of research in the last two decades, see the survey by Golovach et al. [18]. The current state of art is as follows. We know that for each fixed k, the problem is polynomial-time-solvable in P_5-free graphs [22]. On the other hand, for every $k \geqslant 5$, the problem is NP-complete in P_6-free graphs [24]. The complexity of 4-Coloring in P_t-free graphs is also fully understood: it is polynomial-time solvable for $t \leqslant 6$ [34] and NP-complete for $t \geqslant 7$ [24]. Finally, we know that 3-Coloring admits a polynomial time algorithm in P_7-free graphs [1]. Interestingly, we know no proof of NP-hardness of 3-Coloring in P_t-free graphs, for any value of t. The problem is believed to be solvable in polynomial time for every t, and obtaining such an algorithm is one of the main open questions in the area.

Let us point out that all mentioned hardness proofs rule out the existence of subexponential-time algorithms, unless the ETH fails. Furthermore, all algorithmic results hold even for List k-Coloring, except for the case $(k,t) = (4,6)$, which is NP-complete in the list setting [19].

Even though our current toolbox seems to be insufficient to solve 3-Coloring in P_t-free graphs in polynomial time for all t, we can still solve the problem significantly faster than for general graphs. Groenland et al. [20] showed an algorithm with running time $2^{\mathcal{O}(\sqrt{n \log n})}$, for all fixed t. Very recently, Pilipczuk et al. [33] observed that the breakthough algorithm for Independent Set in P_t-free graphs by Gartland and Lokshtanov [17], could be adapted to solve 3-Coloring in time $n^{\mathcal{O}(\log^3 n)}$. They also presented an arguably simpler algorithm with running time $n^{\mathcal{O}(\log^2 n)}$.

The complexity of the Hom(H) and LHom(H) problems in F-free graphs received a lot less attention [14, 25]. On the negative side, Piecyk and Rzążewski [32], showed that if F is connected and is not a subdivided claw, then for every non-bi-arc H, the LHom(H) problem remains NP-complete in F-free graphs and cannot be solved in subexponential time, assuming the ETH.

There are several results about the complexity of LHom(H) in P_t-free graphs. First, Chudnovsky et al. [3] showed that for $k \in \{5, 7, 9\} \cup [10; \infty)$, the LHom($C_k$) problem can be solved in polynomial time for P_9-free graphs. Very recently, Chudnovsky et al. [4] studied some further generalization of the homomorphism problem in subclasses of P_6-free graphs. Furthermore, the already mentioned $2^{\mathcal{O}(\sqrt{n \log n})}$-time algorithm by Groenland et al. [20] actually works for LHom(H) for a large family of graphs H: the requirement is that H does not contain two vertices with two common neighbors. Even more generally, the algorithm can solve a *weighted homomorphism problem*, where, in addition to lists, we allow vertex- and edge-weights. Later, Okrasa and Rzążewski [31] proved that the weighted homomorphism problem cannot be solved in P_t-free graphs in subexponential time, whenever the target graph has two vertices with two common neighbors. However, for some of the hardness reductions it was essential to exploit the existence of vertex- and edge-weights and thus they cannot be translated to the arguably more natural LHom(H) problem.

Our results. In this paper we investigate the fine-grained complexity of LHom(H) in F-free graphs, where F is a subdivided claw. Recall that these are the only connected forbidden graphs for which we can hope for the existence of subexponential-time algorithms.

First, we define the family of *predacious* graphs, and show that they precisely correspond to "hard" cases of LHom(H) in P_t-free graphs. More specifically, we prove the following theorem.

▶ **Theorem 1.** *Let H be a fixed graph.*

a) *If H is not predacious, then for every t, the* LHom*(H) problem can be solved in time $n^{\mathcal{O}(\log^2 n)}$ in n-vertex P_t-free graphs.*

b) *If H is predacious, then there exists t, such that the* LHom*(H) problem cannot be solved in time $2^{o(n)}$ in n-vertex P_t-free graphs, unless the ETH fails.*

The definition of predacious graphs is based on the decomposition theorem by Okrasa et al. [28] that is particularly useful for solving the LHom(H) problem. Using this theorem, each graph H can be decomposed into a family of induced subgraphs, called *factors*. Now, a graph H is predacious, if it has a factor that is simultaneously non-bi-arc and contains a *predator*: two vertices a_1, a_2 with two common neighbors b_1, b_2, such that a_1 and a_2 have incomparable neighborhoods and b_1 and b_2 have incomparable neighborhoods. Note that a predator is a refinement of the essential structure in the dichotomy for the weighted homomophism problem [20, 31].

The proof of Theorem 1 a) builds on the already mentioned decomposition of target graphs by Okrasa et al. [28] and on the recent quasi-polynomial-time algorithm for 3-Coloring P_t-free graphs [33]. The hardness counterpart is proven in two steps. First, we consider a special case that H is bipartite and "undecomposable" (the exact meaning of this is given in Section 2). Okrasa et al. [28] analyzed the structure of such graphs and showed that it is rich enough to build a number of useful gadgets. We use them as building blocks of gadgets required in our hardness reduction. Then, we lift this hardness result to general predacious graphs H, using the idea of *associated bipartite graphs* [13].

Next, we turn our attention to the case that F is an arbitrary subdivided claw. We obtain the dichotomy in two important special cases: that H is irreflexive, and that H is reflexive. Recall that these cases correspond to the first two steps of the complexity dichotomy for LHom(H) [11, 12].

As a warm-up, let us discuss the case that H is irreflexive and F is the simplest subdivided claw, i.e., the claw $S_{1,1,1}$. Recall that 3-Coloring is NP-complete in line graphs [23], which are in particular claw-free. Since the reduction yields an ETH lower bound, we obtain that if H contains a simple triangle, then LHom(H) cannot be solved in subexponential time in claw-free graphs.

So let us consider the case that H is triangle-free. We note that there is no homomorphism $K_3 \to H$, so if the instance graph contains a triangle, we can immediately report a no-instance. On the other hand, $\{S_{1,1,1}, K_3\}$-free graphs are just collections of disjoint paths and cycles, where the problem can be solved in polynomial time using dynamic programming. We generalize this simple classification to the case if F is an arbitrary subdivided claw as follows.

▶ **Theorem 2.** *Let H be a fixed irreflexive graph.*

a) *If H is non-predacious and triangle-free, then for every a, b, c, the* LHom*(H) problem can be solved in time $2^{\mathcal{O}(n^{8/9} \log n)}$ in n-vertex $S_{a,b,c}$-free graphs.*

b) *If H is predacious or contains a triangle, then there exist a, b, c, such that the* LHom*(H) problem cannot be solved in time $2^{o(n)}$ in n-vertex $S_{a,b,c}$-free graphs, unless the ETH fails.*

The algorithm from Theorem 2 a) is based on the existence of the so-called *extended strip decomposition* [6]. A similar approach was used by Chudnovsky et al. [5] to obtain a QPTAS and a subexponential-time algorithm for the Max Independent Set problem in $S_{a,b,c}$-free graphs. However, the decomposition itself is not structured enough to be useful for coloring problems, such as LHom(H). We proceed as follows. First, similarly as before, we restrict ourselves to instances that are $\{S_{a,b,c}, K_3\}$-free. We analyze the structure of such graphs G and show that they admit an extended strip decomposition with a very simple structure.

Very roughly speaking, we can find a "small" set $X \subseteq V(G)$, such that for each connected component C of $G - X$, the vertices of C can be partitioned into "small" sets called *atoms*, that can be arranged in a path-like or cycle-like manner. We exhaustively guess the coloring of X (which is fine, as X is small). For each atom we solve the problem recursively. Finally, we use the path-like or cycle-like arrangement of atoms to combine partial results using dynamic programming, similarly as we did for $\{S_{1,1,1}, K_3\}$-free graphs.

Let us point out that the assumption that H is irreflexive and triangle-free is only used to ensure that the instance is triangle-free. For such instances we can solve LHom(H) in subexponential time for *every* non-predacious graph H.

The hardness counterpart of Theorem 2 is simple. If H is predacious, then we are done by Theorem 1 b), as every P_t-free graph is also $S_{t,t,t}$-free. On the other hand, if H contains a simple triangle, then the problem is hard even in claw-free graphs, as mentioned before.

Finally, we show that for reflexive H the only "easy" cases are bi-arc graphs.

▶ **Theorem 3.** *For every fixed reflexive non-bi-arc graph H, there exist a, b, c, such that the LHom(H) problem cannot be solved in time $2^{o(n)}$ in n-vertex $S_{a,b,c}$-free graphs, unless the ETH fails.*

Unfortunately, we were not able to provide the full complexity dichotomy for $S_{a,b,c}$-free graphs. We conjecture that the distinction between "easy" and "hard" cases is as follows.

▶ **Conjecture 4.** *Assume the ETH. Let H be a non-bi-arc graph. Then for all a, b, c, the LHom(H) problem can be solved in time $2^{o(n)}$ in n-vertex $S_{a,b,c}$-free graphs if and only if none of the following conditions is satisfied:*

a) *H is predacious,*

b) *H contains a simple triangle,*

c) *has a factor that is not bi-arc and contains two incomparable vertices with loops.*

Full version of the paper. The proofs of some statements, marked with (♣), are omitted or just sketched. Complete proofs can be found in the full version of the paper [30].

2 Notation and preliminaries

For a positive integer n, by $[n]$ we denote the set $\{1, 2, \ldots, n\}$. For a set X and integer k, by 2^X we denote the family of all subsets of X and by $\binom{X}{k}$ (resp. $\binom{X}{\leqslant k}$) we denote the family of all subsets of X with exactly (resp. at most) k elements.

For two sets $X, Y \subseteq V(G)$, we say that X is *complete* to Y if every vertex from X is adjacent to every vertex from Y. For $v \in V(G)$, by $N_G(v)$ we denote the set of neighbors of v and by $N_G[v]$ we denote the set $N_G(v) \cup \{v\}$. Note that if v has a loop, then $v \in N_G(v)$, so $N_G(v) = N_G[v]$. We omit the subscript and write $N(v)$ and $N[v]$, respectively, if G is clear from the context.

We say that two vertices u, v of G are *incomparable* if $N(u) \not\subseteq N(v)$ and $N(v) \not\subseteq N(u)$. We say that a set S of vertices is *incomparable* if its elements are pairwise incomparable. Let H be a graph and suppose that there are two distinct vertices a, b of H, such that $N_H(a) \subseteq N_H(b)$. We observe that in any homomorphism to H, if some vertex is mapped to a, we can safely remap it to b. Thus, if for some instance (G, L) of the LHom(H) problem and for some $v \in V(G)$ the list $L(v)$ contains a and b as above, then we can safely remove a from $L(v)$. Thus, without loss of generality, we can always assume that in any instance of LHom(H) each list is an incomparable set in H.

For a graph H, by H^* we denote the bipartite graph with vertex set $\{a', a'' \mid a \in V(H)\}$ and edge set $\{a'b'' \mid ab \in E(H)\}$. We observe that H^* is connected if and only if H is connected and non-bipartite. Moreover, for bipartite H, the graph H^* consists of two disjoint copies of H. Feder et al. [13] proved that H is a bi-arc graph if and only H^* is a bi-arc graph. As H^* is bipartite, we can equivalently say that H is a bi-arc graph if and only if the complement of H^* is a circular-arc graph.

▶ **Definition 5** (Predator). *A* predator *is a tuple (a_1, a_2, b_1, b_2) of vertices, such that $a_1 \neq a_2, b_1 \neq b_2$, and $\{a_1, a_2\}$ and $\{b_1, b_2\}$ are incomparable sets, complete to each other.*

Figure 1 shows some examples of predators. Let us point out that the leftmost structure in Figure 1 is the only predator, which can be bipartite. It will play a special role in our hardness proofs; we call it an *incomparable* C_4. Observe that (a_1, a_2, b_1, b_2) is a predator in H, for some $a_1, a_2, b_1, b_2 \in V(H)$, if and only if $(a_1', a_2', b_1'', b_2'')$ is an incomparable C_4 in H^*. This implies the following observation.

▶ **Observation 6.** *A graph H contains a predator if and only if H^* contains an incomparable C_4.*

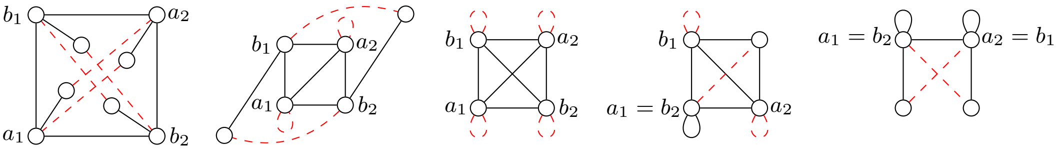


Figure 1 Examples of predators (a_1, a_2, b_1, b_2) and their neighbors. Red dashed lines denote the edges that cannot exist. The edges that are not drawn are possible, but not necessary.

We say that H is a *strong split graph* if $V(H)$ can be partitioned into two sets, P and B, such that $H[P]$ is a reflexive clique and B is independent.

For a bipartite graph H with bipartition classes X, Y, a *bipartite decomposition* is a partition of $V(H)$ into an ordered triple of sets (D, N, R), such that (i) N is non-empty and separates D and R, (ii) $|D \cap X| \geqslant 2$ or $|D \cap Y| \geqslant 2$, (iii) $(D \cup N) \cap X$ is complete to $N \cap Y$ and $(D \cup N) \cap Y$ is complete to $N \cap X$. We say that H is *undecomposable* if it admits no bipartite decomposition.

▶ **Theorem 7** (Okrasa et al. [28, 29]). *Let H be a graph. In time $|V(H)|^{\mathcal{O}(1)}$ we can construct a family $\mathcal{H}$ of $\mathcal{O}(|V(H)|)$ connected graphs, called* factors *of H, such that:*

(1) *H is a bi-arc graph if and only if every $H' \in \mathcal{H}$ is a bi-arc graph,*

(2) *for each $H' \in \mathcal{H}$, the graph H'^* is an induced subgraph of H^* and:*

  a. *H' is a bi-arc graph, or*

  b. *H' a strong split graph and has an induced subgraph H'', which is not a bi-arc graph and is an induced subgraph of H, or*

  c. *$(H')^*$ is undecomposable,*

(3) *for every instance (G, L) of* LHom*(H), the following implication holds:*
If there exists a non-decreasing, convex function $f \colon \mathbb{N} \to \mathbb{R}$, such that for every $H' \in \mathcal{H}$, for every induced subgraph G' of G, and for every H'-lists L' on G', we can decide whether $(G', L') \to H'$ in time $f(|V(G')|)$, then we can solve the instance (G, L) in time $\mathcal{O}\left(|V(H)| f(n) + n^2 \cdot |V(H)|^3\right)$.

Now we are ready to define the class of predacious graphs.

▶ **Definition 8** (Predacious graphs). *Let H be a graph and let $\mathcal{H}$ be the family of factors of H. We say that H is* predacious *if there exists $H' \in \mathcal{H}$ that is not a bi-arc graph and contains a predator.*

3 P_t-free graphs

3.1 Quasi-polynomial-time algorithm

We observe that to obtain Theorem 1 a), it is sufficient to prove the following.

▶ **Theorem 9.** *Let H be a fixed graph that does not contain a predator. Then for every t, the* LHom(H) *problem can be solved in time $n^{\mathcal{O}(\log^2 n)}$ in n-vertex P_t-free graphs.*

Indeed, suppose we have proven Theorem 9 and consider a non-predacious graph H, let $\mathcal{H}$ be the family of its factors given by Theorem 7. Since H is non-predacious, every $H' \in \mathcal{H}$ is either a bi-arc graph, or does not contain a predator. Thus, for each H' we can solve the LHom(H') problem in P_t-free graphs in polynomial time (in the first case) or in time $n^{\mathcal{O}(\log^2 n)}$, using Theorem 9 (in the second case). Now Theorem 1 a) follows from Theorem 7 (3).

Before we proceed to the proof of Theorem 9, let us show one crucial property of graphs H.

▶ **Observation 10.** *Let H be a graph which does not contain a predator. For any incomparable sets $X, Y \subseteq V(H)$, each of size at least 2, there exist $x \in X$ and $y \in Y$ such that $xy \notin E(H)$.*

Proof. For contradiction, suppose that there are two incomparable sets X, Y, each of size at least 2, which are complete to each other. Let x_1, x_2 be distinct elements from X, and y_1, y_2 be distinct elements from Y. Then (x_1, x_2, y_1, y_2) is a predator. ◀

So let us now prove Theorem 9. The algorithm follows the algorithm for 3-Coloring by Pilipczuk et al. [33], which is in turn inspired by the work of Gartland and Lokshtanov [17].

Sketch of proof of Theorem 9. Let (G, L) be an instance of LHom(H), such that graph G is P_t-free. We start with a preprocessing phase, in which we exhaustively perform the following steps. (1) If for some $v \in V(G)$ it holds that $L(v) = \emptyset$, then we terminate and report a no-instance. (2) If for some $v \in V(G)$, the list $L(v)$ contains two vertices $x, y \in V(H)$, such that $N_H(x) \subseteq N_H(y)$, then we remove x from $L(v)$. (3) If for some edge $uv \in E(G)$, and some $x \in L(u)$, the vertex x is non-adjacent in H to every $y \in L(v)$, then we remove x from $L(u)$. (4) If for some $v \in V(G)$ we have $|L(v)| = 1$, we remove v from G. Note that by the previous step the lists of neighbors of v contain only neighbors of the vertex in $L(v)$. (5) We enumerate all $S \in \binom{V(G)}{\leqslant t}$ and all possible H-colorings of $(G[S], L)$. If for some $v \in V(G)$ and some $x \in L(v)$, for some $S \in \binom{V(G)}{\leqslant t}$ such that $v \in S$ there is no $h : (G[S], L) \to H$ such that $h(v) = x$, we remove x from $L(v)$.

We will continue calling the current instance (G, L), let n be its number of vertices of G. The instance satisfies the following properties.

(P1) For every $v \in V(G)$, the set $L(v)$ is incomparable and has at least two elements.

(P2) For every $v \in V(G)$, every $S \in \binom{V(G)}{\leqslant t}$, such that $v \in S$, and every $x \in L(v)$, there exists $h : (G[S], L) \to H$ which maps v to x.

Now let us describe the algorithm. If $n \leqslant 1$, then we report a yes-instance; recall that by property (P1) each list is non-empty. If the instance G is disconnected, we call the algorithm for each connected component independently. If none of the above cases occurs, we perform branching. We will carefully choose a *branching pair* (v, x), where $v \in V(G)$ and $x \in L(v)$, and branch into two possibilities. In the first one, called the *successful* branch, we call the algorithm recursively with the list of v set to $\{x\}$. In the second branch we call the algorithm with x removed from $L(v)$. We report a yes-instance if at least one of the branches reports a yes-instance.

Now let us discuss how we select a branching pair. For each $\{u, u'\} \in \binom{V(G)}{2}$ we define the *bucket* $\mathcal{B}_{u,u'}$. The elements of $\mathcal{B}_{u,u'}$ are all possible pairs (P, h), where P is an induced u-u'-path and h is a list homomorphism from (P, L) to H. We will refer to pairs (P, h) as *colored paths*.

Note that since G is P_t-free, the total size of all buckets is $\mathcal{O}(n^t)$ and they can be enumerated in polynomial time. Furthermore, by property (P2), we know that $\mathcal{B}_{u,u'}$ is non-empty if and only if u and u' are in the same connected component of G. Even more, if w belongs to an induced u-u'-path P, and $x \in L(w)$, then $\mathcal{B}_{u,u'}$ contains a colored path (P, h), such that $h(w) = x$.

Define

$$\delta := \frac{1}{2^{|V(H)|+1} \cdot t} \qquad \text{and} \qquad \varepsilon := \frac{1}{2^{|V(H)|+1} \cdot |V(H)|^t \cdot t} = \frac{\delta}{|V(H)|^t}.$$

▷ Claim 11. If G is a connected P_t-free graph, then there is a pair (v, x), where $v \in V(G)$ and $x \in L(v)$, with the following property. There is a set $Q \subseteq \binom{V(G)}{2}$ of size at least $\delta \cdot \binom{n}{2}$, such that for every $\{u, u'\} \in Q$ there is a subset $\mathcal{P}_{u,u'} \subseteq \mathcal{B}_{u,u'}$ of size at least $\varepsilon \cdot |\mathcal{B}_{u,u'}|$, such that for every $(P, h) \in \mathcal{P}_{u,u'}$, there is $w_P \in V(P) \cap N[v]$, such that $h(w_P) \notin N_H(x)$.

Proof. For $\{u, u'\} \in \binom{V(G)}{2}$, let $\theta(u, u')$ denote the number of induced u-u'-paths in G. By [33, Lemma 5], there is a vertex $v \in V(G)$, such that for at least $\frac{1}{2t}\binom{n}{2}$ pairs $\{u, u'\} \in \binom{V(G)}{2}$ and for at least $\frac{1}{2t}\,\theta(u, u')$ induced u-u'-paths P, the set $N[v]$ intersects $V(P)$. Since the number of distinct H-lists is at most $2^{|V(H)|}$, we observe that by the pigeonhole principle there is a list $L' \subseteq V(H)$ and a subset $Q \subseteq \binom{V(H)}{2}$ of size at least $\frac{1}{2^{|V(H)|+1} \cdot t}\binom{n}{2} = \delta \cdot \binom{n}{2}$, such that for every $\{u, u'\} \in Q$ there exists a set $\mathcal{P}_{u,u'}$ of at least $\delta \cdot \theta(u, u')$ induced u-u'-paths, with the property that for every $P \in \mathcal{P}_{u,u'}$ there exists $w_P \in N[v] \cap V(P)$, such that $L(w_P) = L'$.

By property (P1) we know that each of $L(v)$ and L' is an incomparable set with at least two elements. Thus by Observation 10 there are $x \in L(v)$ and $y \in L'$, which are non-adjacent in H.

Let us argue that the pair (v, x) satisfies the desired conditions. Fix some $\{u, u'\} \in Q$. As every induced u-u' path has at most $t - 1$ elements, we have that $|\mathcal{B}_{u,u'}| \leqslant |V(H)|^t \cdot \theta(u, u')$. On the other hand, by property (P2) for every $P \in \mathcal{P}_{u,u'}$ there exists a homomorphism $h : (P, L) \to H$ such that $h(w_P) = y \notin N_H(x)$. So, summing up, we obtain that the number of such pairs $(P, h) \in \mathcal{B}_{u,u'}$ is at least $|\mathcal{P}_{u,u'}| \geqslant \delta \cdot \theta(u, u') \geqslant \frac{\delta}{|V(H)|^t} \cdot |\mathcal{B}_{u,u'}| = \varepsilon \cdot |\mathcal{B}_{u,u'}|$. ◁

Consider the successful branch for the branching pair (v, x) given by Claim 11. For some $\{u, u'\} \in Q$, let (P, h) be a colored path in $\mathcal{P}_{u,u'}$, and let w_P be as in the claim. Consider the preprocessing phase of the current call. If $w_P = v$, then w_P is removed from the graph, so (P, h) will no longer appear in the bucket of $\{u, u'\}$. Similarly, if $w_P \neq v$, then we remove $h(w_P)$ from $L(w_P)$, so (P, h) will not appear in the bucket of $\{u, u'\}$. Thus when we branch using the pair (v, x), in the successful branch we remove an ε-fraction of elements in a δ-fraction of buckets. This gives the quasi-polynomial running time, we refer to the full version of the paper for a detailed complexity analysis (♣). ◀

3.2 Hardness results for P_t-free graphs

Let H be a predacious graph and let $\mathcal{H}$ be the family of factors of H. Since H is predacious, there is some non-bi-arc factor $H' \in \mathcal{H}$, which contains a predator. By Theorem 7 (2) there are two possible cases:

Case A. H' is a strong split graph as in Theorem 7 (2b) (every such graph H' contains a predator, but we will not use it explicitly), and

Case B. $(H')^*$ is an undecomposable induced subgraph of H^*.

Case A: Strong split target graphs. We show that for strong split graphs H' the LHom(H') problem remains hard even if the instance is a split graph, i.e., its vertex set can be partitioned into a clique and an independent set. Equivalently, split graphs are $\{C_4, C_5, 2P_2\}$-free graphs.

▶ **Theorem 12.** *Let H' be a fixed non-bi-arc strong split graph. Then the* LHom(H') *problem cannot be solved in time $2^{o(n)}$ in n-vertex split graphs, unless the ETH fails.*

Proof. Let P be the set of vertices in H' that have loops, and let B be the set of vertices of H' without loops. Consider an instance (G, L) of LHom(H'). Recall that without loss of generality we can assume that each list $L(v)$ is an incomparable set. As for every $p \in P$ and $b \in B$ it holds that $N_{H'}(b) \subseteq N_{H'}(p)$, no vertex in G has both a vertex from P and a vertex from B in its list. Since every list is non-empty, we can partition the vertex set of $V(G)$ into two sets:

$$X := \{v \in V(G) \mid L(v) \cap P \neq \emptyset\} \quad \text{and} \quad Y := \{v \in V(G) \mid L(v) \cap B \neq \emptyset\}.$$

Furthermore, as B is independent, we can assume that Y is independent; otherwise (G, L) is a no-instance. Let G' be obtained from G by adding all edges with both endvertices in X (except for loops). It is straightforward to verify that $(G, L) \to H'$ if and only if $(G', L) \to H'$. ◀

Now we can show the main result of this subsection.

Proof of Theorem 1 b) in Case A. Let H be as in Case A and let H', H'' be as in Theorem 7 (2b). Since H'' is an induced subgraph of H', it is also a strong split graph, so by Theorem 12 we know that LHom(H'') admits no subexponential-time algorithm in split graphs. As H'' is an induced subgraph of H, every instance of LHom(H'') is also an instance of LHom(H), and we are done. ◀

Case B: Target graphs with the associated bipartite graph undecomposable. First we consider bipartite, undecomposable, non-bi-arc graphs H, which contain a predator. Recall that the only bipartite predator is an incomparable C_4. We will prove the following.

▶ **Theorem 13.** *Let H be a fixed, bipartite, non-bi-arc, undecomposable graph, which contains an incomparable C_4. Then there exists t, such that* LHom(H) *cannot be solved in time $2^{o(n)}$ in n-vertex P_t-free graphs, unless the ETH fails.*

Before we proceed to the proof of Theorem 13, we need to introduce some tools which we will need. For a pair of vertices (a, b) of $V(H)$, an $\mathrm{OR}_3(a, b)$*-gadget* is an instance (F, L) of LHom(H) with *interface vertices* $o_1, o_2, o_3 \in V(F)$, such that $L(o_1) = L(o_2) = L(o_3) = \{a, b\}$, and

$$\{f(o_1)f(o_2)f(o_3) \mid f : (F, L) \to H\} = \{aaa, aab, aba, baa, abb, bab, bba\}.$$

For an incomparable set of vertices S, such that $|S| \geqslant 2$, a $\mathrm{NEQ}(S)$*-gadget* is an instance (F, L) of LHom(H) with *interface vertices* $s_1, s_2 \in V(F)$, such that $L(s_1) = L(s_2) = S$, and

$$\{f(s_1)f(s_2) \mid f : (F, L) \to H\} = \{uv \mid u, v \in S, u \neq v\}.$$

The following structural result is proven by Okrasa et al. [29, Lemma 19 and Corollary 20].

▶ **Lemma 14** (Okrasa et al. [29]). *Let H be a connected, bipartite, non-bi-arc, undecomposable graph with bipartition classes X and Y. Then there exist two incomparable sets of vertices $\{\alpha, \beta\} \subseteq X$ and $\{\alpha', \beta'\} \subseteq Y$, such that $\alpha\alpha', \beta\beta' \in E(H)$, $\alpha\beta', \beta\alpha' \notin E(H)$, and the following conditions hold.*

(1) *For any incomparable two-element set $\{a, b\} \subseteq V(H)$, and for any $\{\gamma, \delta\} \in \{\{\alpha, \beta\}, \{\alpha', \beta'\}\}$, such that $\{a, b, \gamma, \delta\}$ is contained in one bipartition class, there exist a path $D^{\gamma/\delta}_{a/b}$ with endvertices x, y and H-lists L, such that $L(x) = \{a, b\}$, $L(y) = \{\gamma, \delta\}$, and:*

- **(D1)** *there is a list homomorphism $h_a : (D^{\gamma/\delta}_{a/b}, L) \to H$, such that $h_a(x) = a$ and $h_a(y) = \gamma$,*
- **(D2)** *there is a list homomorphism $h_b : (D^{\gamma/\delta}_{a/b}, L) \to H$, such that $h_b(x) = b$ and $h_b(y) = \delta$,*
- **(D3)** *there is no list homomorphism $h : (D^{\gamma/\delta}_{a/b}, L) \to H$, such that $h(x) = a$ and $h(y) = \delta$.*

(2) *There exist an $\mathrm{OR}_3(\alpha, \beta)$-gadget and an $\mathrm{OR}_3(\alpha', \beta')$-gadget.*

▶ **Lemma 15** ([29]). *Let H be a connected, bipartite, non-bi-arc, undecomposable graph, let $S \subseteq V(H)$ be an incomparable set contained in one bipartition class of H. Then there exists a $\mathrm{NEQ}(S)$-gadget.*

We use Lemma 14 and Lemma 15 to construct the so-called *occurrence gadget.*

▶ **Lemma 16.** *Let H be a connected, bipartite, non-bi-arc and undecomposable graph, and let $\{a, b\}$, γ, δ be as in Lemma 14 (1). Then there is a $\mathrm{Var}(a, b)$-gadget (G, L) with interface vertices v, t, f, such that $L(v) = \{a, b\}$, $L(t) = L(f) = \{\gamma, \delta\}$, and:*

(1) *for any homomorphism $h : (G, L) \to H$, if $h(v) = a$, then $h(t) = \gamma$ and $h(f) = \delta$,*

(2) *for any homomorphism $h : (G, L) \to H$, if $h(v) = b$, then $h(t) = \delta$ and $h(f) = \gamma$.*

Proof. We use Lemma 14 to construct gadgets $(D^{\gamma/\delta}_{a/b}, L)$ and $(D^{\gamma/\delta}_{b/a}, L)$ with endvertices, respectively, $x_1, y_1 \in V(D^{\gamma/\delta}_{a/b})$ and $x_2, y_2 \in V(D^{\gamma/\delta}_{b/a})$. We then use Lemma 15 for $S = \{\gamma, \delta\}$ to construct a $\mathrm{NEQ}(S)$-gadget (F, L) with interface vertices $s_1, s_2 \in V(F)$.

We identify vertices x_1 and x_2 into a single vertex v. We identify vertices y_1 and s_1 into a single vertex t, and we identify vertices y_2 and s_2 into a single vertex f, see Figure 2 (top left). ◀

We proceed to the proof of Theorem 13.

Proof of Theorem 13. Let (a_1, a_2, b_1, b_2) be an incomparable C_4 in H. Let X and Y be the bipartition classes of H, so that $a_1, a_2 \in X$ and $b_1, b_2 \in Y$.

We reduce from 3-Sat. Consider a formula Φ of 3-Sat with variables $x_1, \ldots, x_N$ and clauses $C_1, \ldots, C_M$. We can assume that each clause has exactly three literals. We construct an instance (G_Φ, L) of LHom(H) as follows. We introduce a biclique with partite sets $V := \{v_1, \ldots, v_N\}$ and $U := \{u_1, \ldots, u_{3M}\}$. Vertices in V correspond to the variables of Φ, while vertices in U correspond to literals in Φ, i.e., the occurrences of the variables in clauses. For a clause C_i, by U_i we denote the three-element subset of vertices of U corresponding to the literals of C_i. For every $j \in [N]$ we set $L(v_j) := \{a_1, a_2\}$ and for every $i \in [3M]$ we set $L(u_i) := \{b_1, b_2\}$.

Mapping the vertex v_j to a_1 (a_2, resp.) will correspond to making the variable v_j true (false, resp.). Similarly, we will interpret u_j being mapped to b_1 (b_2, resp.) as setting the corresponding literal true (false, resp.). So we need to ensure that (i) the coloring of vertices in V is consistent with the coloring of vertices in U, and (ii) for each clause C_i, at least one vertex in U_i is mapped to b_1.

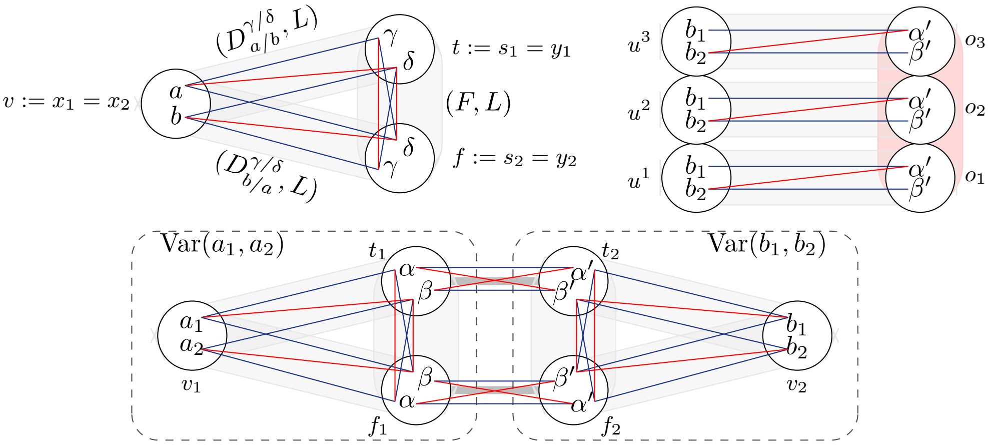


Figure 2 A schematic view of a $\mathrm{Var}(a, b)$-gadget (top left), an $\mathrm{OR}_3(b_1, b_2)$-gadget (top right) and a positive occurrence gadget (bottom). On every picture, the blue lines indicate that there exists an H-coloring of the respective part of the graph, which assigns chosen values to white vertices, and the red ones indicate that there is no such H-coloring. The red area indicates an $\mathrm{OR}_3(\alpha', \beta')$-gadget with interface vertices o_1, o_2, o_3.

To ensure property (i), we will introduce two types of *occurrence gadgets*. We use Lemma 16 to construct two variable gadgets $\mathrm{Var}(a_1, a_2)$ and $\mathrm{Var}(b_1, b_2)$ and add an edge between their t-vertices and another one between f-vertices. This way we obtain a *positive occurrence gadget*, see Figure 2 (bottom). A *negative occurrence gadget* is obtained from a positive occurrence gadget by adding a copy of a $\mathrm{NEQ}(\{b_1, b_2\})$-gadget, constructed by Lemma 15, with interface vertices s_1, s_2, and identifying s_1 with v_2. The occurrence gadgets have two special vertices: a *variable vertex* v_1, and a *literal vertex*, which is v_2 for the positive occurrence gadget, and s_2 for the negative occurrence gadget. Consider a vertex $u_i \in U$, which corresponds to an occurrence of a variable x_j, and thus to the vertex v_j. If u_i corresponds to a positive (resp., negative) literal, we introduce a positive (resp., negative) occurrence gadget, and identify v_j with its variable vertex and u_i with its literal vertex. One can readily verify that the constructed gadgets can indeed be used to ensure property (i).

Consider a set $U_i = \{u^1, u^2, u^3\}$, corresponding to the literals of some clause C_i. We observe that in order to ensure property (ii), we need to construct an $\mathrm{OR}_3(b_1, b_2)$-gadget, whose interface vertices are precisely u^1, u^2, and u^3. We call Lemma 14 to construct an $\mathrm{OR}_3(\alpha', \beta')$-gadget with interface vertices o_1, o_2, o_3 and three copies of the graph $D^{\beta'/\alpha'}_{b_2/b_1}$. For $s \in \{1, 2, 3\}$, we identify one endvertex of the s-th copy of $D^{\beta'/\alpha'}_{b_2/b_1}$ (the one with the list $\{b_1, b_2\}$) with u^s, and the other endvertex (the one with the list $\{\alpha', \beta'\}$) with o_s, see Figure 2 (top right). Again, it is straightforward to verify that the constructed subgraph is indeed an $\mathrm{OR}_3(b_1, b_2)$-gadget with interface vertices u^1, u^2, u^3.

The discussion above implies that $(G_\Phi, L) \to H$ if and only if Φ is satisfiable. Let t' be the maximum of the numbers of vertices in the negative occurrence gadget and in the $\mathrm{OR}_3(b_1, b_2)$-gadget and define $t := t' + 4$. By a simple case analysis it can be verified that G_Φ is P_t-free (♣). ◀

Finally, we can prove Theorem 1 b) in Case B.

Proof of Theorem 1 b) in Case B. For contradiction, suppose that there exists a graph H, satisfying the assumptions, and for every t there is an algorithm A_t, which solves every P_t-free instance of $\textsc{LHom}(H)$ in subexponential time. Let H' be a factor of H as in the assumptions of Case B and observe that H'^* satisfies the assumptions of Theorem 13. Let t be given by Theorem 13 for H'^*.

Let (G, L') be an instance of $\textsc{LHom}(H'^*)$ constructed as in the proof of Theorem 13. Since H'^* is an induced subgraph of H^*, (G, L') is also an instance of $\textsc{LHom}(H^*)$. Note that G is bipartite and P_t-free, and no list intersects both bipartition classes of H^*. Define $L : V(G) \to 2^{V(H)}$ as follows: $L(v) := \{a \mid \{a', a''\} \cap L(v) \neq \emptyset\}$. It is straightforward to verify that $(G, L') \to H^*$ if and only if $(G, L) \to H$ [29, Proposition 43]. Thus we can use A_t to decide if $(G, L) \to H$ or, equivalently, if $(G, L') \to H'^*$, in subexponential time. By Theorem 13 this contradicts the ETH. ◀

4 $S_{a,b,c}$-free graphs

4.1 Subexponential-time algorithm for $\{S_{a,b,c}, K_3\}$-free graphs

To describe the algorithm, we first need to introduce the notion of an *extended strip decomposition* [6, 5]. For a graph G, by $T(G)$ we denote the set of all triangles in G, i.e., three-element sets $\{x, y, z\}$ of pairwise adjacent vertices. We will denote a triangle $\{x, y, z\}$ shortly by xyz.

Let G be a simple graph. An *extended strip decomposition* (D, η) of G consists of:

- a simple graph D and a function $\eta : V(D) \cup E(D) \cup T(D) \to 2^{V(G)}$,
- for each $xy \in E(D)$, subsets $\eta(xy, x), \eta(xy, y) \subseteq \eta(xy)$,

which satisfy the following properties:

1. $\{\eta(o) \mid o \in V(D) \cup E(D) \cup T(D)\}$ is a partition of $V(G)$,
2. for every $x \in V(D)$ and every distinct $y, z \in N_D(x)$, the set $\eta(xy, x)$ is complete to $\eta(xz, x)$,
3. every $uv \in E(G)$ is contained in one of the sets $\eta(o)$ for $o \in V(D) \cup E(D) \cup T(D)$ or:
   - $u \in \eta(xy, x), v \in \eta(xz, x)$ for some $x \in V(D)$ and $y, z \in N_D(x)$, or
   - $u \in \eta(xy, x), v \in \eta(x)$ for some $xy \in E(D)$, or
   - $u \in \eta(xyz)$ and $v \in \eta(xy, x) \cap \eta(xy, y)$ for some $xyz \in T(D)$.

We will sometimes refer to elements of $V(D) \cup E(D) \cup T(D)$ as *objects* of D.

The following subsets of $V(G)$ are called *atoms* of a decomposition (D, η): (1) for an object $o \in V(D) \cup T(D)$, the set $\eta(o)$, (2) for $xy \in E(D)$, the set $\eta(xy) - (\eta(xy, x) \cup \eta(xy, y))$, the set $\eta(x) \cup \eta(xy) - \eta(xy, y)$, and the set $\eta(x) \cup \eta(y) \cup \eta(xy) \cup \bigcup_{xyz \in T(D)} \eta(xyz)$.

The following theorem is the main combinatorial tool used in our algorithm.

▶ **Theorem 17** (♣). *Let $t \geqslant 4$, $\sigma \in (0, \frac{1}{100t})$, and let G be a connected $(S_{t,t,t}, K_3)$-free graph on n vertices with $\Delta(G) < \sigma^8 \cdot n$. Then there exists $X \subseteq V(G)$ and an extended strip decomposition (D, η) of $G - X$ with each atom of size at most α, such that:*

(1) *$\alpha \leqslant (1 - \sigma^7)n$ and $|X| \leqslant \sigma(n - \alpha)$,*

(2) *$\eta(xyz) = \emptyset$ for every $xyz \in T(D)$,*

(3) *D is a simple graph with maximum degree at most 2,*

(4) *if for some edge xy of D we have $\eta(xy, x) = \emptyset$, then x is of degree 1 in D.*

Sketch of proof. By a result of Chudnovsky et al. [5, Lemma 6.5] there is $X \subseteq V(G)$ and an extended strip decomposition (D', η') of $G - X$, satisfying (1). We aim to modify (D', η') in order to obtain a decomposition with the desired structure. We will still denote the extended strip decomposition obtained after each step of modification as (D', η').

To ensure properties (2), (3), and (4), we use the fact that G is triangle-free. Let us start with (2) and consider a triangle xyz in D'. Define $A_{xy} := \eta(xy, x) \cap \eta(xy, y)$. Sets A_{yz} and A_{xz} are defined in an analogous way. Recall that the neighborhood of $\eta(xyz)$ is contained in $A_{xy} \cup A_{yz} \cup A_{xz}$. However, these three sets are complete to each other, so at least one of them must be empty. It turns out that we can "absorb" $\eta(xyz)$ into $\eta(o)$, where o is either a vertex or an edge of xyz, without violating the properties of an extended strip decomposition and increasing the maximum atom size.

Now let us discuss property (3). Suppose D' has a vertex x with at least three neighbors, say y, y', y''. As the sets $\eta(xy, x)$, $\eta(xy, x)$, $\eta(xy'', x)$ are pairwise complete to each other, at least one of them, say $\eta(xy, x)$, must be empty. We introduce a new vertex x' to D', add the edge $x'y$ with $\eta(x'y) := \eta(xy)$ and $\eta(x'y, y) := \eta(xy, y)$, and remove xy from D'. Observe that the degree of x was reduced by 1. We repeat this step exhaustively. Property (4) is ensured in a similar way. ◀

▶ **Theorem 18 (♣).** *Let H be a connected graph with no predator. Then for every $a, b, c \geqslant 0$, the* LHom*(H) problem can be solved in time $2^{\mathcal{O}(n^{8/9} \log n)}$ in n-vertex $\{S_{a,b,c}, K_3\}$-free graphs.*

Sketch of proof. We assume that n is large, as otherwise we solve the problem exhaustively. We will present a recursive algorithm. Let $F(n)$ be the running time bound on instances with n vertices.

Similarly as we did in Section 3.1, we can ensure that every list is an incomparable set of size at least two and for every $uv \in E(G)$ and every $a \in L(v)$ there exists $b \in L(u)$ such that $ab \in E(H)$.

First, suppose that exists a vertex $v \in V(G)$ such that $\deg_G(v) \geqslant n^{1/9}$. This implies that there exists a list L' assigned to at least $\ell := n^{1/9}/2^{|V(H)|}$ neighbors of v. By Observation 10 there exist $a \in L(v)$ and $b \in L'$ such that $ab \notin E(H)$. We branch on assigning a to v; either we remove a from $L(v)$ or color v with a and remove b from the lists of all neighbors of v. Since in the second branch at least ℓ lists are shortened, we obtain that the complexity in this case is $F(n) = 2^{\mathcal{O}(n^{8/9} \log n)}$.

Now suppose that the maximum degree of G is smaller than $n^{1/9}$. Theorem 17 called for $t := \max(a, b, c, 4)$ and $\sigma := n^{1-/9}$ yields $X \subseteq V(G)$ and an extended strip decomposition (D, η) of $G - X$, satisfying the conditions stated in the statement. Let α be the maximum size of an atom of (D, η). If $x \in V(D)$ has two neighbors y and z, then, by Theorem 17 (4), $\eta(xy, x) \neq \emptyset$ and $\eta(xz, x) \neq \emptyset$. As $\eta(xy, x)$ is complete to $\eta(xz, x)$ and the maximum degree of G is at most $n^{1/9}$, we observe that $|\eta(xy, x)| < n^{1/9}$ and $|\eta(xz, x)| < n^{1/9}$.

We proceed as follows. We exhaustively guess the H-coloring of vertices of X; there are at most $|V(H)|^{|X|}$ possibilities. In each branch we need to decide if the H-coloring of X can be extended to all vertices of G. For an edge $uv \in E(G)$, such that $u \in X$ and $v \notin X$, we remove from $L(v)$ every non-neighbor of the color of u. Now the problem is reduced to solving the instance of LHom(H) on each component G' of $G - X$ independently. We observe that $V(G') \subseteq \bigcup_{o \in V(D') \cup E(D') \cup T(D')} \eta(o)$ for some connected component D' of D. Recall that D' is a path or a cycle.

▷ Claim 19 (♣). We can solve the instance (G', L) of LHom(H) in time $|V(H)|^{4n^{1/9}} \cdot F(\alpha) \cdot n^{\mathcal{O}(1)}$.

Sketch of Proof. Suppose D' is a path with consecutive vertices $x_1, \ldots, x_m$. If $m \leqslant 2$, then $|V(G')| \leqslant \alpha$ and we solve the problem recursively in time $F(\alpha)$. Otherwise, for every edge $x_i x_{i+1}$, except for $x_1 x_2$ and $x_{m-1} x_m$, we enumerate all pairs (f, g), such that $f : (G[\eta(x_i x_{i+1}, x_i)], L) \to H$ and $g : (G[\eta(x_i x_{i+1}, x_{i+1})], L) \to H$. Now for each such pair we verify if the partial H-coloring given by f and g can be extended to an H-coloring of $G([\eta(x_i x_{i+1})], L)$. We can do it in time $F(\alpha)$ by recursively solving an appropriate instance of LHom(H). Similarly, for each vertex x_i, except for x_1, x_m, we enumerate all list H-colorings of $\eta(x_{i-1}x_i, x_i)$ and $\eta(x_i x_{i+1}, x_i)$ and recursively check if they can be extended to a homomorphism $(G[\eta(x_i)], L) \to H$. To deal with the extremities of D', for each coloring of $\eta(x_1x_2, x_2)$ (resp. $\eta(x_{m-1}x_m, x_{m-1})$) we test if it can be extended to an H-coloring of $(G[\eta(x_1) \cup \eta(x_1x_2)], L)$ (resp. $(G[\eta(x_m) \cup \eta(x_{m-1}x_m)], L)$); note that each of these instances has at most α vertices. Then we use dynamic programming to decide if $(G', L) \to H$.

Suppose now that D' is a cycle $x_1, x_2, \ldots, x_m$. We guess the coloring of $\eta(x_1x_2, x_1)$, adjust the lists of its neighbors, and remove $\eta(x_1x_2, x_1)$ from G'. We modify D' by introducing a vertex x_1' and the edge $x_1'x_2$ to D', and removing the edge x_1x_2. Sets η are modified as in the proof of Claim 19. After the modification D' is a path and we continue as in the previous case. ◁

So let us now estimate the total running time in case that the maximum degree of G is at most $n^{1/9}$. Recall that we exhaustively guess the coloring of $|X|$ and then, for every connected component of $G - X$, we try to extend it, using Claim 19. Thus the overall complexity $F(n)$ in the considered case is described by $F(n) \leqslant |V(H)|^{|X|} \cdot |V(H)|^{4n^{1/9}} \cdot F(\alpha) \cdot n^{\mathcal{O}(1)}$. Applying $|X| \leqslant n^{-1/9}(n - \alpha)$ and $1 \leqslant \alpha \leqslant n - n^{2/9}$, and the inductive assumption, we conclude that $F(n) = 2^{\mathcal{O}(n^{8/9} \log n)}$. ◀

Combining Theorem 18 with Theorem 7, we immediately obtain the following corollary.

▶ **Corollary 20.** *Let H be a non-predacious graph. Then for every $a, b, c \geqslant 0$, the LHom(H) problem can be solved in time $2^{\mathcal{O}(n^{8/9} \log n)}$ in n-vertex $\{S_{a,b,c}, K_3\}$-free graphs.*

Now Theorem 2 a) follows from Corollary 20. Since H is irreflexive and triangle-free, if G is not triangle-free, we report a no-instance. In the other case, we use the algorithm from Corollary 20.

4.2 Hardness results

▶ **Theorem 3 b).** *Let H be a graph, which is predacious or contains a simple triangle. Then there is t, such that LHom(H) cannot be solved in time $2^{o(n)}$ in n-vertex $S_{t,t,t}$-free graphs, unless the ETH fails.*

Proof. The first case of the theorem follows directly from Theorem 1 b), as P_t-free graphs are $S_{t,t,t}$-free. The second case follows from the hardness of 3-Coloring in line graphs [23]. ◀

▶ **Theorem 21.** *Let H be a connected non-bi-arc graph such that H^* is undecomposable and there are three distinct vertices u_1, u_2, u_3 of H with loops, such that $S = \{u_1, u_2, u_3\}$ is incomparable. Then there is t, such that LHom(H) cannot be solved in time $2^{o(n)}$ in $S_{t,t,t}$-free graphs, unless the ETH fails.*

Proof. Let G be an instance of 3-Coloring with $V(G) = \{v_1, v_2, \ldots, v_N\}$. We construct an instance (G', L) of LHom(H) such that G is 3-colorable if and only if $(G', L) \to H$. First, for every $i \in [N]$ we introduce to G' a graph K^i, which is a complete graph with the vertex set $V(K^i) := \{x_{ij} \mid v_j \in N_G(v_i)\}$. Intuitively, the vertex x_{ij} represents the connection of v_i and v_j from the point of view of v_i. We set $L(x_{ij}) := S$ for all relevant i, j. Now, for each edge v_iv_j of G, we introduce a copy of the NEQ(S)-gadget given by Lemma 15, and identify its two interface vertices with x_{ij} and x_{ji}, respectively. Suppose for now that we can ensure the following property.

($\star$) For each $i \in [N]$ and each $f : (K^i, L) \to H$, all vertices of K^i are mapped to the same element of S, and for each $u \in S$ there is $f : (K^i, L) \to H$ that maps all vertices of K^i to u.

With the property above at hand, we can interpret the mapping of vertices in K^i as coloring v_i with one of three possible colors. The properties of the NEQ(S)-gadget imply that G is 3-colorable if and only if the constructed graph admits a list homomorphism to H.

Now let us argue how to ensure property ($\star$). For each $i \in [N]$ we add an independent set Q^i and make it complete to K^i. The size of Q^i and the lists of its vertices depend on the structure of H.

For $\{\ell, \ell', \ell''\} = [3]$, a *private neighbor* of $u_\ell \in S \subseteq V(H)$ is a vertex $w_\ell \in N(u_\ell) - (N(u_{\ell'}) \cup N(u_{\ell''}))$. Note that if u_ℓ does not have a private neighbor, then, since S is incomparable, there exist $w_{\ell\ell'} \in N(u_\ell) \cap N(u_{\ell'}) - N(u_{\ell''})$ and $w_{\ell\ell''} \in N(u_\ell) \cap N(u_{\ell''}) - N(u_{\ell'})$.

We consider three cases. If for each $\ell \in [3]$, the vertex u_ℓ has a private neighbor, then $Q^i := \{q^i\}$, and $L(q^i) := \{w_1, w_2, w_3\}$. Otherwise, if there are exactly two vertices in S which have private neighbors, say u_2 and u_3, we set $Q^i := \{q^i, r^i\}$, $L(q^i) := \{w_{12}, w_2, w_3\}$ and $L(r^i) := \{w_{13}, w_2, w_3\}$. Last, if there is at most one vertex in S which has private neighbors, say u_3, we set $Q^i := \{q^i, r^i, s^i\}$ and $L(q^i) := \{w_{12}, w_{13}\}$, $L(r^i) := \{w_{12}, w_{23}\}$, and $L(s^i) := \{w_{13}, w_{23}\}$. It is straightforward to verify that in each of the above cases the property ($\star$) holds.

That completes the construction of (G', L). By the reasoning above we observe that $(G', L) \to H$ if and only if G is 3-colorable. Let $t \geqslant 2$ be the number of vertices in the NEQ(S)-gadget given by Lemma 15. A straightforward analysis of the structure of G' implies that G' is $S_{t,t,t,}$-free (♣). ◀

With Theorem 21 at hand, we can prove Theorem 3.

Proof of Theorem 3. Let H' be a vertex-minimal induced non-bi-arc subgraph of H. Feder and Hell [11] proved that H' (i) is an induced cycle with at least four vertices, or (ii) consists of an independent set $\{x, y, z\}$ and three paths, each joining two vertices from $\{x, y, z\}$ and avoiding the neighborhood of the third one. The minimality of H' implies that H'^* is undecomposable (see e.g. [9]). Now observe that H' contains an incomparable set of size 3: in case (i) we can take any three vertices of H', and in case (ii) this set is $\{x, y, z\}$. Thus the claim follows from Theorem 21. ◀

5 Conclusion

Recall that while for P_t-free graphs, in Theorem 1 we were able to fully characterize the "easy" and "hard" cases of LHom(H), for the case of $S_{a,b,c}$-free graphs we obtained a full dichotomy only for irreflexive (Theorem 2) and for reflexive (Theorem 3) graphs H. In order to complete the dichotomy, we need to consider graphs H that are neither irreflexive nor reflexive. Some hardness results for such graphs follow already from Theorem 3 b) and Theorem 21. We were also able to obtain a few more ad-hoc hardness results, which we do not present here. All our results seem to support the following conjecture.

▶ **Conjecture 4.** *Assume the ETH. Let H be a non-bi-arc graph. Then for all a, b, c, the* LHom(H) *problem can be solved in time $2^{o(n)}$ in n-vertex $S_{a,b,c}$-free graphs if and only if none of the following conditions is satisfied:*

a) *H is predacious,*

b) *H contains a simple triangle,*

c) *has a factor that is not bi-arc and contains two incomparable vertices with loops.*

References

1 Flavia Bonomo, Maria Chudnovsky, Peter Maceli, Oliver Schaudt, Maya Stein, and Mingxian Zhong. Three-coloring and list three-coloring of graphs without induced paths on seven vertices. *Combinatorica*, 38(4):779–801, 2018. doi:10.1007/s00493-017-3553-8.

2 Andrei A. Bulatov. H-Coloring dichotomy revisited. *Theor. Comput. Sci.*, 349(1):31–39, 2005. doi:10.1016/j.tcs.2005.09.028.

3 Maria Chudnovsky, Shenwei Huang, Pawel Rzążewski, Sophie Spirkl, and Mingxian Zhong. Complexity of C_k-coloring in hereditary classes of graphs. 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 31:1–31:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.31.

4 Maria Chudnovsky, Jason King, Michał Pilipczuk, Paweł Rzążewski, and Sophie Spirkl. Finding large H-colorable subgraphs in hereditary graph classes. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 35:1–35:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.35.

5 Maria Chudnovsky, Marcin Pilipczuk, Michal Pilipczuk, and Stéphan Thomassé. Quasi-polynomial time approximation schemes for the maximum weight independent set problem in H-free graphs. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2260–2278. SIAM, 2020. doi:10.1137/1.9781611975994.139.

6 Maria Chudnovsky and Paul D. Seymour. The three-in-a-tree problem. *Combinatorica*, 30(4):387–417, 2010. doi:10.1007/s00493-010-2334-4.

7 Víctor Dalmau, László Egri, Pavol Hell, Benoit Larose, and Arash Rafiey. Descriptive complexity of list H-coloring problems in logspace: A refined dichotomy. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 487–498. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.52.

8 László Egri, Pavol Hell, Benoit Larose, and Arash Rafiey. Space complexity of list H-colouring: a dichotomy. 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 349–365. SIAM, 2014. doi:10.1137/1.9781611973402.26.

9 László Egri, Dániel Marx, and Paweł Rzążewski. Finding list homomorphisms from bounded-treewidth graphs to reflexive graphs: a complete complexity characterization. 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 27:1–27:15. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.27.

10 Thomas Emden-Weinert, Stefan Hougardy, and Bernd Kreuter. Uniquely colourable graphs and the hardness of colouring graphs of large girth. *Comb. Probab. Comput.*, 7(4):375–386, 1998. URL: http://journals.cambridge.org/action/displayAbstract?aid=46667.

11 Tomás Feder and Pavol Hell. List homomorphisms to reflexive graphs. *Journal of Combinatorial Theory, Series B*, 72(2):236–250, 1998. doi:10.1006/jctb.1997.1812.

12 Tomás Feder, Pavol Hell, and Jing Huang. List homomorphisms and circular arc graphs. *Combinatorica*, 19(4):487–505, 1999. doi:10.1007/s004939970003.

13 Tomás Feder, Pavol Hell, and Jing Huang. Bi-arc graphs and the complexity of list homomorphisms. *Journal of Graph Theory*, 42(1):61–80, 2003. doi:10.1002/jgt.10073.

14 Tomás Feder, Pavol Hell, and Jing Huang. List homomorphisms of graphs with bounded degrees. *Discrete Mathematics*, 307(3-5):386–392, 2007. doi:10.1016/j.disc.2005.09.030.

15 Tomás Feder, Pavol Hell, Sulamita Klein, and Rajeev Motwani. List partitions. *SIAM J. Discrete Math.*, 16(3):449–478, 2003. URL: http://epubs.siam.org/sam-bin/dbq/article/38405.

16 Tomás Feder, Pavol Hell, David G. Schell, and Juraj Stacho. Dichotomy for tree-structured trigraph list homomorphism problems. *Discrete Applied Mathematics*, 159(12):1217–1224, 2011. doi:10.1016/j.dam.2011.04.005.

17 Peter Gartland and Daniel Lokshtanov. Independent set on P_k-free graphs in quasi-polynomial time. *CoRR*, abs/2005.00690, 2020. To appear in FOCS 2020 Proc. arXiv:2005.00690.

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. doi:10.1002/jgt.22028.

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. doi:10.1016/j.ic.2014.02.004.

20 Carla Groenland, Karolina Okrasa, Paweł Rzążewski, Alex Scott, Paul Seymour, and Sophie Spirkl. H-colouring P_t-free graphs in subexponential time. *Discrete Applied Mathematics*, 267:184–189, 2019.

21 Pavol Hell and Jaroslav Nešetřil. On the complexity of H-coloring. *J. Comb. Theory, Ser. B*, 48(1):92–110, 1990. doi:10.1016/0095-8956(90)90132-J.

22 Chính T. Hoàng, Marcin Kaminski, Vadim V. Lozin, Joe Sawada, and Xiao Shu. Deciding k-colorability of P_5-free graphs in polynomial time. *Algorithmica*, 57(1):74–81, 2010. doi:10.1007/s00453-008-9197-8.

23 Ian Holyer. The NP-completeness of edge-coloring. *SIAM J. Comput.*, 10(4):718–720, 1981. doi:10.1137/0210055.

24 Shenwei Huang. Improved complexity results on k-coloring P_t-free graphs. *Eur. J. Comb.*, 51:336–346, 2016. doi:10.1016/j.ejc.2015.06.005.

25 Marcin Kamiński and Anna Pstrucha. Certifying coloring algorithms for graphs without long induced paths. *Discret. Appl. Math.*, 261:258–267, 2019. doi:10.1016/j.dam.2018.09.031.

26 Gábor Kun and Mario Szegedy. A new line of attack on the dichotomy conjecture. *Eur. J. Comb.*, 52:338–367, 2016. doi:10.1016/j.ejc.2015.07.011.

27 Daniel Leven and Zvi Galil. NP-completeness of finding the chromatic index of regular graphs. *J. Algorithms*, 4(1):35–44, 1983. doi:10.1016/0196-6774(83)90032-9.

28 Karolina Okrasa, Marta Piecyk, and Paweł Rzążewski. Full complexity classification of the list homomorphism problem for bounded-treewidth graphs. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 74:1–74:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.74.

29 Karolina Okrasa, Marta Piecyk, and Paweł Rzążewski. Full complexity classification of the list homomorphism problem for bounded-treewidth graphs. *CoRR*, abs/2006.11155, 2020. arXiv:2006.11155.

30 Karolina Okrasa and Paweł Rzążewski. Complexity of the list homomorphism problem in hereditary graph classes. *CoRR*, abs/2010.03393, 2020. arXiv:2010.03393.

31 Karolina Okrasa and Paweł Rzążewski. Subexponential algorithms for variants of the homomorphism problem in string graphs. *J. Comput. Syst. Sci.*, 109:126–144, 2020. doi:10.1016/j.jcss.2019.12.004.

32 Marta Piecyk and Paweł Rzążewski. Fine-grained complexity of the list homomorphism problem: feedback vertex set and cutwidth. *CoRR*, abs/2009.11642, 2020. Extended abstract available in STACS 2021 Proc. arXiv:2009.11642.

33 Michał Pilipczuk, Marcin Pilipczuk, and Paweł Rzążewski. Quasi-polynomial-time algorithm for Independent Set in P_t-free graphs via shrinking the space of induced paths. In Valerie King and Hung Viet Le, editors, *Proceedings of the Fourth SIAM Symposium on Simplicity in Algorithms (SOSA), Alexandria, Virginia, USA, January 11-12, 2021 (Virtual conference)*, pages 204–209. SIAM, 2021. doi:10.1137/1.9781611976496.23.

34 Sophie Spirkl, Maria Chudnovsky, and Mingxian Zhong. Four-coloring P_6-free graphs. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1239–1256. SIAM, 2019. doi:10.1137/1.9781611975482.76.

Spectrum Preserving Short Cycle Removal on Regular Graphs

Pedro Paredes
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

We describe a new method to remove short cycles on regular graphs while maintaining spectral bounds (the nontrivial eigenvalues of the adjacency matrix), as long as the graphs have certain combinatorial properties. These combinatorial properties are related to the number and distance between short cycles and are known to happen with high probability in uniformly random regular graphs.

Using this method we can show two results involving high girth spectral expander graphs. First, we show that given $d \geqslant 3$ and n, there exists an explicit distribution of d-regular $\Theta(n)$-vertex graphs where with high probability its samples have girth $\Omega(\log_{d-1} n)$ and are ϵ-near-Ramanujan; i.e., its eigenvalues are bounded in magnitude by $2\sqrt{d-1}+\epsilon$ (excluding the single trivial eigenvalue of d). Then, for every constant $d \geqslant 3$ and $\epsilon > 0$, we give a deterministic poly(n)-time algorithm that outputs a d-regular graph on $\Theta(n)$-vertices that is ϵ-near-Ramanujan and has girth $\Omega(\sqrt{\log n})$, based on the work of [26].

2012 ACM Subject Classification Mathematics of computing → Spectra of graphs; Theory of computation → Expander graphs and randomness extractors

Keywords and phrases Ramanujan Graphs, High Girth Regular Graphs

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.55

Related Version *Full Version*: https://arxiv.org/abs/2002.07211

Funding *Pedro Paredes*: 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).

Acknowledgements I am very grateful to Ryan O'Donnell for numerous comments and suggestions, as well as very thorough feedback on an earlier draft of this paper.

1 Introduction

Let's consider d-regular graphs of n vertices. The study of short cycles and *girth* (defined as the length of the shortest cycle of a graph) in such graphs dates back to at least the 1963 paper of Erdős and Sachs [10], who showed that there exists an infinite family with girth at least $(1 - o_n(1)) \log_{d-1} n$. On the converse side, a simple path counting argument known as the "Moore bound" shows that this girth is upper bounded by $(1 + o_n(1)) 2 \log_{d-1} n$. Though simple, this is the best known upper bound. Given these bounds, it is common to call an infinite family of d-regular n-vertex graphs *high girth* if their girth is $\Omega(\log_{d-1} n)$.

The first explicit construction of high girth regular graphs is attributed to Margulis [23], who gave a construction of graphs that achieve girth $(1 - o_n(1)) \frac{4}{9} \log_{d-1} n$. A series of works initiated by Lubotzky-Phillips-Sarnak [21] and then improved by several other people [24, 27, 19] culminated in the work of Dahan [9], who proves that for all large enough d there are explicit d-regular n-vertex graphs of girth $(1 - o_n(1)) \frac{4}{3} \log_{d-1} n$.

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 55; pp. 55:1–55:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Another relevant problem consists of generating random distributions that produce regular graphs with high girth. Results regarding the probabilistic aspects of certain structures (like cycles) in graphs often give us tools to count the number of graphs that satisfy certain conditions, like how many regular graphs have girth at least some value. The distribution of short cycles in uniformly random regular graphs was first studied by Bollobás [7], who proved, that for a fixed k the random variables representing the number of cycles of length exactly k in a uniformly random d-regular graph are asymptotically independent Poisson with mean $(d-1)^k/2k$. Subsequently, McKay-Wormald-Wysocka [25] gave a more precise description of this by finding the asymptotic probability of a random d-regular graph having a certain number of cycles of any length up to $c \log_{d-1} n$, for $c < 1/2$. More recently, Linial and Simkin [20] showed that a random greedy algorithm that is given $d \geqslant 3$, $c \in (0,1)$ and an even n, produces a d-regular n-vertex graph with girth at least $c \log_{d-1} n$ with high probability.

The literature of regular graphs with high girth is closely connected to the literature of *spectral expanders*. Before defining this, let's consider some notation.

▶ **Definition 1.** *Let G be an n-vertex d-regular multigraph. We write $\lambda_i = \lambda_i(G)$ for the eigenvalues of its adjacency matrix A_G, and we always assume they are ordered with $\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_n$. A basic fact is that $\lambda_1 = d$ always; this is called the trivial eigenvalue and corresponds to the all ones vector. We also write $\lambda(G) = \max\{\lambda_2, |\lambda_n|\}$.*

Roughly, a graph with good spectral expansion properties is a graph that has small λ. More formally, an infinite sequence (G_n) of d-regular graphs is called a family of expanders if there is a constant $\delta > 0$ such that $\lambda(G) \leqslant (1-\delta)d$ for all n, or in other words, all eigenvalues are strictly separated from the trivial eigenvalue. This terminology was first introduced by [29] and later it was shown [1] that uniformly random d-regular graphs are spectral expanders with high probability.

The celebrated Alon-Boppana bound shows that λ cannot be arbitrarily small:

▶ **Theorem 2** ([1, 28, 11])**.** *For any d-regular n-vertex graph G we have that $\lambda_2(G) \geqslant 2\sqrt{d-1} - O(1/\log^2 n)$.*

Using some number-theoretic ideas, Lubotzky-Phillips-Sarnak [21], and independently Margulis [24], proved this bound is essentially tight by showing the existence of infinite families of d-regular graphs that meet the bound $\lambda(G) \leqslant 2\sqrt{d-1}$, if $d-1$ is an odd prime. In light of this, Lubotzky-Phillips-Sarnak introduced the following definition:

▶ **Definition 3** (Ramanujan graphs)**.** *A d-regular graph G is called Ramanujan whenever $\lambda(G) \leqslant 2\sqrt{d-1}$.*

These results were improved by Morgenstern [27], who showed the same for all d where $d-1$ is a prime power.

It is still open whether there exist infinite families of Ramanujan graphs for all d. However, if one relaxes this to only seek ϵ-near-Ramanujan graphs (graphs that satisfy $\lambda \leqslant 2\sqrt{d-1}+\epsilon$), then the answer is positive. Friedman [12] proved that uniformly random d-regular n-vertex graphs satisfy $\lambda \leqslant 2\sqrt{d-1} + o_n(1)$ with high probability. This proof was recently simplified by Bordenave [8].

▶ **Theorem 4** ([12, 8])**.** *Fix any $d > 3$ and $\epsilon > 0$ and let G be a uniformly random d-regular n-vertex graph. Then*

$$\boldsymbol{Pr}\left[\lambda(G) \leqslant 2\sqrt{d-1} + \epsilon\right] \geqslant 1 - o_n(1).$$

In fact [8], G achieves the subconstant $\epsilon = \widetilde{O}(1/\log^2 n)$ with probability at least $1 - 1/n^{.99}$.

Recently, it was shown how to achieve a result like the above but deterministically [26]. We write a more precise statement of this below.

▶ **Theorem 5** ([26]). *Given any n, $d \geqslant 3$ and $\epsilon > 0$, there is a deterministic polynomial-time algorithm that constructs a d-regular N-vertex graph with the following properties:*

- $N = n(1 + o_n(1))$;
- $\lambda(G) \leqslant 2\sqrt{d-1} + \epsilon$;

We refer the reader interested in a more thorough history of the literature of Ramanujan graphs to the introduction of [26]. Also, for a comprehensive list of applications and connections of Ramanujan graphs and expanders to computer science and mathematics, see [14].

In this work we concern ourselves with bridging these two worlds, looking for families of regular graphs that are both good spectral expanders and also have high girth. This bridge can be seen in several of the aforementioned works. The explicit construction of high girth regular graphs by Margulis [23] was a motivator to his work on Ramanujan graphs [24]. Additionally, the constructions of [21] and [27] produce graphs that are both Ramanujan and have girth $(1 - o_n(1))\frac{4}{3}\log_{d-1} n$, according to the previously stated restrictions on d.

More recently, Alon-Ganguly-Srivastava [3] showed that for a given d such that $d - 1$ is prime and $\alpha \in (0, 1/6)$, there is a construction of infinite families of graphs with girth at least $(1 - o_n(1))(2/3)\alpha \log_{d-1} n$ and λ at most $(3/\sqrt{2})\sqrt{d-1}$ with many eigenvalues localized on small sets of size $O(n^\alpha)$. Their motivation comes from the theory of quantum ergodicity in graphs, which relates high-girth expanding graphs to delocalized eigenvectors. See [3] for more on this. Our main result is based on some of the techniques of this work.

One other motivation to search for graphs with simultaneous good spectral expansion and high girth is its application to the theory of error-correcting codes, particularly for *Low Density Parity Check* or *LDPC* codes, originally introduced by Gallager [13]. The connection with high girth regular graphs was first pointed out by Margulis in [23]. The property of high-girth is desirable since the decoding of such codes relies on an iterative algorithm whose performance is worse in the presence of short cycles. Additionally, using graphs with good spectral properties to generate these codes heuristically seems to lead to good performance, as pointed out by several works [30, 18, 22].

1.1 Our results

We can now state our results and put them in perspective. Let's first introduce some useful definitions and notation.

▶ **Definition 6** (Bicycle-free at radius r). *A multigraph is said to be bicycle-free at radius r if the distance-r neighborhood of every vertex has at most one cycle.*

▶ **Definition 7** ((r, τ)-graph). *Let r and τ be a positive integers. Then, we call a graph G a (r, τ)-graph if it satisfies the following conditions:*

- *G is bicycle-free at radius at least r;*
- *The number of cycles of length at most r is at most τ.*

Our main result is the following short cycle removal theorem:

▶ **Theorem 8.** *There exists a deterministic polynomial-time algorithm* fix *that, given as input a d-regular n-vertex (r, τ)-graph G satisfying $r \leqslant (2/3)\log_{d-1}(n/\tau) - 5$ outputs a graph* $\mathrm{fix}(G)$ *satisfying*

- $\mathrm{fix}(G)$ *is a d-regular graph with* $n + O(\tau \cdot (d-1)^{r/2+1})$ *vertices;*
- $\lambda(\mathrm{fix}(G)) \leqslant \max\{\lambda(G), 2\sqrt{d-1}\} + O_d(1/r)$*;*
- $\mathrm{fix}(G)$ *has girth at least* r*.*

The key fact in our proof of this statement is a theorem proved by Kahale [15], originally used to construct Ramanujan graphs with better expansion of sublinear sized subsets. See also [3] and [2] for other applications of this technique. We will prove this theorem in Section 2.

The preconditions of this theorem are not arbitrary. Even though random uniformly n-vertex d-regular graphs have constant girth with high probability, they are bicycle-free at radius $\Omega(\log_{d-1} n)$ and the number of cycles of length at most $c \log_{d-1} n$ (for small enough c) is $o(n)$ with high probability. Recall that from Theorem 4 we also know that being near-Ramanujan is also a property that occurs with high probability in random regular graphs. So a statement like the above can be used to produce distributions over regular graphs that have high girth and are near-Ramanujan with high probability. With this in mind, we introduce the following definition:

▶ **Definition 9** ((Λ, g)-good graphs)**.** *We call a graph G a (Λ, g)-good graph if $\lambda(G) \leqslant \Lambda$ and girth(G) $\geqslant g$.*

Let $\mu_d(n)$ be a distribution over d-regular graphs with $\sim n$ vertices. We say $\mu_d(n)$ is (Λ, g)-good if $G \sim \mu_d(n)$ is (Λ, g)-good with probability at least $1 - o_n(1)$.

Additionally, we call the distribution explicit if sampling an element is doable in polynomial time.

We shall prove the following using Theorem 8 in Appendix A:

▶ **Theorem 10.** *Given $d \geqslant 3$ and n, let G be a uniformly random d-regular n-vertex graph. For any $c < 1/4$ and $\epsilon > 0$, $\mathrm{fix}(G)$ is a $(2\sqrt{d-1} + \epsilon, c \log_{d-1} n)$-good explicit distribution.*

Recall that the upper bound on the girth of a regular graph is $(1 + o_n(1))2 \log_{d-1} n$, so this distribution has optimal girth up to a constant. Based on our proof of the above and using some classic results about the number of d-regular n-vertex graphs, we can show a lower bound on the number of $(2\sqrt{d-1} + \epsilon,\ c \log_{d-1} n)$-good graphs in some range.

▶ **Corollary 11.** *Let $d \geqslant 3, n$ be integers and $\epsilon > 0, c > 1/4$ reals. The number of d-regular graphs with number of vertices in $[n, n + O(n^{3/8})]$, which are $(2\sqrt{d-1} + \epsilon,\ c \log_{d-1} n)$-good, is at least*

$$\Omega\left(\left(\frac{d^d n^d}{e^d (d!)^2}\right)^{n/2}\right).$$

We prove both of these results in Appendix A.

Finally, we show a slightly stronger version of result of [26] by plugging our short cycle removal theorem into their construction.

▶ **Theorem 12.** *Given any integer n and constants $d \geqslant 3$, $\epsilon > 0$ and c, there is a deterministic polynomial-time (in n) algorithm that constructs a d-regular N-vertex graph with the following properties:*

- $N = n(1 + o_n(1))$*;*
- $\lambda(G) \leqslant 2\sqrt{d-1} + \epsilon$*;*
- *G has girth at least* $c\sqrt{\log n}$*.*

Note that this only works for large enough n. Also, the running time from the theorem above has an exponential dependency on d, ϵ and c. The proof of this statement as well as the precise dependencies on these constants will be worked out in Appendix B.

1.2 Models of random regular graphs

We will introduce some classic models of random regular graphs, which we will use throughout the paper.

▶ **Definition 13** ($\mathcal{G}_d(n)$)**.** *Let $\mathcal{G}_d(n)$ denote the set of d-regular n-vertex graphs. We write $G \sim \mathcal{G}_d(n)$ to denote that G is sampled uniformly at random from $\mathcal{G}_d(n)$.*

Sampling from $\mathcal{G}_d(n)$ is not easy a priori; the standard way to do so is using the *configuration model*, which was originally defined by Bollobás [7].

▶ **Definition 14** (Configuration model)**.** *Given integers $n > d > 0$ with nd even, the configuration model produces a random n-vertex, d-regular undirected multigraph (with loops) G. This multigraph is induced by a uniformly random matching on the set of "half-edges", $[n] \times [d] \cong [nd]$ (where $(v, i) \in [n] \times [d]$ is thought of as half of the ith edge emanating from vertex v). Given a matching, the multigraph G is formed by "attaching" the matched half-edges.*

This model corresponds exactly to the uniform distribution on not necessarily simple d-regular n-vertex graphs. It also not hard to see that the conditional distribution of the d-regular n-vertex configuration model when conditioned on it being a simple graph is exactly the uniform distribution on $\mathcal{G}_d(n)$. The probability that the sampled graph is simple is $\Omega_d(1)$.

The configuration model has the advantage that is easy to sample and to analyze. For reference, the proof of Theorem 4 was done in terms of the configuration model and so the theorem also applies to it.

2 Short cycles removal

In this section we prove Theorem 8. Recall that we are given a d-regular n-vertex (r, τ)-graph G with the constraint specified in Theorem 8 and we wish to find some d-regular graph $\mathrm{fix}(G)$ on $\sim n$ vertices such that $\lambda(\mathrm{fix}(G)) \leqslant \lambda(G) + o_r(1)$ and its girth is at least r.

Briefly, the algorithm that achieves this works by removing one edge per small cycle from G, effectively breaking apart all such cycles, and then fixing the resulting off degree vertices by adding d-ary trees in a certain way. We will now more carefully outline this method and then proceed to fill in some details as well as show it works as desired.

Before starting, we introduce some notation which will be helpful.

▶ **Definition 15** ($\mathrm{Cyc}_g(G)$)**.** *Given a graph G, let $\mathrm{Cyc}_g(G)$ denote the collection of all cycles in G of length at most g. Recall that if $\mathrm{Cyc}_g(G)$ is empty then G is said to have girth exceeding g.*

▶ **Definition 16** ($B_\delta(S)$)**.** *Given a set of vertices S in a graph G, let $B_\delta(S)$ denote the collection of vertices in G within distance δ of S. We will occasionally abuse this notation and write $B_\delta(v)$ instead of $B_\delta(\{v\})$ for a vertex v.*

Let E_c be a set containing exactly one arbitrary edge per cycle in $\mathrm{Cyc}_r(G)$. Note that the bicycle-freeness property implies E_c is a matching. Let H_t be a graph with the same vertex set as G obtained by removing all edges in E_c from G. To prevent ambiguity, whenever we pick something arbitrarily let's suppose the algorithm fix uses the lexicographical order of node labels as a tiebreaker. We also partition the endpoints of each edge as described in the following definition:

▶ **Definition 17** ($V_i(E)$)**.** *Given a matching E, we let $V_1(E)$ and $V_2(E)$ be two disjoint sets of vertices constructed as follows: for all $e = (u, v) \in E$ place u in $V_1(E)$ and v in $V_2(E)$ (so each endpoint is in exactly one of the two sets).*

Note that according to the above definition we have $|V_1(E_c)| = |V_2(E_c)| = |E_c| \leqslant \tau$. For ease of notation we also define:

▶ **Definition 18** ($\phi_E(v)$)**.** *Given a matching E and $(u, v) \in E$ such that $u \in V_1(E)$ and $v \in V_2(E)$, we denote by ϕ_E the function that maps endpoints to endpoints, so we have $\phi_E(u) = v$ and $\phi_E(v) = u$.*

We will often abuse notation and drop the E from ϕ_E when it is clear from context.

Since we break apart each cycle in $\mathrm{Cyc}_r(G)$, we can conclude that H_t has girth greater than r. However, note that in removing edges, H_t is no longer d-regular.

To fix this, consider the following object which we refer to as a d-regular tree of height h: a finite rooted tree of height h where the root has d children but all other non-leaf vertices have $d - 1$ children. This definition implies that every non-leaf vertex in a d-regular tree has degree d.

We shall add two d-regular trees to H_t in order to fix the off degrees, while maintaining the desired girth and bound on λ. The idea of using d-regular trees is based on the degree-correction gadget used in [3] for their construction of high-girth near-Ramanujan graphs with localized eigenvectors. As such, we will use some of the tools used in their proofs.

Let h be an integer parameter we shall fix later. Let T_1 and T_2 be two d-regular trees of height h and let L_1 and L_2 be the sets of leaves of each one. Note that $|L_1| = |L_2| = d(d-1)^{h-1} \approx (d-1)^h$. We shall add the two trees to H_t and then pair up elements of $V_1(E_c)$ with elements of L_1 (and analogously for $V_2(E_c)$ and L_2) and merge the paired up vertices. However, we have to deal with two potential issues:

- $|L_i| \neq |V_i(E_c)|$, in which case we cannot get an exact pairing between these sets;
- This procedure might result in the creation of small cycles (potentially even cycles of length $O(1)$).

To expand on the latter point, we describe a potential problematic instance. Suppose we can somehow pick h such that $|L_i| = |V_i(E_c)|$ and then arbitrarily pair up their elements. Suppose there are two edges in E_C corresponding to two cycles of constant length and denote their endpoints by $v_1 \in V_1(E_C), v_2 \in V_2(E_C)$ and $u_1 \in V_1(E_C), u_2 \in V_2(E_C)$. If the distance in T_1 of v_1 and u_1 given by the pairing of $V_1(E_c)$ and L_1 is small (constant, for example) and the same applies to the distance in T_2 of v_2 and u_2, then there is a cycle of small length (constant, for example) in the graph resulting from adding the two trees to H_t.

To address this issue we remove some extra edges from G that are somehow "isolated" and group them with edges from E_C. The goal is to have the endpoints of any two edges in E_C be far apart in T_1 and T_2 distance, but close to some of the endpoints of the extra edges. With this in mind, we set $h = \lceil \log_{d-1} \tau \rceil + \lceil r/2 \rceil + 1$ so that $|L_i| \approx \tau \cdot (d-1)^{r/2+1}$, which is close to the number of extra edges we want to remove. This choice will also be helpful later when we analyze the spectral properties of the construction.

Formally, this leads us to the following proposition:

▶ **Proposition 19.** *There is a set of edges E_t of G such that the following is true for $i \in \{1, 2\}$:*

- *$|V_i(E_t) \cup V_i(E_c)| = d(d-1)^{h-1}$;*
- *for all distinct $u, v \in V_i(E_t) \cup V_i(E_c)$, we have $v \notin B_r(u)$ and $u \notin B_r(v)$.*

Additionally, we can find such a set in polynomial time.

Proof. We will describe the efficient algorithm that does this.

We are going to incrementally grow our set E_t, one edge at the time, until $|V_i(E_t) \cup V_i(E_c)| = d(d-1)^{h-1}$, so suppose E_t is initially an empty set. We start by, for all $e = (u, v) \in E_c$, marking all vertices in $B_{1+r}(\{v, u\})$. Note that we marked at most $\tau \cdot (d(d-1)^r) \leqslant 2\tau(d-1)^{r+1}$ vertices.

Notice that, since we marked all vertices at distance $1 + r$ from any vertex in $V_i(E_c)$, we can safely pick any unmarked vertex and an arbitrary neighbor and add that edge to E_t.

We can now describe a procedure to add a single edge to E_t:

- Pick an unmarked vertex u and an arbitrary neighbor v of u;
- Add (u, v) to E_t;
- Mark all vertices in $B_{1+r}(\{u, v\})$.

By the same reasoning as before, as long as we have an unmarked vertex, this procedure works. If we repeat the above t times, we are left with at least $n - 2\tau(d-1)^{r+1} - 2t(d-1)^{r+1}$ unmarked vertices. We claim the procedure can be successfully repeated at least $2\tau(d-1)^{r/2+2}$ times. In such a case, the number of unmarked vertices left is at least:

$$n - 2\tau(d-1)^{r+1} - 4\tau(d-1)^{r/2+2}(d-1)^{r+1} \geqslant n - 6\tau(d-1)^{3r/2+3},$$

which is always greater than 0 when $r \leqslant \frac{2}{3}\log_{d-1}(n/\tau) - 5$. Hence, we always have at least one unmarked vertex to pick throughout the procedure.

Note that the number of repetitions we require exactly matches the size of $|E_t|$ so we need this to be exactly $d(d-1)^{h-1} - \tau \leqslant 2\tau(d-1)^{r/2+2}$, which means our algorithm always succeeds. ◀

We will state some simple properties of this construction that will be relevant later on.

▶ **Fact 20.** $|V_i(E_t)| \geqslant \tau \cdot (d-1)^{\lceil r/2 \rceil}$

Proof. We simply have: $|V_i(E_t)| = |E_t| = d(d-1)^{h-1} - \tau \geqslant \tau \cdot (d-1)^{\lceil r/2 \rceil}$. ◀

▶ **Fact 21.** *For all $e \in E_t$, there is at most one cycle in $B_r(e)$ in G and if there is a cycle it has length greater than r.*

Proof. That there is at most one cycle in $B_r(e)$ is obvious since G is bicycle-free at radius r. So, let's suppose there is a cycle C in $B_r(e)$ with length less than or equal to r. Then, there is at least one edge $e' \in C$ that is also in E_c, but in that case $e' \in B_r(e)$, which contradicts the definition of E_t. ◀

We can now extend our definition of H_t. Let H be the graph obtained from G by removing all edges in E_c and in E_t.

Recall our plan to add T_1 and T_2, two d-regular trees of height h (recall $h = \lceil \log_{d-1} \tau \rceil + \lceil r/2 \rceil + 1$), to H while pairing up elements of L_i with endpoints of removed edges. We will now describe a pairing process that achieves high girth (and later we will see how it also achieves low λ).

First, consider a canonical ordering of L_1 and L_2 based on visit times from a breath-first search, as illustrated in Figure 1 for $d = 3$. Given this ordering, the following is easy to see:

▶ **Fact 22.** *The tree distance between two leaves with indices i and j is at least* $2(1 + \log_{d-1}((|i-j|+1)/d))$.

Proof. Let's show that the lowest common ancestor of the two leaves is at least $1+\log_{d-1}((|i-j|+1)/d)$, this proves the claim since we need to travel this distance twice, from the ith indexed leaf to the ancestor and then back to the jth indexed leaf. Let V_0 be the set of $|i-j|+1$ leaves with indices between i and j. Let's construct the smallest subtree that includes V_0 from bottom up and compute its height, which is an upper bound to the desired lowest common ancestor. First, group elements of V_0 in groups of at most $d-1$ consecutive indices and add one representative of each group to a set V_1. Each group corresponds to a node that parents all of its elements. There are at most $|V_0|/(d-1)$ such groups, so $|V_1| \leqslant |V_0|/(d-1)$. Repeat the same procedure until $|V_a| \leqslant 1$, in which case a is an upper bound to the height of the goal subtree, and by induction we have that $|V_{i+1}| \leqslant |V_i|/(d-1)$, so $a \geqslant \log_{d-1}|V_0|$.

This is not quite right because if the last grouping corresponds to the root of the tree, we need to group elements in d groups, because this is the degree of the root, so by accounting for this we have $a \geqslant 1+\log_{d-1}(|V_0|/d)$. ◀

Now, consider the following pairing of elements in L_1 and $V_1(E_t) \cup V_1(E_c)$: pick an arbitrary element of $V_1(E_c)$ and pair it up with the first leaf of L_1. Now pick $(d-1)^{\lceil r/2 \rceil}$ distinct elements of $V_1(E_t)$ and pair them up with the next leaves of L_1. Repeat this procedure, of pairing one element of $V_1(E_c)$ with $(d-1)^{\lceil r/2 \rceil}$ elements of $V_1(E_t)$ with a contiguous block of leaves until we exhaust all elements of $V_1(E_c)$. Note that by Fact 20, there always are enough elements in E_t to perform this pairing. Pair up any remaining leaves with the remaining elements of $V_1(E_t)$ arbitrarily. Now repeat the same procedure but for L_2 and $V_2(E_t) \cup V_2(E_c)$ with the same groupings (so the endpoints of an edge in either E_t or E_c are mapped to the same leaves of L_1 and L_2). This pairing procedure is pictured in Figure 2 below.

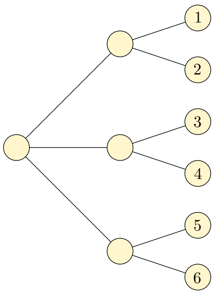


Figure 1 Leaf ordering for $d = 3$.

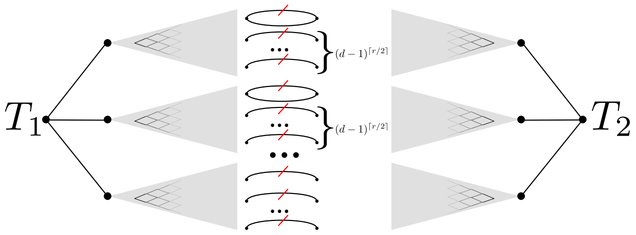


Figure 2 Example pairing.

Let $\text{fix}(G)$ be defined as the graph resulting from applying the method described in the previous paragraph to fix the degrees of H. It is now obvious that $\text{fix}(G)$ is a d-regular graph and we only add $|T_1|+|T_2| = O(\tau \cdot (d-1)^{r/2+1})$ new vertices, so it has $n+O(\tau \cdot (d-1)^{r/2+1})$ total vertices. We will now analyze the resulting girth and λ value and prove Theorem 8 in the process.

2.1 Analyzing the girth of $\text{fix}(G)$

Here we prove that the girth of $\text{fix}(G)$ is at least r. Let's start by supposing, for the sake of contradiction, that there is a cycle C of length less than r. We know that the girth of H is more than r by definition, so C has to use an edge from T_1 or T_2. Without loss of generality, let's assume that C contains at least one edge from T_1. Since T_1 is a tree, C has to eventually exit T_1 and use some edges from H, so in particular it uses some vertex $v \in L_1$. We will show that in this case C has length at least r, which is a contradiction. Thus, we have to handle two cases: $v \in V_1(E_c)$ and $v \in V_1(E_t)$.

Let us start with the $v \in V_1(E_c)$ case. Let's follow C starting in v and show that to loop back to v, C would require to traverse at least r edges. So, we start in v and go into T_1 by following the only edge in T_1 that connects to v. Then, the cycle C has to use some edges from T_1 and finally exit through some other vertex in L_1 before eventually looping back to v. Suppose that $u \in L_1$ is such a vertex. Due to our grouping of elements in E_t with $(d-1)^{\lceil r/2 \rceil}$ elements in E_c, if u is in $V_1(E_c)$, we know that the tree indices of v and u differ by at least $(d-1)^{\lceil r/2 \rceil}$. Hence, plugging this into the bound from Fact 22, the tree distance between v and u is at least $r-1$, which would imply C has length at least r. So u has to be in $V_1(E_t)$.

Continuing our traversal of C, we now exit T_1 through u and need to loop back to v. From our construction in Proposition 19 we know that the distance in H between v and u is at least r, so any short path in $\text{fix}(G)$ between these vertices has to go through T_1 or T_2. Again, our Proposition 19 construction gives that the distance in H between v and any other vertex in L_1 is at least r, so such a short path will have to use some edges in T_2.

Finally, we claim that the distance from u to any vertex w in L_2 is at least r. If $w \neq \phi(u)$, we know from our Proposition 19 construction that the distance between u and w is at least r. Otherwise, if there is a path P of length less than r from u to w, then the cycle $P + uw$ has length at most r and is in $B_r(\{u, w\})$, which contradicts Fact 21. In conclusion, it is not possible to loop back to v using less than r steps, which concludes the proof of the $v \in V_1(E_c)$ case.

The proof for the $v \in V_1(E_t)$ case is already embedded in the previous proof, so we will just sketch it. Using the same argument we start by following C into T_1 and eventually exiting through some vertex $u \in V_1(E_t)$. As we saw before, the H distance between u and v is at least r and the H distance between u and any other vertex in L_1 or any vertex in L_2 is at least r, so we cannot loop back to v from u, which concludes the proof of this case.

2.2 Bounding $\lambda(\text{fix}(G))$

We finally analyze the spectrum of $\text{fix}(G)$ by proving that $\lambda(\text{fix}(G)) \leqslant \lambda(G) + O_d(1/r)$. This argument is similar to the proof in Section 4 of [3], but adapted to our construction.

First, observe that the adjacency matrix of $\text{fix}(G)$, which we will denote by simply A, can be written in the following way: $A = A_G - A_{E_c} - A_{E_t} + A_{T_1} + A_{T_2}$, where A_G is the adjacency matrix of G defined on the vertex set of $\text{fix}(G)$ (which is to say G with a few isolated vertices from the added trees), A_{E_c} is the adjacency matrix of the cycle edges removed, and so on. Also, let V_G be the set of vertices from G, V_1 the set of vertices from T_1 and V_2 the set of vertices from T_2, so $V = V_G \cup V_1 \cup V_2$. In this section we will prove $\lambda(A) \leqslant \lambda(G) + O_d(1/r)$.

Let g be any unit eigenvector of A orthogonal to the all ones vector, so $\sum_{v \in V} g_v^2 = 1$ and $\sum_{v \in V} g_v = 0$. We have that $|\sum_{v \in V_1 \cup V_2} g_v| \leqslant \sqrt{2|T_i|}$ by Cauchy-Schwarz (since this vector is supported on only $2|T_i|$ entries), which in turn implies that $|\sum_{v \in V_G} g_v| \leqslant \sqrt{2|T_i|}$.

It suffices to show that $|g^T A g| \leqslant \lambda(G) + O_d(1/r)$. To do so, we shall analyze the contributions of A_G, A_{E_c}, A_{E_t}, A_{T_1} and A_{T_2} to $|g^T A g|$.

To bound the contribution of A_{T_1} and A_{T_2}, we use a lemma proved by Alon-Ganguly-Srivastava:

▶ **Lemma 23** ([3, Lemma. 4.1]). *Let W_i be the set of non-leaf vertices of T_i. Then for any vector f we have:*

$$|f^T A_{T_i} f| \leqslant 2\sqrt{d-1} \sum_{w \in W_i} f_w^2 + \sqrt{d-1} \sum_{v \in L_i} f_v^2.$$

Recall that the edges in $E_t \cup E_c$ define a perfect matching between L_1 and L_2, so we have the following:

$$|g^T(A_{E_c} + A_{E_t})g| = \left| \sum_{uv \in E_t \cup E_c} 2g_u g_v \right| \leqslant \sum_{v \in L_1 \cup L_2} g_v^2.$$

Finally, let g_G be the projection of g to the subspace spanned by V_G. Observe that $|g^T A_G g| = |g_G^T A_G g_G|$. Now, let $\mathbf{1}_G$ be the all ones vector supported on the set V_G and $g_\perp$ be a vector orthogonal to $\mathbf{1}_G$ such that $g_G = a\mathbf{1}_G + g_\perp$, for some constant a. We have that $\mathbf{1}_G^T g_G = a\mathbf{1}_G^T \mathbf{1}_G$, which implies

$$|a| = \left| \frac{\sum_{v \in V_G} (g_G)_v}{n} \right| \leqslant \frac{\sqrt{2|T_i|}}{n}.$$

Now observe:

$$|g_G^T A_G g_G| \leqslant |g_\perp^T A_G g_\perp| + |(a\mathbf{1}_G)^T A_G (a\mathbf{1}_G)| \leqslant \lambda(G) \sum_{v \in V_G} g_v^2 + \frac{2|T_i|d}{n}.$$

Note that $\sum_{v \in V_G} g_v^2 \leqslant 1$. We claim that the term $\frac{2|T_i|d}{n}$ is $O_d(1/r)$. We have $|T_i| = O(\tau \cdot (d-1)^{r/2+1})$ and we know from the problem constraints that $r \leqslant (2/3)\log_{d-1}(n/\tau) - 5$ which implies $\tau \cdot (d-1)^{r/2+1}/n \leqslant O((d-1)^{-r}) = O_d(1/r)$.

We can now plug everything together and apply Lemma 23 to obtain:

$$|g^T A g| \leqslant \lambda(G) + (\sqrt{d-1} + 1) \sum_{v \in L_1 \cup L_2} g_v^2 + O_d(1/r).$$

We will conclude our proof by showing that $\sum_{v \in L_1 \cup L_2} g_v^2$ is $O(1/r)$. It should be clear from the symmetry of our construction that we only need to prove $\sum_{v \in L_1} g_v^2 = O(1/r)$, since the same is analogous for L_2.

The following lemma can be proved using a known method by Kahale [15, Lemma 5.1]. This statement is similar to one found in [2, Lemma 3.2] and its proof is also very similar. For completeness, we present a self-contained proof of that based on the one from [2].

▶ **Lemma 24.** *Let v be some vertex of V. Let l be a positive integer such that $B_l(v)$ forms a tree. Let X_i be the set of all vertices at distance exactly i from v in $\mathrm{fix}(G)$, so $X_0 = \{v\}$. Let f be any non zero eigenvector with eigenvalue $|\mu| \geqslant 2\sqrt{d-1}$. Then, for $1 \leqslant i \leqslant l$:*

$$\sum_{u \in X_i} f^2(u) \geqslant \sum_{u \in X_{i-1}} f^2(u)$$

Proof. We will proceed by induction on i. First of all, let's establish the $i = 1$ case. Note that we have $\sum_{u \in X_1} f(u) = \mu f(v)$.

By Cauchy-Schwarz we get $d \cdot \sum_{u \in X_1} f^2(u) \geqslant \mu^2 f^2(v)$, and using the fact that $|\mu| \geqslant 2\sqrt{d-1}$ we obtain the desired:

$$\sum_{u \in X_1} f^2(u) \geqslant \frac{\mu^2}{d} f^2(v) \geqslant f^2(v).$$

Let's now assume that the statement is true for $i-1$ and prove that this implies it is true for i. Let u be some vertex in X_{i-1}. Recall that $B_l(v)$ is a tree and let u' be its parent in X_{i-2} and $w_1, \ldots w_{d-1}$ be its children in X_i. We have $f(u') + \sum_{i=1}^{d-1} f(w_i) = \mu f(u)$. Note that $f(u') = \sqrt{d-1} f(u')/\sqrt{d-1}$ and apply Cauchy-Schwarz to obtain:

$$\left(\frac{f^2(u')}{d-1} + \sum_{i=1}^{d-1} f^2(w_i) \right)(2d-2) \geqslant \mu^2 f^2(u),$$

which implies

$$\frac{f^2(u')}{d-1} + \sum_{i=1}^{d-1} f^2(w_i) \geqslant \frac{\mu^2}{2d-2} f^2(u) \geqslant 2f^2(u),$$

where the last inequality follows from the fact that $|\mu| \geqslant 2\sqrt{d-1}$.

We can finally sum the above for all $u \in X_{i-1}$, noting that from the fact that $B_l(v)$ is a tree we know that each element in X_{i-2} appears $d-1$ times (as the parent of $d-1$ vertices) and each element in X_i appears once:

$$\sum_{u \in X_{i-2}} f^2(u) + \sum_{u \in X_i} f^2(u) \geqslant 2 \sum_{u \in X_{i-1}} f^2(u).$$

We now apply the induction hypothesis and obtain the result:

$$\sum_{u \in X_i} f^2(u) \geqslant 2 \sum_{u \in X_{i-1}} f^2(u) - \sum_{u \in X_{i-2}} f^2(u) \geqslant \sum_{u \in X_{i-1}} f^2(u). \qquad ◀$$

Our plan is to pick the parameters l and v from Lemma 24 and use it to show that $\sum_{v \in L_1} g_v^2 = O(1/r)$. Let μ be the eigenvalue associated with g and suppose that $|\mu| > 2\sqrt{d-1}$, otherwise $|\mu| \leqslant \lambda(G)$, which would imply the result. Set v to be the root of T_1. We will show that if we pick $l = h + \lfloor r/2 \rfloor$, where $h = \lceil \log_{d-1} \tau \rceil + \lceil r/2 \rceil + 1$ is the height of T_1 and T_2, then $B_l(v)$ forms a tree.

Note that $B_h(v)$ is exactly T_1, so it obviously forms a tree. To observe what happens in $B_l(v) \setminus B_h(v)$, we first prove the following proposition, whose proof uses some of the ideas of Section 2.1:

▶ **Proposition 25.** *Let u be a vertex in L_1. Let $\mathcal{P}(u)$ be the set of non-empty paths that start in u and whose first step does not go into T_1. Then, the shortest path in $\mathcal{P}(u)$ that ends in any vertex in L_1 has length at least r.*

Proof. As in the previous girth proof, we have two cases, $u \in V_1(E_c)$ and $u \in V_1(E_t)$. The latter case is obvious from the proof in Section 2.1, since if $u \in V_1(E_t)$ then the H distance to any node in L_1 is at least r (from Proposition 19) and the H distance to any node in L_2 is also at least r (from Fact 21). So, suppose $u \in V_1(E_c)$.

Let's follow the same proof strategy as before, so let $P \in \mathcal{P}(u)$ be the shortest path and let's follow P starting in u. Again, from Proposition 19 the H distance of u to any node in L_1 is at least r. However, u might reach $\phi(u)$ in a short number of steps (namely, if the cycle corresponding to $(u, \phi(u))$ is short). So, let's follow P to $\phi(u)$ and into T_2. We are now in the exact same situation as in the setup of the proof in Section 2.1 (but starting in T_2), so the result follows. ◀

Let u be some vertex in L_1. Let's say a vertex w is at $\mathcal{P}$-distance δ from u if the shortest path $P \in \mathcal{P}(u)$ that ends in w has length δ. Additionally, let $S_\delta(u)$ be the set of vertices that are at a $\mathcal{P}$-distance of at most δ from u. From Proposition 25, we know that for all distinct $u, w \in L_1$, the sets $S_{\lfloor r/2 \rfloor}(u)$ and $S_{\lfloor r/2 \rfloor}(w)$ are disjoint. Thus, we have that for $u \in L_1$ the vertices in $S_{\lfloor r/2 \rfloor}(u)$ form disjoint trees rooted at u, which shows that $B_l(v)$ forms a tree.

We can now apply Lemma 24 and conclude that for all $1 \leqslant i \leqslant l$, we have $\sum_{u \in X_i} g_u^2 \geqslant \sum_{u \in X_{i-1}} g_u^2$. So the sequence $(\sum_{u \in X_i} g_u^2)_i$ is an increasing sequence. Note that $X_h = L_1$, so $\sum_{u \in X_h} g_u^2 = \sum_{u \in L_1} g_u^2$. Additionally, we know that the total sum of $(\sum_{u \in X_i} g_u^2)_i$ is at most one (since g is a unit vector and the X_i are disjoint), so we have that $\lfloor r/2 \rfloor \cdot \sum_{u \in X_h} g_u^2 \leqslant \sum_{i=h}^{l} \sum_{u \in X_i} g_u^2 \leqslant 1$ and finally $\sum_{u \in L_1} g_u^2 = \sum_{u \in X_h} g_u^2 \leqslant 1/\lfloor r/2 \rfloor = O(1/r)$.

This concludes the proof of Theorem 8.

3 Open problems

- Can we improve Theorem 12 to obtain high girth?
  Something like this could be proved by showing that when 2-lifting a graph with large enough girth, with sufficiently high probability the girth of the resulting graph increases. This would boost the girth of the graph generated by the first step of the construction of [26] during the repeated 2-lift step. However, it is unclear if this can be done. Alternatively, one could show that bicycle-freeness increases with good probability as we 2-lift, but this is also unclear.
  A different strategy would be to find a different way to derandomize Theorem 4 such that the we can generate a starter graph of larger size. However, it is unclear if this strategy could work since the tool used to derandomize this, namely (δ, k)-wise uniform permutations (defined in Appendix C), cannot be improved to derandomize this to the required extent.
- Can we obtain Theorem 10 for higher values of c; for example, can we build a distribution that is $(2\sqrt{d-1}+\epsilon,\ .99\log_{d-1} n)$-good?
  One promising strategy would be to show that the graphs produced by the distribution described in [20], which were shown to have girth at least $.99\log_{d-1} n$ with high probability, are also near-Ramanujan with high probability. Numerical calculations seem to indicate that the answer is positive, as pointed out in one of the open problems given in [20].

References

1 Noga Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

2 Noga Alon. Explicit expanders of every degree and size. *arXiv preprint*, 2020. arXiv:2003.11673.

3 Noga Alon, Shirshendu Ganguly, and Nikhil Srivastava. High-girth near-ramanujan graphs with localized eigenvectors. *arXiv preprint*, 2019. arXiv:1908.03694.

4 Noga Alon and Shachar Lovett. Almost k-wise vs. k-wise independent permutations, and uniformity for general group actions. *Theory of Computing*, 9:559–577, 2013.

5 Edward A. Bender and E. Rodney Canfield. The asymptotic number of labeled graphs with given degree sequences. *J. Combinatorial Theory Ser. A*, 24(3):296–307, 1978. doi:10.1016/0097-3165(78)90059-6.

6 Yonatan Bilu and Nathan Linial. Lifts, discrepancy and nearly optimal spectral gap. *Combinatorica*, 26(5):495–519, 2006. doi:10.1007/s00493-006-0029-7.

7 Béla Bollobás. A probabilistic proof of an asymptotic formula for the number of labelled regular graphs. *European J. Combin.*, 1(4):311–316, 1980. doi:10.1016/S0195-6698(80)80030-8.

8 Charles Bordenave. A new proof of Friedman's second eigenvalue theorem and its extension to random lifts. Technical Report 1502.04482v4, arXiv, 2019. To appear in Annales scientifiques de l'École normale supérieure. arXiv:1502.04482v4.

9 Xavier Dahan. Regular graphs of large girth and arbitrary degree. *Combinatorica*, 34(4):407–426, 2014. doi:10.1007/s00493-014-2897-6.

10 Paul Erdős and Horst Sachs. Reguläre graphen gegebener tailenweite mit minimaler knollenzahl. *Wiss. Z. Univ. Halle-Willenberg Math. Nat.*, 12:251–258, 1963.

11 Joel Friedman. Some geometric aspects of graphs and their eigenfunctions. *Duke Mathematical Journal*, 69(3):487–525, 1993.

12 Joel Friedman. A proof of Alon's second eigenvalue conjecture and related problems. *Memoirs of the American Mathematical Society*, 195(910):viii+100, 2008.

13 R. G. Gallager. Low-density parity-check codes. *IRE Trans.*, IT-8:21–28, 1962. doi:10.1109/tit.1962.1057683.

14 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *American Mathematical Society Bulletin*, 43(4):439–561, 2006.

15 Nabil Kahale. Eigenvalues and expansion of regular graphs. *Journal of the ACM (JACM)*, 42(5):1091–1106, 1995.

16 Eyal Kaplan, Moni Naor, and Omer Reingold. Derandomized constructions of k-wise (almost) independent permutations. *Algorithmica. An International Journal in Computer Science*, 55(1):113–133, 2009.

17 Martin Kassabov. Symmetric groups and expander graphs. *Inventiones Mathematicae*, 170(2):327–354, 2007.

18 John Lafferty and Dan Rockmore. Codes and iterative decoding on algebraic expander graphs. In *the Proceedings of ISITA*. Citeseer, 2000.

19 Felix Lazebnik and Vasiliy A. Ustimenko. Explicit construction of graphs with an arbitrary large girth and of large size. *Discrete Appl. Math.*, 60(1-3):275–284, 1995. ARIDAM VI and VII (New Brunswick, NJ, 1991/1992). doi:10.1016/0166-218X(94)00058-L.

20 Nati Linial and Michael Simkin. A randomized construction of high girth regular graphs. *arXiv preprint*, 2019. arXiv:1911.09640.

21 A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988. doi:10.1007/BF02126799.

22 Mohammad M Mansour and Naresh R Shanbhag. Construction of ldpc codes from ramanujan graphs. In *36th Annu. Conf. on Information Sciences and Systems*, 2002.

23 G. A. Margulis. Explicit constructions of graphs without short cycles and low density codes. *Combinatorica*, 2(1):71–78, 1982. doi:10.1007/BF02579283.

24 G. A. Margulis. Explicit group-theoretic constructions of combinatorial schemes and their applications in the construction of expanders and concentrators. *Problemy Peredachi Informatsii*, 24(1):51–60, 1988.

25 Brendan D. McKay, Nicholas C. Wormald, and Beata Wysocka. Short cycles in random regular graphs. *Electron. J. Combin.*, 11(1):Research Paper 66, 12, 2004. URL: http://www.combinatorics.org/Volume_11/Abstracts/v11i1r66.html.

26 Sidhanth Mohanty, Ryan O'Donnell, and Pedro Paredes. Explicit near-ramanujan graphs of every degree. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 510–523, 2020.

27 Moshe Morgenstern. Existence and explicit constructions of $q+1$ regular Ramanujan graphs for every prime power q. *J. Combin. Theory Ser. B*, 62(1):44–62, 1994. doi:10.1006/jctb.1994.1054.

28 A. Nilli. On the second eigenvalue of a graph. *Discrete Mathematics*, 91(2):207–210, 1991.

29 Mark S Pinsker. On the complexity of a concentrator. In *7th International Telegraffic Conference*, volume 4, pages 1–318. Citeseer, 1973.

30 Joachim Rosenthal and Pascal O Vontobel. Constructions of ldpc codes using ramanujan graphs and ideas from margulis. In *in Proc. of the 38-th Allerton Conference on Communication, Control, and Computing*. Citeseer, 2000.

A A near-Ramanujan graph distribution of girth $\Omega(\log_{d-1} N)$

Recall Theorem 4, which says that uniformly random d-regular graphs are near-Ramanujan. We will combine this result with our machinery of Section 2 to show Theorem 10, namely that there exists a distribution over graphs that is $(2\sqrt{d-1}+\epsilon, c\log_{d-1} n)$-good for any $\epsilon > 0$ and $c < 1/4$, which we will show is the distribution resulting from applying algorithm fix to a sample of $\mathcal{G}_d(n)$.

First, we note that $\mathcal{G}_d$ has nice bicycle-freeness. We quote the relevant result from [8], which we restate below:

▶ **Lemma 26** ([8, Lemma 9]). *Let $d \geqslant 3$ and r be positive integers. Then $G \sim \mathcal{G}_d(n)$ is bicycle-free at radius r with probability $1 - O((d-1)^{4r}/n)$.*

An obvious corollary of this is that for any constant $c < 1/4$, we have that $G \sim \mathcal{G}_d(n)$ is bicycle free at radius $c \log_{d-1} n$ with high probability.

To bound the number of short cycles in $\mathcal{G}_d(n)$ we use a classic result that very accurately estimates the number of short cycles in random regular graphs.

▶ **Lemma 27** ([25, Section 2]). *Let $G \sim \mathcal{G}_d(n)$ and X_i be the random variable that denotes the number of cycles of length i in G. Let $R_i = \max\{(d-1)^i/i, \log n\}$. Then*

$$\boldsymbol{Pr}\left[X_i \leqslant R_i, \text{ for all } 3 \leqslant i \leqslant 1/4 \log_{d-1} n\right] = 1 - o_n(1).$$

Given the above, we obtain the following bound, for all $c < 1/4$:

$$\sum_{i=1}^{c \log_{d-1} n} \max\{(d-1)^i/i, \log n\} = O(n^c).$$

So we obtain the following proposition:

▶ **Proposition 28.** *For any $c < 1/4$ and any $\epsilon > 0$, $G \sim \mathcal{G}_d(n)$ is a $(c \log_{d-1} n, O(n^c))$-graph and satisfies $\lambda(G) \leqslant 2\sqrt{d-1} + \epsilon$ with probability $1 - o_n(1)$.*

Finally, we want to apply Theorem 8, so first we need to verify its preconditions. For all $c < 1/4$ we have that $(2/3) \log_{d-1}(n/n^c) = (2/3)(1-c) \log_{d-1} n \geqslant c \log_{d-1} n$. Also note that $n^c(d-1)^{c/2 \log_{d-1} n + 1} = n^{3c/2} = O(n^{3/8})$, so when applying Theorem 8 the resulting graph has $n + O(n^{3/8}) = n(1 + o_n(1))$ vertices. Thus, we obtain Theorem 10.

▶ Remark 29. Recall that $\mathcal{G}_d(n)$ is the same as the conditional distribution of the d-regular n-vertex configuration model when conditioned on it being a simple graph. Indeed, a graph drawn from the d-regular n-vertex configuration model is simple with probability $\Omega_d(1)$. A result very similar to Lemma 27 also holds for the configuration model and thus the results of this section also hold for the configuration model.

A.1 Counting near-Ramanujan graphs with high girth

We will briefly prove Corollary 11 using the result we just proved. For simplicity, we are going to work with the configuration model, using the observation of Remark 29.

Our proof will use a classic result on the number of not necessarily simple d-regular n-vertex graphs, which is the same as the number of graphs in the n-vertex d-regular configuration model. It is easy to show [5] that for nd even, the number of such graphs is

$$\sim \left(\left(\frac{d^d n^d}{e^d (d!)^2} \right)^{n/2} \right).$$

Hence, the core claim we need to prove, is the following:

▶ **Proposition 30.** *Let G_1 and G_2 be distinct graphs that follow the preconditions of Theorem 8. Then $\operatorname{fix}(G_1)$ and $\operatorname{fix}(G_2)$ are also distinct.*

This proposition implies that given any two good d-regular n-vertex graphs, applying fix produces two distinct graphs. From our proof of Theorem 10 we also know that the result of applying fix adds at most $O(n^{3/8})$ vertices. Finally, since a $(1 - o_n(1))$ fraction of the graphs are good an thus when we apply fix they result in $(2\sqrt{d-1} + \epsilon,\, c \log_{d-1} n)$-good graphs, the result follows. For briefness, we will not give a detailed proof but only a sketch of the proof.

Proof sketch of Proposition 30. Recall the H graph from the description of fix and let H_1 be such graph corresponding to G_1 and define H_2 analogously. If H_1 and H_2 are distinct, then $\mathrm{fix}(G_1)$ and $\mathrm{fix}(G_2)$ are also distinct. This follows from the fact that the vertices of the two added trees will have to be matched up in an isomorphism between $\mathrm{fix}(G_1)$ and $\mathrm{fix}(G_2)$.

We claim that if G_1 and G_2 are distinct, then H_1 and H_2 are distinct. Let S_i be the set of vertices that were endpoints of edges removed from cycles in G_1 and G_2, respectively. Note that there are at least two such vertices in S_i and also we cannot remove multiple edges adjacent to one vertex since this would imply the existence of two cycles in a small neighborhood, breaking the bicycle-freeness assumption. We can ignore the other removed edges since the local neighborhoods of edges removed from cycles are necessarily distinct from the local neighborhoods of the other removed edges. Now, the edges removed from G_i form a perfect matching on S_i that adds exactly $|S_i|/2$ cycles to H_i. Also, there is exactly one perfect matching that adds $|S_i|/2$ cycles to H_i to recover G_i. That means that there is only one G_i that could have generated H_i, which implies the claim. ◀

B Explicit near-Ramanujan graphs of girth $\Omega(\sqrt{\log n})$

In this section we prove Theorem 12, building on the construction in the proof of Theorem 5. We note that the original construction has no guarantees on the girth of the constructed graph other than a constant girth. We will briefly recap the main tools and ideas from the paper.

B.1 Review of constructing explicit near-Ramanujan graphs

Given a d-regular n-vertex graph $G = (V, E)$, let $w \in \{\pm 1\}^E$ be an edge-signing of G. The *2-lift of G given w* is defined as the following d-regular $2n$-vertex graph $G_2 = (V_2, E_2)$:

$$V_2 = V \times \{\pm 1\} \qquad E_2 = \{\{(u,\sigma),(v,\sigma \cdot w(u,v))\} : (u,v) \in E, \sigma \in \{\pm 1\}\}.$$

It was observed in [6] that the spectrum of G_2 is given by the union of the spectra of G and $\widetilde{G}_w$, where the latter refers to the eigenvalues of the adjacency matrix of G signed according to w, where each nonzero entry is $w(u,v)$ for $\{u,v\} \in E$.

This connection between the spectrum of an edge-signing of a graph and a 2-lift gave rise to the following theorem, which was proved in [26]. Below we write $\rho(G) = \max\{|\lambda_i| : i \in [n]\}$ for the *spectral radius* of G.

▶ **Theorem 31** ([26, Theorem 3.1]). *Let $G = (V, E)$ be an arbitrary d-regular n-vertex graph ($d \geqslant 3$). Assume G is bicycle-free at radius $r \gg (\log\log n)^2$. Then for a uniformly random edge-signing w, except with probability at most n^{-100} we have:*

$$\rho(\widetilde{G}_w) \leqslant 2\sqrt{d-1} \cdot \left(1 + \frac{(\log\log n)^4}{r^2}\right).$$

Furthermore, this can be derandomized: given a constant C there is a generator $h : \{0,1\}^s \to \{\pm 1\}^E$ computable in time $\mathrm{poly}(N^{C \log d})$, with seed length $s = O(\log(2C) + \log\log n + C \cdot \log(d) \cdot \log(n))$, such that for $u \in \{0,1\}^s$ chosen uniformly at random, with probability at most n^{-100} we have:

$$\rho(\widetilde{G}_{h(u)}) \leqslant 2\sqrt{d-1} \cdot \left(1 + \frac{(\log\log n)^4}{r^2}\right) + \frac{\sqrt{d}}{C^2}.$$

This theorem is a powerful tool that, combined with the above observation, allows one to double the number of vertices in a near-Ramanujan graph while keeping it near-Ramanujan, as long as the bicycle-freeness is good enough. It is easy to show that if G is bicycle-free at radius r, then *any* 2-lift of G is also bicycle-free at radius r. So, the strategy employed by [26] is to start with a graph with a smaller number of vertices that is bicycle-free at a big enough radius and 2-lift it enough times until the graph has the required number of vertices.

To generate this starting graph, the authors first showed how out to weakly derandomize [8]. Formally, the following is proved:

▶ **Theorem 32** ([26, Theorem 4.8]). *For a large enough universal constant α and any integer $n > 0$, given d, ϵ and c such that:*

$$3 \leqslant d \leqslant \alpha^{-1}\sqrt{\log n},\ \alpha^3 \cdot \left(\frac{\log\log n}{\log_{d-1} n}\right)^2 \leqslant \epsilon \leqslant 1,\ c < 1/4.$$

Let G be chosen from the d-regular n-vertex uniform configuration model. Then, except with probability at most $n^{-.99}$, the following hold:

- *G is bicycle-free at radius $c\log_{d-1} n$;*
- *$\lambda(G) \leqslant 2\sqrt{d-1}\cdot(1+\epsilon)$;*

Furthermore, this can be derandomized: there is a generator $h : \{0,1\}^s \to \mathcal{G}_d(n)$, with seed length $s = O(\log^2(n)/\sqrt{\epsilon})$ computable in time $\mathrm{poly}(n^{\log(n)/\sqrt{\epsilon}})$*, such that for $u \in \{0,1\}^s$ chosen uniformly at random, with probability at most $n^{-.99}$ we have that the above statements remain true for $G = h(u)$.*

Using these two theorems we can setup the construction of [26]. So, first assume we are given $n, d \geqslant 3$ and $\epsilon > 0$ and we wish to construct a d-regular graph G with n vertices with $\lambda(G) \leqslant 2\sqrt{d-1} + \epsilon$. The construction is now the following:

1. Use Theorem 32 to construct a d-regular graph G_0 with a small number of vertices $n_0 = n_0(n)$. If we pick n_0 to be $2^{O(\sqrt{\log n})}$ then the generator seed length is $O(\log(n)/\sqrt{\epsilon})$ and is computable in time $\mathrm{poly}(n^{1/\sqrt{\epsilon}})$, so we can enumerate over all possible seeds and find at least one that produces a graph that is bicycle-free at radius $\Omega(\log(n_0)) = \Omega(\sqrt{\log n}) \gg (\log\log n)^2$ and has $\lambda(G_0) \leqslant 2\sqrt{d-1}\cdot(1+\epsilon)$ in $\mathrm{poly}(n)$ time.
2. Next, we can repeatedly apply Theorem 31 to double the number of vertices of G_0, by choosing C to be $\sim d^{1/4}/\sqrt{\epsilon}$. We then enumerate over all seeds until we find one that produces a good graph, which only requires $\mathrm{poly}(n)$ time. On each application the bicycle-freeness radius is maintained (so we can keep applying Theorem 31) and the number of vertices of doubles. After roughly $\log(n/n_0)$ applications, the resulting graph has $n(1 + o_n(1))$ vertices and $\lambda(G) \leqslant 2\sqrt{d-1}\cdot(1+\epsilon)$.

B.2 Improving the girth of the construction

We are finally ready to prove Theorem 12. We are going to apply a similar strategy as the one from Appendix A. Instead of derandomizing Lemma 27 we are going to obtain a simpler bound, which is good enough to obtain the desired. We note however, that Lemma 27 can be derandomized and for completeness we show how to in Appendix C.

We start by proving the following lemma:

▶ **Lemma 33.** *Let G be a d-regular n-vertex graph with $\lambda(G) \geqslant 2\sqrt{d-1}$ and such that G is bicycle-free at radius $\alpha\log_{d-1} n$, for $\alpha \leqslant 2$. Then we can apply* fix *to G and obtain a graph such that:*

- $\mathrm{fix}(G)$ *is d-regular and has* $n(1+o_n(1))$ *vertices;*
- $\lambda(\mathrm{fix}(G)) \leqslant \lambda(G) + o_n(1)$*;*
- $\mathrm{fix}(G)$ *has girth* $(\alpha/3)\log_{d-1} n$*.*

Before proving this lemma, we prove a core proposition in a slightly more generic way.

▶ **Proposition 34.** *Let G be a d-regular graph that is bicycle-free at radius $2r$, then*

$$|\mathrm{Cyc}_r(G)| \leqslant n/(d-1)^r.$$

Proof. Pick one vertex per cycle in $\mathrm{Cyc}_r(G)$ and place it in a set S. We claim that for every distinct $u, v \in S$, $B_r(u) \cap B_r(v) = \emptyset$. Suppose this wasn't the case and suppose there is some w such that $w \in B_r(u) \cap B_r(v)$, for some pair u, v. Note that $B_{2r}(w)$ includes the two length r cycles that correspond to u and v, which contradicts bicycle-freeness in G.

Given the above, we have that the sets $B_r(u)$ for $u \in S$ are pairwise disjoint and also we know that $|B_r(u)| = d(d-1)^{r-1}$. Hence we have:

$$|\mathrm{Cyc}_r(G)| \cdot d(d-1)^{r-1} \leqslant n,$$

which implies the desired result. ◀

And we can prove the above lemma.

Proof of Lemma 33. By plugging G into Proposition 34 we can conclude that G is a $(\alpha \log_{d-1} n, n^{1-\alpha/2})$-graph. We wish to apply Theorem 8 so first recall its preconditions. By definition $\lambda(G) \geqslant 2\sqrt{d-1}$. However, the precondition on the radius of bicycle-freeness does not hold, since $(2/3)\log_{d-1}(n/n^{1-\alpha/2}) = (\alpha/3)\log_{d-1} n$ which is less than $\alpha \log_{d-1} n$. If we instead use the fact that G is also trivially a $((\alpha/3)\log_{d-1} n, n^{1-\alpha/2})$-graph, then the precondition is satisfied.

Thus, we can apply Theorem 8 and we obtain that $\mathrm{fix}(G)$ satisfies all the required conditions, which concludes the proof. ◀

Given this lemma, we will modify the first step of the construction of [26] to produce a graph G_0 with girth $c\sqrt{\log n}$. Note that, similarly to bicycle-freeness, the girth of a graph can only increase when applying any 2-lift, so this strategy guarantees that after step 2 of the construction, the final graph has the desired girth, which would imply Theorem 12.

First, when enumerating over all seeds to generate G_0 in step 1, we look for one that guarantees that G_0 is bicycle-free at radius $(1/5)\log_{d-1} n_0$ (recall that by Theorem 32 a $1 - o_n(1)$ fraction of the seeds satisfy this). Next, we apply Lemma 33 and obtain $\mathrm{fix}(G_0)$ with girth $(1/15)\log_{d-1} n_0$ and the desired value of $\lambda(G_0)$. Let $\kappa = 15c/\log_{d-1} 2$. We can set n_0 to $2^{\kappa\sqrt{\log n}}$, in which case G_0 has girth $c\sqrt{\log n}$.

Note that the above only works as long as $\kappa \leqslant \sqrt{\log n}$, otherwise $n_0 > n$. Also, from Theorem 31 and Theorem 32, we need $d \leqslant (\log n)^{1/8}/C$ and $\epsilon \gg \sqrt{d}(\log\log n)^4/(\log n)$ (the details on how to obtain these can be found on [26]).

Finally, we can precisely determine the running time of this algorithm. From Theorem 32, constructing G_0 takes time $\mathrm{poly}(n_0^{\log(n_0)/\sqrt{\epsilon}}) = \mathrm{poly}(n^{\log(c/\log_{d-1}(2))/\sqrt{\epsilon}})$ and using Theorem 31 with the appropriate choice of C takes time $\mathrm{poly}(n^{d^{1/4}\log(d)/\sqrt{\epsilon}})$.

C Derandomizing the number of short cycles

To make the statement of this section more precise, we will first define a known derandomization tool.

▶ **Definition 35** ((δ, k)-wise uniform permutations)**.** *Let $\delta \in [0,1]$ and $k \in \mathbb{N}^+$. Let $[n]_k$ denote the set of all sequences of k distinct indices from $[n]$. A random permutation $\pi \in S_n$ is said to be (δ, k)-wise uniform if, for every sequence $(i_1, \dots, i_k) \in [n]_k$, the distribution of $(\pi(i_1), \dots, \pi(i_k))$ is δ-close in total variation distance from the uniform distribution on $[n]_k$. When $\delta = 0$, we simply say that the permutation is (truly) k-wise uniform.*

Kassabov [17] and Kaplan–Naor–Reingold [16] independently obtained a deterministic construction of (δ, k)-wise uniform permutations with seed length $O(k \log n + \log(1/\delta))$.

▶ **Theorem 36** ([16, 17])**.** *There is a deterministic algorithm that, given δ, k, and n, runs in time $\operatorname{poly}(n^k/\delta)$ and outputs a multiset $\Pi \subseteq S_n$ (closed under inverses) of cardinality $S = \operatorname{poly}(n^k/\delta)$ (a power of 2) such that, for $\pi \sim \Pi$ chosen uniformly at random, π is a (δ, k)-wise uniform permutation.*

This theorem is required to obtain the generator mentioned in Theorem 32 and is the reason why (δ, k)-wise uniform permutations are useful tools to apply here. We will also need a convenient theorem of Alon and Lovett [4]:

▶ **Theorem 37** ([4])**.** *Let $\boldsymbol{\pi} \in S_n$ be a (δ, k)-wise uniform permutation. Then one can define a (truly) k-wise uniform permutation $\boldsymbol{\pi}' \in S_n$ such that the total variation distance between $\boldsymbol{\pi}$ and $\boldsymbol{\pi}'$ is $O(\delta n^{4k})$.*

We can now define a "derandomized" version of the configuration model, using this tool.

▶ **Definition 38.** *Recall how the configuration model is defined by a perfect matching of a set $[nd]$ of "half-edges".*

Let's denote this matching by M and define a way to generate it using random permutations. First a uniformly random permutation $\pi \in S_{nd}$ is chosen; then we set $M_{\pi(j),\pi(j+1)} = M_{\pi(j+1),\pi(j)} = 1$ for each odd $j \in [nd]$.

We can write the adjacency matrix A of G as the sum, over all $i, i' \in [d]$, of $M_{(v,i),(v',i')}$. Hence

$$\boldsymbol{A}_{v,v'} = \sum_{i,i'=1}^{d} \sum_{\substack{odd \\ j\in[nd]}} (1[\pi(j) = (v,i)] \cdot 1[\pi(j+1) = (v', i')] + 1[\pi(j) = (v', i')] \cdot 1[\pi(j+1) = (v,i)]).$$

The d-regular n-vertex (δ, k)-wise uniform configuration model is defined by using (δ, k)-wise uniform permutations instead. Similarly, we define the d-regular n-vertex k-wise uniform configuration model.

We can now describe the proposition we wish to prove.

▶ **Proposition 39.** *Fix $d \geqslant 3$, n and $k \geqslant c \log_{d-1} n$, where $c < 1/4$. Let G be drawn from the d-regular n-vertex $4k$-wise configuration model and X_i be the random variable that denotes the number of cycles of length i in G. Let $R_i = \max\{(d-1)^i/i, \log n\}$. Then*

$$\boldsymbol{Pr}\left[X_i \leqslant R_i,\ \textit{for all } 1 \leqslant i \leqslant 1/4 \log_{d-1} n\right] = 1 - o_n(1).$$

By Theorem 37, these statements remain true in the $(\delta, 4k)$-wise uniform versions of the model, $\delta \leqslant 1/n^{16k+1}$.

Proof. The proof follows almost directly from the proof of Lemma 27. First, note that X_i can be written as a polynomial of degree at most i in the entries of G's adjacency matrix, by summing over the products of the edge indicators of all possible cycles of length i in G. Thus, from our formula in Definition 38, it can be written as a polynomial of degree at most $2k$ in the permutation indicators $1[\pi(j) = (v, i)]$. So we can compute $\mathbf{E}[X_i]$ assuming that X_i is drawn from the fully uniform configuration model. Similarly, X_i^2 can be written as a polynomial of degree at most $4k$ in the permutation indicators, so we can compute $\mathbf{Var}[X_i]$ assuming that X_i is drawn from the fully uniform configuration model.

From [25] we have the following estimates, that only apply when $(d-1)^{2i-1} = o(n)$:

$$\mathbf{E}[X_i] = \frac{(d-1)^i}{2i}(1 + O(i(i+d)/n)) \qquad \mathbf{Var}[X_i] = \mathbf{E}[X_i] + O(i(i+d)/n)\mathbf{E}[X_i]^2 .$$

By applying Chebyshev's inequality to each X_i, just like in [25], we get the desired result. ◀

We can finally rewrite Theorem 32 in the language of the d-regular n-vertex (δ, k)-wise uniform configuration model and tack on the result we just proved.

▶ **Theorem 40.** *For a large enough universal constant α and any integer $n > 0$, fix $3 \leqslant d \leqslant \alpha^{-1}\sqrt{\log n}$ and $c < 1/4$, and let $\varepsilon \leqslant 1$ and k satisfy*

$$\varepsilon \geqslant \alpha^3 \cdot \left(\frac{\log\log n}{\log_{d-1} n}\right)^2, \qquad k \geqslant \alpha \log(n)/\sqrt{\varepsilon}.$$

Let G be chosen from the d-regular n-vertex k-wise uniform configuration model. Then except with probability at most $1/n^{.99}$, the following hold:

- *G is bicycle-free at radius $c \log_{d-1} n$;*
- *The total number of cycles of length at most $c \log_{d-1} n$ is $O(n^c)$;*
- *$\lambda(G) \leqslant 2\sqrt{d-1} \cdot (1 + \varepsilon)$.*

Finally, by Theorem 37, these statements remains true in the (δ, k)-wise uniform configuration model, $\delta \leqslant 1/n^{16k+1}$.

Fine-Grained Complexity of the List Homomorphism Problem: Feedback Vertex Set and Cutwidth

Marta Piecyk
Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland

Paweł Rzążewski
Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland
Institute of Informatics, University of Warsaw, Poland

Abstract

For graphs G, H, a homomorphism from G to H is an edge-preserving mapping from $V(G)$ to $V(H)$. In the list homomorphism problem, denoted by $\textsc{LHom}(H)$, we are given a graph G, whose every vertex v is equipped with a list $L(v) \subseteq V(H)$, and we need to determine whether there exists a homomorphism from G to H which additionally respects the lists L. List homomorphisms are a natural generalization of (list) colorings.

Very recently Okrasa, Piecyk, and Rzążewski [ESA 2020] studied the fine-grained complexity of the problem, parameterized by the treewidth of the instance graph G. They defined a new invariant $i^*(H)$, and proved that for every relevant graph H, i.e., such that $\textsc{LHom}(H)$ is NP-hard, this invariant is the correct base of the exponent in the running time of any algorithm solving the $\textsc{LHom}(H)$ problem.

In this paper we continue this direction and study the complexity of the problem under different parameterizations. As the first result, we show that $i^*(H)$ is also the right complexity base if the parameter is the size of a minimum feedback vertex set of G, denoted by $\text{fvs}(G)$. In particular, for every relevant graph H, the $\textsc{LHom}(H)$ problem

- can be solved in time $i^*(H)^{\text{fvs}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$, if a minimum feedback vertex set of G is given,
- cannot be solved in time $(i^*(H) - \varepsilon)^{\text{fvs}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$, for any $\varepsilon > 0$, unless the SETH fails.

Then we turn our attention to a parameterization by the cutwidth $\text{ctw}(G)$ of G. Jansen and Nederlof [TCS 2019] showed that $\textsc{List } k\textsc{-Coloring}$ (i.e., $\textsc{LHom}(K_k)$) can be solved in time $c^{\text{ctw}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$ for an absolute constant c, i.e., the base of the exponential function does not depend on the number of colors. Jansen asked whether this behavior extends to graph homomorphisms. As the main result of the paper, we answer the question in the negative. We define a new graph invariant $mim^*(H)$, closely related to the size of a maximum induced matching in H, and prove that for all relevant graphs H, the $\textsc{LHom}(H)$ problem cannot be solved in time $(mim^*(H) - \varepsilon)^{\text{ctw}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$ for any $\varepsilon > 0$, unless the SETH fails. In particular, this implies that, assuming the SETH, there is no constant c, such that for every odd cycle the non-list version of the problem can be solved in time $c^{\text{ctw}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$.

2012 ACM Subject Classification Mathematics of computing → Graph coloring; Theory of computation → Problems, reductions and completeness; Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases list homomorphisms, fine-grained complexity, SETH, feedback vertex set, cutwidth

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.56

Related Version *Full Version*: `https://arxiv.org/abs/2009.11642` [35]

Funding Supported by Polish National Science Centre grant no. 2018/31/D/ST6/00062.

Acknowledgements We are grateful to Bart M. P. Jansen for introducing us to the problem and to Karolina Okrasa for many fruitful discussions.

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 56; pp. 56:1–56:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The k-Coloring problem, which asks whether an input graph G admits a proper coloring with k colors, is arguably one of the best studied computational problems. The problem is known to be notoriously hard: it is polynomial-time solvable (and, in fact, very simple) only for $k \leqslant 2$, and NP-complete otherwise, even on very restricted classes of graphs [15, 19, 20, 26].

For such a hard problem, an interesting direction of research is to study their *fine-grained complexity* depending on some parameters of input instances, in order to understand where the boundary of easy and hard cases lies. Such investigations usually follow two paths in parallel. On one hand, we extend our algorithmic toolbox in order to solve the problem efficiently in various settings. On the other hand, we try to show hardness of the problem, using appropriate reductions.

In order to obtain meaningful lower bounds, the basic assumption of the classical complexity theory, i.e., $\mathsf{P} \neq \mathsf{NP}$, is not strong enough. The usual assumptions used in this context are the Exponential Time Hypothesis (ETH) and the Strong Exponential Time Hypothesis (SETH), both formulated by Impagliazzo and Paturi [21, 22]. The ETH asserts that 3-Sat with n variables cannot be solved in time $2^{o(n)} \cdot n^{\mathcal{O}(1)}$, while the SETH says that CNF-Sat with n variables and m clauses cannot be solved in time $(2-\varepsilon)^n \cdot (n+m)^{\mathcal{O}(1)}$ for any $\varepsilon > 0$.

In case of k-Coloring, the most natural parameter is the number of vertices. While the brute-force approach to solve the problem on an instance G takes time $k^{|V(G)|} \cdot |V(G)|^{\mathcal{O}(1)}$, we know better algorithms where the base of the exponential function does not depend on k. The currently best algorithm is due to Björklund et al. [3] and has complexity $2^{|V(G)|} \cdot |V(G)|^{\mathcal{O}(1)}$. On the other hand, the standard hardness reduction shows that the problem cannot be solved in time $2^{o(|V(G)|)} \cdot |V(G)|^{\mathcal{O}(1)}$, unless the ETH fails [9].

Similarly, we can ask how the complexity depends on some parameters, describing the structure of the instance. The most famous structural parameter is arguably the *treewidth* of the graph, denoted by $\mathrm{tw}(G)$ [2, 5, 36]. Intuitively, treewidth measures how tree-like the graph is. Thus, on graphs with bounded treewidth, we can mimick the bottom-up dynamic programming algorithms that works very well on trees. In case of k-Coloring, the complexity of such a straightforward approach is $k^{\mathrm{tw}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$, provided that G is given along with its tree decomposition of width $\mathrm{tw}(G)$. One might wonder whether this could be improved, in particular, if one can design an algorithm with running time $c^{\mathrm{tw}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$, where c is a constant that does not depend on k, as it was possible in the case if the parameter is $|V(G)|$. Lokshtanov, Marx, and Saurabh [30] proved that this is unlikely, and an algorithm with running time $(k-\varepsilon)^{\mathrm{tw}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$, for any $\varepsilon > 0$, would contradict the SETH. This lower bounds holds even if we replace treewidth with pathwidth $\mathrm{pw}(G)$; the latter result is stronger, as we always have $\mathrm{tw}(G) \leqslant \mathrm{pw}(G)$.

Another way to measure how close a graph G is to a tree or a forest is to analyze the size $\mathrm{fvs}(G)$ of a minimum *feedback vertex set*, i.e., the minimum number of vertices that need to be removed from G to break all cycles. If G is given with a minimum feedback vertex set S, we can solve k-Coloring by enumerating all possible colorings of S, and trying to extend them on the forest $G-S$ using dynamic programming. The running time of such a procedure is $k^{\mathrm{fvs}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$. This is complemented by a hardness result of Lokshtanov et al. [30], who showed that the problem cannot be solved in time $(k-\varepsilon)^{\mathrm{fvs}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$ for any $\varepsilon > 0$, unless the SETH fails. Let us point that $\mathrm{pw}(G)$ and $\mathrm{fvs}(G)$ are incomparable parameters, so this result is incomparable with the previously mentioned lower bound. These two lower bounds were later unified by Jaffke and Jansen [23], who considered the parameterization by the *distance to a linear forest*.

The above examples show a behavior which is typical for many other parameters: the running time of the algorithm depends on the number k of colors and this dependence is necessary under standard complexity assumptions [16, 27, 23]. Thus, it was really surprising that Jansen and Nederlof [25] showed that for any k, the k-Coloring problem can be solved in time $c^{\text{ctw}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$, where c is an absolute constant and $\text{ctw}(G)$ is the *cutwidth* of G. Intuitively, we can imagine $\text{ctw}(G)$ as follows. We fix some ordering of the vertices of G and place them on a horizontal line in this ordering. The edges of G are drawn as arcs above the line; we do not care about intersections. Now, the width of this layout is the maximum number of edges that can be cut by a vertical line. The cutwidth is the minimum width over all linear layouts of vertices of G. The substantial difference between cutwidth and the previously mentioned parameters is that cutwidth corresponds to a number of edges, not a number of vertices. Also, it is known that $\text{pw}(G) \leqslant \text{ctw}(G)$ [4].

Actually, Jansen and Nederlof [25] presented two algorithms for k-Coloring, parameterized by the cutwidth. The first one is deterministic and has running time $2^{\omega \cdot \text{ctw}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$, where $\omega < 2.373$ is the matrix multiplication exponent [7, 39]. The second one is randomized and works in time $2^{\text{ctw}(G)} \cdot |V(H)|^{\mathcal{O}(1)}$. Also, the authors show that the latter complexity is optimal under the SETH.

Let us point out that all the algorithms mentioned above work also for the more general List k-Coloring problem, where each vertex v of G is equipped with a list $L(v) \subseteq \{1, 2, \ldots, k\}$, and we additionally require that the assigned color comes from this list. The general direction of our work is to investigate how further the techniques developed for k-Coloring can be generalized.

Graph homomorphisms. A homomorphism from a graph G to a graph H (called *target*) is an edge-preserving mapping from $V(G)$ to $V(H)$. In the Hom(H) problem we ask if the input graph G admits a homomorphism to H, which is usually treated as a fixed graph. Observe that if H is K_k, i.e., a complete graph on k vertices, then Hom(H) is equivalent to k-Coloring. The complexity classification of Hom(H) was provided by the seminal paper by Hell and Nešetřil [18]: the problem is polynomial-time solvable if H is bipartite or has a vertex with a loop, and NP-complete otherwise. This problem can also be considered in a list setting, where every vertex v of G is equipped with an H*-list* $L(v) \subseteq V(H)$, and we ask for a homomorphism from G to H, which additionally respects lists L. The corresponding computational problem is denoted by LHom(H).

The complexity dichotomy for LHom(H) was proven in three steps: first, for reflexive graphs H (i.e., where every vertex has a loop) by Feder and Hell [11], then for irreflexive graphs H (i.e., with no loops) by Feder, Hell, and Huang [12], and finally, for all graphs H, again by Feder, Hell, and Huang [13]. The problem appears to be polynomial-time solvable if H is a so-called *bi-arc graph.* We will now skip the definition of this class and just mention a special case if H is irreflexive and bipartite: then the LHom(H) problem is in P if the complement of H is a circular-arc graphs, and otherwise the problem is NP-complete. This special case will play a prominent role in our paper.

Let us point out that despite the obvious similarity of Hom(H) and LHom(H), the methods used to prove lower bounds are very different. For Hom(H), all hardness results use some algebraic tools, which allow us to capture the structure of the whole graph H. On the other hand, hardness proofs for LHom(H) are purely combinatorial and are based on the analysis of some small subgraphs of H.

A brute-force approach to solving an instance G of Hom(H) (and LHom(H)) has complexity $|V(H)|^{|V(G)|} \cdot |V(G)|^{\mathcal{O}(1)}$. This can be improved if H has some special structure: several algorithms with running time $\mathcal{O}^*(f(H)^{|V(G)|})$ were obtained, where f is a function of some structural parameter of H [14, 38, 37]. A natural open question was whether one can obtain a $c^{|V(G)|} \cdot |V(G)|^{\mathcal{O}(1)}$ algorithm, where c is a constant that does not depend on H [38]. This question was finally answered in the negative by Cygan et al. [8], who proved that the brute force algorithm is essentially optimal under the ETH.

The fine-grained complexity of the HOM(H) problem, parameterized by the treewidth of G, was studied recently by Okrasa and Rzążewski [34]. The analogous question for LHOM(H) was first investigated by Egri et al. [10] for reflexive graphs H, and then by Okrasa et al. [32] for the general case. The authors defined a new graph invariant $i^*(H)$, and proved the following, tight bounds.

▶ **Theorem 1** (Okrasa, Piecyk, Rzążewski [32]). *Let H be a connected, non-bi-arc graph.*

a) *Every instance (G, L) of LHOM(H) can be solved in time $i^*(H)^t \cdot |V(G)|^{\mathcal{O}(1)}$, provided that G is given along with a tree decomposition of width t.*

b) *There is no algorithm that solves every instance (G, L) of LHOM(H) in time $(i^*(H) - \varepsilon)^{\mathrm{pw}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$ for any $\varepsilon > 0$, unless the SETH fails.*

To the best of our knowledge, the complexity depending on other structural parameters of G was not investigated. In this paper, we make some progress to fill this gap. In particular, our main motivation is the following question by Jansen [24], repeated by Okrasa, Piecyk, Rzążewski [32].

▶ **Question 2** (Jansen [24]). *Is there a universal constant c, such that for every H, every instance G of the HOM(H) problem can be solved in time $c^{\mathrm{ctw}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$?*

Our results. As our first result, we complement the recent result of Okrasa et al. [32] and show tight complexity bounds, parameterized by the size of a minimum feedback vertex set of the instance.

▶ **Theorem 3.** *Let H be a connected, non-bi-arc graph.*

a) *Every instance (G, L) of LHOM(H) can be solved in time $i^*(H)^s \cdot |V(G)|^{\mathcal{O}(1)}$, provided that G is given along with a feedback vertex set of size s.*

b) *There is no algorithm that solves every instance (G, L) of LHOM(H) in time $(i^*(H) - \varepsilon)^{\mathrm{fvs}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$ for any $\varepsilon > 0$, unless the SETH fails.*

Let us point out that the algorithmic part of the theorem, i.e., the statement a), follows directly from Theorem 1 a), as given a graph G and its feedback vertex set S, we can in polynomial time construct a tree decomposition of G with width $|S| + 1$. The proof of the lower bound follows the general direction of the hardness proof for k-COLORING by Lokshtanov et al. [30]. However, as we are showing hardness for all non-bi-arc graphs H, the gadgets are significantly more complicated. In their construction we use some machinery developed by Okrasa et al. [32]. Unfortunately, most of the gadgets used by Okrasa et al. [32] cannot be used as a black box, as they contain many vertex-disjoint cycles. However, we are able to adjust the constructions so that they work in our setting.

Furthermore, similarly to the proof of Theorem 1 b), the proof of Theorem 3 b) is split into two parts: first we prove hardness for the special case if H is bipartite, and then we reduce the general case to the bipartite one.

Then we turn our attention to the setting, where the parameter is the cutwidth of the instance graph. Recall that $\mathrm{ctw}(G) \geqslant \mathrm{pw}(G) \geqslant \mathrm{tw}(G)$. Furthermore, given a linear layout of G with width w, we can in polynomial time construct a tree decomposition of G with width at most w [4]. Thus by Theorem 1 a) we know that LHOM(H) can be solved in time $(i^*(H))^{\mathrm{ctw}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$. On the other hand, we know that this algorithm cannot be optimal for all H, as $i^*(K_k) = k$, while LIST k-COLORING, i.e., LHOM(K_k), can be solved in time $2^{\omega \cdot \mathrm{ctw}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$ [25].

We introduce another parameter $mim^*(H)$, closely related to the size of a maximum induced matching in H, and show the following lower bound.

▶ **Theorem 4.** *For every connected non-bi-arc graph H, there is no algorithm that solves every instance (G, L) of* LHOM(H) *in time $(mim^*(H) - \varepsilon)^{\mathrm{ctw}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$ for any $\varepsilon > 0$, unless the SETH fails.*

As a sanity check, we point out that $mim^*(K_k) = 2$, so our lower bounds are consistent with the results of Jansen and Nederlof [25].

Next, we focus on the non-list variant of the problem, i.e., HOM(H). Note that here we only consider graphs H that are irreflexive and non-bipartite, as otherwise the problem is polynomial-time solvable. Furthermore, we restrict our attention to graphs H that are *projective cores* (see Section 6 for a characterization of these graphs). It is known that almost all irreflexive graphs are non-bipartite, connected projective cores [1, 17, 31, 34]. For this class of graphs H, we show the following lower bounds, answering Question 2 in the negative.

▶ **Theorem 5.** *For every connected non-bipartite, irreflexive projective core H, there is no algorithm that solves every instance G of* HOM(H) *in time $(mim^*(H) - \varepsilon)^{\mathrm{ctw}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$ for any $\varepsilon > 0$, unless the SETH fails.*

In particular, odd cycles are projective cores [28]. Furthermore, for any odd cycle C_k it holds that $mim^*(C_k) = \lfloor 2k/3 \rfloor$. Thus, we obtain the following as a corollary from Theorem 5.

▶ **Corollary 6.** *Assuming the SETH, there is no universal constant c, such that for every odd cycle C, the* HOM(C) *problem can be solved in time $c^{\mathrm{ctw}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$ for every instance G.*

We conclude the paper with pointing out some directions for future investigations.

Full version. Due to the page limit, the proofs of some statements, marked with (♠), are omitted or just sketched. The complete proofs can be found in the full version that is available on arXiv [35]. There we also discuss the consequences of our hardness results, if we only assume the ETH. Finally, we generalize the algorithm for k-COLORING by Jansen and Nederlof [25] so that it could be used to solve LHOM(H) for every graph H, as well as some other closely related problems.

2 Notation and preliminaries

For a positive integer n, we define $[n] := \{1, \ldots, n\}$. For a set X, by 2^X we denote the set of all subsets of X. Unless explicitly stated otherwise, all logarithms are of base 2, i.e., $\log x := \log_2 x$.

Let G be a graph. For a set $S \subsetneq V(G)$, by $G - S$ we denote the graph induced by $V(G) \setminus S$. For a vertex $v \in V(G)$, by $N_G(v)$ we denote the set of neighbors of v and by $\deg_G(v)$ its degree, i.e., $|N_G(v)|$. If the graph G is clear from the context, we write $N(v)$ and $\deg(v)$ instead of $N_G(v)$ and $\deg_G(v)$. Note that $v \in N(v)$ if and only if v is a vertex with a loop. We say that two vertices $u, v \in V(G)$ are *incomparable* if $N(u) \not\subseteq N(v)$ and $N(v) \not\subseteq N(u)$. A set $S \subseteq V(G)$ is *incomparable* if all its vertices are pairwise incomparable. Equivalently, we can say that for every distinct $u, v \in S$, there is a vertex $u' \in N(u) \setminus N(v)$. A set $S \subseteq V(G)$ is *strongly incomparable* if for every $u \in S$ there exists its *private neighbor* $u' \in N(u)$, such that u' is non-adjacent to every vertex in $S \setminus \{u\}$. Clearly, a strongly incomparable set of vertices is incomparable.

For two graphs G and H, we write $\varphi : G \to H$ if φ is a homomorphism from G to H. If G is given with H-lists L, we write $\varphi : (G, L) \to H$ if φ is a homomorphism from G to H, respecting lists L. We also write $G \to H$ (resp., $(G, L) \to H$) to indicate that some homomorphism $\varphi : G \to H$ (resp., $\varphi : (G, L) \to H$) exists. As graph homomorphisms generalize graph colorings, we will often use the term *coloring* to refer to a homomorphism. Moreover, we refer to the vertices of H as *colors*.

For a graph G, H-lists L, and a set $S \subseteq V(G)$ we define $L(S) := \bigcup_{v \in S} L(v)$. If it does not lead to confusion, for a set V such that $V(G) \subseteq V$ and H-lists $L : V \to 2^{V(H)}$, we will denote the instance $(G, L|_{V(G)})$ by (G, L), in order to simplify the notation.

Let H be a graph. A *walk* $\mathcal{P}$ in H is a sequence $p_1, \ldots, p_\ell$ of vertices of H such that $p_i p_{i+1} \in E(H)$ for $i \in [\ell - 1]$. We define the *length* of a walk $\mathcal{P} = p_1, \ldots, p_\ell$ as $\ell - 1$. We also write $\mathcal{P} : p_1 \to p_\ell$ to emphasize that $\mathcal{P}$ starts in p_1 and ends in p_ℓ. For walks $\mathcal{P} = p_1, \ldots, p_\ell$ and $\mathcal{Q} = q_1, \ldots, q_\ell$ of equal length we say $\mathcal{P}$ *avoids* $\mathcal{Q}$ if $p_1 \neq q_1$ and for every $i \in [\ell - 1]$ it holds that $p_i q_{i+1} \notin E(H)$. By $\overline{\mathcal{P}}$ we denote walk $\mathcal{P}$ reversed, i.e., if $\mathcal{P} = p_1, \ldots, p_\ell$, then $\overline{\mathcal{P}} = p_\ell, \ldots, p_1$. It is straightforward to observe that if $\mathcal{P}$ avoids $\mathcal{Q}$, then $\overline{\mathcal{Q}}$ avoids $\overline{\mathcal{P}}$.

Graph parameters. By $\text{tw}(G)$ and $\text{pw}(G)$ we denote, respectively, the *treewidth* and the *pathwidth* of a graph G. A set $F \subseteq V(G)$, such that $G - F$ does not contain any cycle, is called a *feedback vertex set* of G. We denote the size of a minimum feedback vertex set in G by $\text{fvs}(G)$.

Let $\pi = (v_1, \ldots, v_n)$ be a linear ordering of vertices of G, we will call it a *linear layout* of G. A *cut* of π is a partition of $V(G)$ into two subsets: $\{v_1, \ldots, v_p\}$ and $\{v_{p+1}, \ldots, v_n\}$, for some $p \in [n-1]$. We say that an edge $v_i v_j$, where $i < j$, *crosses* the cut $(\{v_1, \ldots, v_p\}, \{v_{p+1}, \ldots, v_n\})$, if $i \leqslant p$ and $j > p$. The *width* of the linear layout π is the maximum number of edges that cross any cut of π. Finally, we define the *cutwidth* $\text{ctw}(G)$ of G as the minimum width over all linear layouts of G.

Given a linear layout of G with width k, we can in polynomial time construct a path decomposition of G with width at most k, so in particular $\text{pw}(G) \leqslant \text{ctw}(G)$ [4]. On the other hand, for every graph G it holds that $\text{ctw}(G) \leqslant \text{pw}(G) \cdot \Delta(G)$ [6]. As we also have that $\text{ctw}(G) \geqslant \Delta(G)/2$, we can intuitively think that $\text{ctw}(G)$ is bounded if and only if both $\Delta(G)$ and $\text{pw}(G)$ are bounded.

Bipartite graphs H, for which LHom(H) is NP-hard. Recall that Feder et al. [12] proved that in this case the LHom(H) problem is polynomial-time solvable if H is a complement of a circular-arc graph and NP-complete otherwise. We will rely on the following structural result.

▶ **Lemma 7** (Okrasa et al. [32, 33]). *Let H be a bipartite graph, whose complement is not a circular-arc graph. Then in each bipartition class there exists a triple (α, β, γ) of vertices such that:*

(1) *there exist $\alpha', \beta' \in V(H)$, such that the edges $\alpha\alpha', \beta\beta'$ induce a matching in H,*

(2) *vertices α, β, γ are pairwise incomparable,*

(3) *there exist walks $\mathcal{X}, \mathcal{X}' : \alpha \to \beta$ and $\mathcal{Y}, \mathcal{Y}' : \beta \to \alpha$, such that $\mathcal{X}$ avoids $\mathcal{Y}$ and $\mathcal{Y}'$ avoids $\mathcal{X}'$,*

(4) *at least one of the following holds:*

  a) *H contains an induced C_6 with consecutive vertices $w_1, \ldots, w_6$ and $\alpha = w_1, \beta = w_5, \gamma = w_3$,*

  b) *H contains an induced C_8 with consecutive vertices $w_1, \ldots, w_8$ and $\alpha = w_1, \beta = w_5, \gamma = w_3$,*

c) *the set $\{\alpha, \beta, \gamma\}$ is strongly incomparable and for any a, b, c, such that $\{a, b, c\} = \{\alpha, \beta, \gamma\}$, there exist walks $\mathcal{X}_c : \alpha \to a$ and $\mathcal{Y}_c : \alpha \to b$, and $\mathcal{Z}_c : \beta \to c$, such that $\mathcal{X}_c, \mathcal{Y}_c$ avoid $\mathcal{Z}_c$ and $\mathcal{Z}_c$ avoids $\mathcal{X}_c, \mathcal{Y}_c$.*

Incomparable sets, decompositions, and main invariants. In this section we still consider H to be a bipartite graph. First, let us define parameters $i(H)$ and $\text{mim}(H)$.

▶ **Definition 8** ($i(H)$ and $mim(H)$)**.** *Let H be a bipartite graph. By $i(H)$ (resp. $mim(H)$) we denote the maximum size of an incomparable set (resp. strongly incomparable set) in H, which is fully contained in one bipartition class.*

Let S be a strongly incomparable set, contained in one bipartition class, and let S' be the set of private neighbors of vertices of S. We observe that the set $S \cup S'$ induces a matching in H of size $|S|$. On the other hand, if M is an induced matching, then the endpoints of edges from M contained in one bipartition class form a strongly incomparable set of size $|M|$. Thus $mim(H)$ can be equivalently defined as the size of a maximum induced matching in H.

Okrasa et al. [32] studied a certain decomposition of bipartite graphs. Its exact definition is not important for us, so we skip it in the conference version. The only thing we need to know is that every bipartite graph, whose complement is not a circular-arc graph, contains an induced subgraph, which is *undecomposable* (i.e., does not admit this decomposition) and its complement is not a circular-arc graph, see e.g. [33, Theorem 46]. This leads to the following definitions.

▶ **Definition 9** ($i^*(H)$ and $mim^*(H)$ for bipartite H)**.** *Let H be a bipartite graph, whose complement is not a circular-arc graph. Define*

$$\begin{aligned} i^*(H) &:= \max\{i(H') \colon H' \textit{ is an undecomposable, connected, induced} \\ &\qquad \textit{subgraph of } H, \textit{ whose complement is not a circular-arc graph}\}, \\ mim^*(H) &:= \max\{mim(H') \colon H' \textit{ is an undecomposable, connected, induced} \\ &\qquad \textit{subgraph of } H, \textit{ whose complement is not a circular-arc graph}\}. \end{aligned}$$

Observe that if H is bipartite, connected, undecomposable, and the complement of H is not a circular-arc graph, then $i^*(H) = i(H)$ and $mim^*(H) = mim(H)$.

3 Bipartite H, parameter: the size of a minimum feedback vertex set

In this section we prove Theorem 3 b) in the case that H is bipartite. Assume that the complement of H is not a circular-arc graph, and let (α, β, γ) be the triple given by Lemma 7.

We will introduce two gadgets. The first one is a graph called an *assignment gadget* and has two special vertices. Its main goal is to ensure that a certain coloring of one special vertex forces a certain coloring of the other special vertex.

▶ **Definition 10** (Assignment gadget)**.** *Let S be an incomparable set in H contained in the same bipartition class as α, β, γ and let $v \in S$. An* assignment gadget *is a graph A_v with H-lists L and with special vertices x, y, such that:*

(A1.) *$L(x) = S$ and $L(y) = \{\alpha, \beta, \gamma\}$,*

(A2.) *for every $u \in S$ and for every $a \in \{\alpha, \beta\}$ there exists a list homomorphism $\varphi : (A_v, L) \to H$ such that $\varphi(x) = u$ and $\varphi(y) = a$,*

(A3.) *there exists a list homomorphism $\varphi : (A_v, L) \to H$ such that $\varphi(x) = v$ and $\varphi(y) = \gamma$,*

(A4.) *for every list homomorphism $\varphi : (A_v, L) \to H$ it holds that if $\varphi(y) = \gamma$, then $\varphi(x) = v$,*

(A5.) $A_v - \{x\}$ *is a tree,*

(A6.) $\deg(x) = (|S|-1)^2$ *and* $\deg(y) = |S|-1$*.*

The second gadget is called a *switching gadget.* It is a path T with a special internal vertex q, whose list is $\{\alpha,\beta,\gamma\}$, and endvertices with the same list $\{\alpha,\beta\}$. Coloring both endvertices of T with the same color, i.e., coloring both with α or both with β, allows us to color q with one of α,β, but "switching sides" from α to β forces coloring q with γ.

▶ **Definition 11** (Switching gadget). *A* switching gadget *is a path T of even length with H-lists L, endvertices p, r, called respectively the* input *and the* output *vertex, and one special internal vertex q, called a q*-vertex*, in the same bipartition class as p, r, such that:*

(S1.) $L(p) = L(r) = \{\alpha,\beta\}$ *and* $L(q) = \{\alpha,\beta,\gamma\}$*,*

(S2.) *for every $a \in \{\alpha,\beta\}$ there exists a list homomorphism $\varphi : (T, L) \to H$, such that $\varphi(p) = \varphi(r) = a$ and $\varphi(q) \neq \gamma$,*

(S3.) *there exists a list homomorphism $\varphi : (T, L) \to H$, such that $\varphi(p) = \alpha$, $\varphi(r) = \beta$, and $\varphi(q) = \gamma$,*

(S4.) *for every list homomorphism $\varphi : (T, L) \to H$, if $\varphi(p) = \alpha$ and $\varphi(r) = \beta$, then $\varphi(q) = \gamma$.*

Note that in a switching gadget we do not care about homomorphisms that map p to β and r to α.

Later, when discussing assignment and switching gadgets, we will use the notions of x-, y-, p-, q-, and r-vertices to refer to the appropriate vertices introduced in the definitions of the gadgets.

▶ **Lemma 12** (♠). *Let H be an undecomposable, connected, bipartite graph, whose complement is not a circular-arc graph. Let (α,β,γ) be the triple from Lemma 7. Let S be an incomparable set in H contained in the same bipartition class as α,β,γ, such that $|S| \geqslant 2$. Then for every $v \in S$ there exists an assignment gadget A_v.*

Sketch of proof. The construction of an assignment gadget A_v is performed in three steps. First, for every $u \in S \setminus \{v\}$, we construct a gadget $\widetilde{F}_u$ with two special vertices x_u, c_u with lists $L(x_u) = S$ and $L(c_u) = \{\alpha,\beta\}$, such that there are list homomorphisms $\varphi : (\widetilde{F}_u, L) \to H$ that map c_u to β and x_u to any vertex from S, or map c_u to α and x_u to any vertex from $S \setminus \{u\}$, but mapping c_u to α and x_u to u is forbidden. The gadget was first introduced in [33, first step in Lemma 4], but our construction is slightly different as we additionally ensure that $\widetilde{F}_u - \{x_u\}$ is a tree and $\deg(x_u) = |S|-1$ and $\deg(c_u) = 1$. We point out that using the original gadgets intruduces many vertex-disjoint cycles, which increases the size of a smallest feedback vertex set in the constructed graph.

In the second step, for every $u \in S \setminus \{v\}$ we construct a path P_u with endvertices c'_u and y_u with lists $L(c'_u) = \{\alpha,\beta\}$ and $L(y_u) = \{\alpha,\beta,\gamma\}$, such that it is possible to map the pair (c'_u, y_u) to any pair of $\{\alpha,\beta\}^2$, or to (α,γ), but the pair (β,γ) is forbidden.

The construction of P_u depends on the case in Lemma 7 (4). In case (4a), P_u is the path with lists of consecutive vertices $\{w_1, w_5\}, \{w_2, w_6\}, \{w_1, w_3, w_5\}$. In case (4b), P_u is the path with lists of consecutive vertices $\{w_1, w_5\}, \{w_2, w_6\}, \{w_1, w_3, w_7\}, \{w_2, w_4, w_6, w_8\}, \{w_1, w_3, w_5\}$.

Finally, in case (4c), we will use walks given by Lemma 7 (4c). We construct an auxiliary path P'_u, such that the list of its i-th vertex is the set of i-th vertices of the walks $\mathcal{X}_\alpha, \mathcal{Y}_\alpha, \mathcal{Z}_\alpha$. Similarly we construct a path P''_u using walks $\overline{\mathcal{X}_\gamma}, \overline{\mathcal{Y}_\gamma}, \overline{\mathcal{Z}_\gamma}$ and a path P'''_u using walks $\mathcal{X}_\gamma, \mathcal{Y}_\gamma, \mathcal{Z}_\gamma$. We obtain P_u by identifying the last vertex of P'_u with the first vertex of P''_u, and the last vertex of P''_u with the first vertex of P'''_u. We set c'_u to be the first vertex of P'_u and y_u to be the last vertex of P'''_u.

Next, we introduce a copy of $\widetilde{F}_u$ and identify the vertex c_u from $\widetilde{F}_u$ with the vertex c'_u from P_u. Now, if x_u is mapped to some vertex from $S \setminus \{u\}$, then y_u can be mapped to any of α, β, γ. However, if x_u is mapped to u, then y_u can only be mapped to α or β. Let F_u be the graph obtained in this step.

Finally, we obtain the assignment gadget A_v by introducing the gadget F_u for every $u \in S \setminus \{v\}$, and identifying all x_u's into one vertex x and all y_u's into one vertex y. ◀

▶ **Lemma 13** (♠)**.** *Let H be an undecomposable, connected, bipartite graph, whose complement is not a circular-arc graph. Let (α, β, γ) be the triple from Lemma 7. Then there exists a switching gadget T.*

Sketch of proof. Again, we consider cases of Lemma 7 (4). In cases (4a) and (4b) the path T is the path with lists of consecutive vertices: $\{w_1, w_5\}$, $\{w_2, w_4\}$, $\{w_1, w_3, w_5\}$, $\{w_2, w_4\}$, $\{w_1, w_5\}$. We set p, q, r to be, respectively, the first, the third, and the fifth vertex of T.

In case (4c) we construct T similarly as we constructed P_u in the proof of Lemma 12, using walks given by Lemma 7 (3) and (4c). We construct a path T' using walks $\mathcal{X}_\beta, \mathcal{Y}_\beta, \mathcal{Z}_\beta$, a path T'' using walks $\overline{\mathcal{X}}_\alpha, \overline{\mathcal{Y}}_\alpha, \overline{\mathcal{Z}}_\alpha$, and a path T''' using walks $\mathcal{X}', \mathcal{Y}'$. We obtain T by identifying the last vertex of T' with the first vertex of T'', and the last vertex of T'' with the first vertex of T'''. The vertices p, q, r are, respectively, the first vertex of T, the last vertex of T', and the last vertex of T. ◀

Reduction. Suppose that we can construct both, the assignment gadget and the switching gadget. Let us show that this is sufficent to prove Theorem 3 b) in the case that H is bipartite. The proof is an extension of the construction of Lokshtanov et al. for the List k-Coloring problem [30].

▶ **Theorem 14.** *For every connected bipartite graph H, whose complement is not a circular-arc graph, there is no algorithm that solves every instance (G, L) of LHom(H) in time $(i^*(H) - \varepsilon)^{\mathrm{fvs}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$ for any $\varepsilon > 0$, unless the SETH fails.*

Proof. Let us point out that it is sufficient to show the theorem if we additionally assume that H is undecomposable. Indeed, assume the SETH and suppose the theorem holds for every bipartite undecomposable graph H', and it does not hold for every bipartite graph H. Then there exist a connected, bipartite graph H, whose complement is not a circular-arc graph, and an algorithm that solves LHom(H) for every instance (G, L) in time $(i^*(H) - \varepsilon)^{\mathrm{fvs}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$ for some $\varepsilon > 0$. Let H' be an induced subgraph of H such that H' is connected, undecomposable, is not a complement of a circular-arc graph, and $i(H') = i^*(H)$. Any instance (G, L) of LHom(H') can be seen as an instance of LHom(H) such that only vertices of H' appear on lists L. Thus we can solve any instance (G, L) of LHom(H') in time $(i^*(H) - \varepsilon)^{\mathrm{fvs}(G)} \cdot |V(G)|^{\mathcal{O}(1)} = (i(H') - \varepsilon)^{\mathrm{fvs}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$, a contradiction.

So from now on we assume that H is undecomposable. In particular, $i^*(H) = i(H)$. Let ϕ be an instance of CNF-Sat with n variables and m clauses. Let $\varepsilon > 0$ and $k := i(H)$. Let S be a maximum incomparable set contained in one bipartition class of H, i.e., $|S| = k$. Let α, β, γ be the vertices of H, in the same bipartition class as S, given by Lemma 7. Let α', β' be the vertices such that edges $\alpha\alpha', \beta\beta'$ induce a matching in H, they exist by Lemma 7. Observe that $k \geqslant 3$, since vertices α, β, γ are pairwise incomparable. Moreover, we define $\lambda := \log_k(k - \varepsilon)$. Observe that $\lambda < 1$. We choose a positive integer p sufficiently large so that $\lambda \frac{p}{p-1} < 1$ and define $t := \left\lceil \frac{n}{\lfloor \log k^p \rfloor} \right\rceil = \left\lceil \frac{n}{\lfloor p \cdot \log k \rfloor} \right\rceil$.

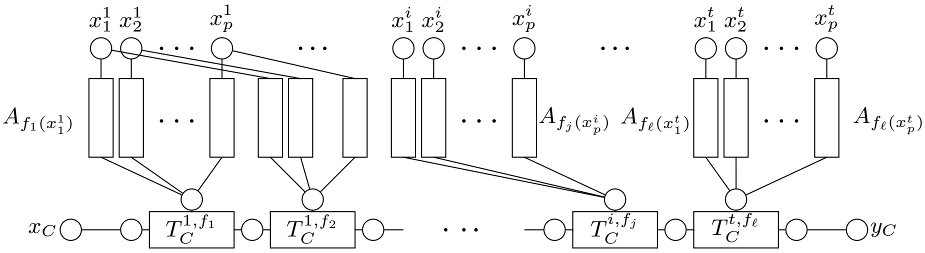


Figure 1 The path P_C for a clause C and vertices x_s^i for $i \in [t], s \in [p]$.

We will construct a graph G with H-lists L such that $\text{fvs}(G) \leqslant t \cdot p$ and $(G, L) \to H$ if and only if ϕ is satisfiable. We partition the variables of ϕ into t sets $F_1, \ldots, F_t$ called *groups*, such that $|F_i| \leqslant \lfloor \log k^p \rfloor$. For each $i \in [t]$ we introduce p vertices $x_1^i, \ldots, x_p^i$ and for every $s \in [p]$ we set $L(x_s^i) := S$. We will interpret a coloring of these vertices as a truth assignment of variables in F_i. Note that there are at most $2^{\lfloor \log k^p \rfloor} \leqslant k^p$ possible truth assigments of variables in F_i and there are k^p possible colorings of $x_1^i, \ldots, x_p^i$, respecting lists L. Thus we can define an injective mapping that assigns a distinct coloring of vertices $x_1^i, \ldots, x_p^i$ to each truth assignment of the variables in F_i, note that some colorings may remain unassigned.

For every clause C of ϕ we introduce a path P_C constructed as follows. Consider a group F_i that contains at least one variable from C, and a truth assignment of F_i that satisfies C. Recall that this assignment corresponds to a coloring f of vertices $x_1^i, \ldots, x_p^i$. We introduce a switching gadget $T_C^{i,f}$, whose q-vertex is denoted by $q_C^{i,f}$. We fix an arbitrary ordering of all switching gadgets introduced for the clause C. For every switching gadget but the last one, we identify its output vertex with the input vertex of the succesor. We add vertices x_C with $L(x_C) = \{\alpha'\}$ and y_C with $L(y_C) = \{\beta'\}$. We add an edge between x_C and the input of the first switching gadget, and between y_C and the output of the last switching gadget. This completes the construction of P_C.

Now consider a switching gadget $T_C^{i,f}$ introduced in the previous step. Recall that C is a clause of ϕ, and f is a coloring of $x_1^i, \ldots, x_p^i$ corresponding to a truth assignment of variables in F_i, which satisfies C. Let us define $v_s := f(x_s^i)$ for $s \in [p]$. For every $s \in [p]$, we call Lemma 12 to construct the assignment gadget A_{v_s}. We identify the x-vertex of A_{v_s} with x_s^i and the y-vertex with $q_C^{i,f}$. This completes the construction of (G, L) (see Figure 1). The properties of the gadgets ensure that ϕ is satisfiable if and only if $(G, L) \to H$ (♠). Furthermore, $|V(G)| = (n+m)^{\mathcal{O}(1)}$ and $\bigcup_{i=1}^t \{x_1^i, \ldots, x_p^i\}$ is a feedback vertex set in G, so $\text{fvs}(G) \leqslant t \cdot p$ (♠).

Suppose that the instance (G, L) of LHom(H) can be solved in time $(k - \varepsilon)^{\text{fvs}(G)} \cdot |V(G)|^{\mathcal{O}(1)} \leqslant (k-\varepsilon)^{t \cdot p} \cdot |V(G)|^{\mathcal{O}(1)}$. Recall that ϕ is satisfiable if and only if $(G, L) \to H$. By a careful analysis of the exponent in the complexity bound (♠) we conclude that ϕ can be solved in time $(2-\delta)^n \cdot (n+m)^{\mathcal{O}(1)}$ for some $\delta > 0$, which contradicts the SETH. ◀

Let us point out that the pathwidth of the graph constructed in the proof above is bounded by $t \cdot p + f(H)$, for some function f of H (see also [30]).

4 Bipartite H, parameter: cutwidth

Similarly as in the previous section, let us first prove Theorem 4 in the case that H is bipartite. We will modify the reduction from Theorem 14. To get an intuition about what needs to be done, recall that in order to obtain a bound on the cutwidth, we need to bound the pathwidth and the maximum degree. Also, as we already observed, the pathwidth of the instance constructed in Theorem 14 is upper-bounded by the correct value, so we need to take care of vertices of large degree.

▶ **Theorem 15.** *For every connected bipartite graph H, whose complement is not a circular-arc graph, there is no algorithm that solves every instance (G, L) of* LHOM(H)*, given with a linear layout of width at most w, in time $(mim^*(H) - \varepsilon)^w \cdot |V(G)|^{\mathcal{O}(1)}$ for any $\varepsilon > 0$, unless the SETH fails.*

Proof. First, similarly to the proof of Theorem 14 in the case that H is bipartite, it is sufficient to show the proof in case that H is undecomposable. In particular $mim^*(H) = mim(H)$.

Let S be a strongly incomparable set in H of size $k = mim(H)$, contained in one bipartition class. Let S' be a set such that $S \cup S'$ induces a matching of size k in H, and let (α, β, γ) be the triple given by Lemma 7, such that α, β, γ are in the same bipartition class as S. Let ϕ be an instance of CNF-SAT with n variables and m clauses. Let $\varepsilon > 0$. As in the proof of Theorem 14, we choose an integer p so that $\log_k(k - \varepsilon) \cdot \frac{p}{p-1} < 1$ and set $t := \left\lceil \frac{n}{\lfloor p \cdot \log k \rfloor} \right\rceil$. We will construct an instance $(\widetilde{G}, \widetilde{L})$ of LHOM(H) with a linear layout of width at most $t \cdot p + f(H)$, where f is some function of H, such that $(\widetilde{G}, \widetilde{L}) \to H$ if and only if ϕ is satisfiable. We repeat the construction of the instance (G, L) of LHOM(H), such that $(G, L) \to H$ if and only if ϕ is satisfiable, from the proof of Theorem 14. This is possible since S is in particular incomparable. Furthermore, in the construction of (G, L) we did not use the fact that S was maximum, we only needed that $|S| \geqslant 2$, which is the case as $\{\alpha, \beta\}$ is strongly incomparable. We are going to modify the instance (G, L) into the desired instance $(\widetilde{G}, \widetilde{L})$.

Before we do that, let us fix an arbitrary ordering of clauses $C_1, \ldots, C_m$ in ϕ, which implies the ordering of paths P_C in G. Then we can fix an ordering of all q-vertices in G, so that a q-vertex q_1 precedes a q-vertex q_2, if:

- q_1 belongs to the path P_{C_i} and q_2 belongs to the path P_{C_j}, such that $i < j$, or
- q_1 and q_2 belong to the same path P_C, and q_1 precedes q_2 on P_C (the order of the vertices of each path P_C is such that x_C is the first vertex and y_C is the last vertex).

Finally, we fix an ordering of the assignment gadgets in G. Recall that every q-vertex $q_C^{i,f}$ is a y-vertex of p assignment gadgets whose x-vertices are, respectively, $x_1^i, \ldots, x_p^i$. We fix an ordering of the assignment gadgets so that the assignment gadget A_1 precedes the assignment gadget A_2 if:

- the y-vertex of A_1 precedes the y-vertex of A_2 in the fixed order of the q-vertices, or
- A_1 and A_2 have the same y-vertex $q_C^{i,f}$ and x-vertices of A_1 and A_2 are, respectively, x_j^i and x_s^i, with $j < s$.

Now we are ready to modify the instance (G, L). It turns out that we only need to take care of q-vertices and x-vertices, as their large degree forces large cutwidth. The construction of $(\widetilde{G}, \widetilde{L})$ will be thus performed in two steps.

Step 1. Splitting q-vertices. Recall that every q-vertex of a switching gadget is a y-vertex of p assignment gadgets and the degree of each y-vertex in the assignment gadget is $k - 1$. For every q-vertex q, in order to reduce its degree, we will split q into $p \cdot (k - 1)$ vertices $q_1, \ldots, q_{p \cdot (k-1)}$. In this step, the construction depends on the structure of H. Let us consider two cases.

Case I. The set $\{\alpha, \beta, \gamma\}$ is strongly incomparable. Let $\overline{\alpha}, \overline{\beta}, \overline{\gamma}$ be vertices such that edges $\alpha\overline{\alpha}, \beta\overline{\beta}, \gamma\overline{\gamma}$ induce a matching in H. We replace every q-vertex q from a path P_C with $p \cdot (k - 1)$ vertices $q_1, \ldots, q_{p \cdot (k-1)}$, each for every neighbor of q inside assignment gadgets.

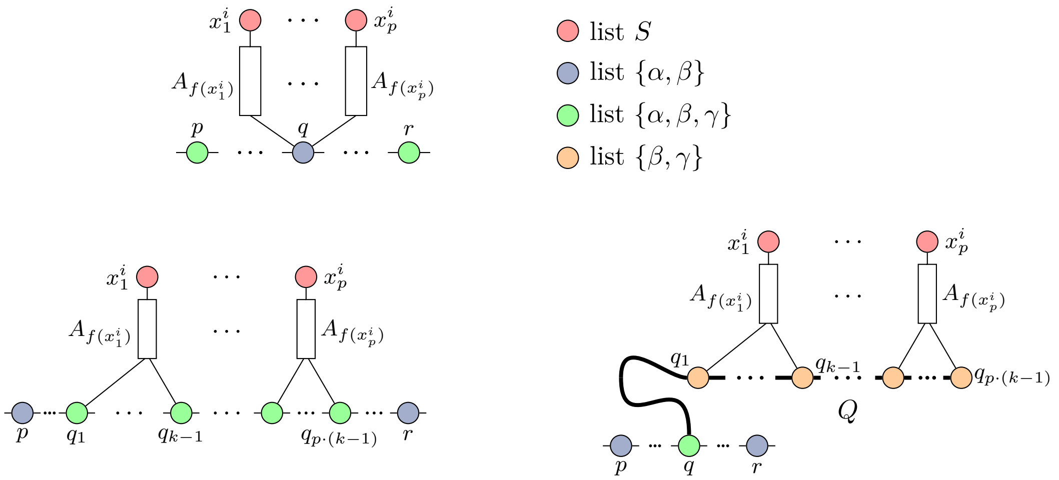


Figure 2 The switching gadget T and the group of vertices x_s^i for $s \in [p]$ before the step of splitting q-vertices (top), after the step in the case that $\{\alpha, \beta, \gamma\}$ is a strongly incomparable set (bottom, left), and after introducing the path Q (marked by the bold curve) in the case that $\{\alpha, \beta, \gamma\}$ is not strongly incomparable (bottom, right).

For every $j \in [p \cdot (k-1) - 1]$ we introduce a path Q_j of length 2 with lists of consecutive vertices $\{\alpha, \beta, \gamma\}, \{\overline{\alpha}, \overline{\beta}, \overline{\gamma}\}, \{\alpha, \beta, \gamma\}$, and we identify its endvertices with q_j and q_{j+1}. In the same way, we introduce paths Q_0 and $Q_{p\cdot(k-1)}$ and we identify endvertices of Q_0 with q_1 and the vertex preceding q on P_C, and we identify endvertices of $Q_{p\cdot(k-1)}$ with $q_{p\cdot(k-1)}$ and the vertex following q on P_C (see Figure 2). Finally, let us fix an ordering $a_1, a_2, \ldots, a_{p\cdot(k-1)}$ of neighbors of q in assignment gadgets such that for $j \in [p-1]$ vertices of the assignment gadget with the x-vertex x_j^i precede vertices of the assignment gadget with the x-vertex x_{j+1}^i. The order of the neighbors from the same assignment gadget is arbitrary. For every $j \in [p \cdot (k-1)]$ we add an edge between q_j and a_j (see Figure 2). This completes the step of splitting q-vertices in this case.

Case II: The set $\{\alpha, \beta, \gamma\}$ is not strongly incomparable. By Lemma 7 this means that H contains an induced C_6 or C_8 with consecutive vertices $w_1, \ldots, w_6(, w_7, w_8)$ and $\alpha = w_1$, $\beta = w_5$, $\gamma = w_3$. In this case we leave each q-vertex q in the graph, but we introduce a path Q with H-lists L, with q as one of endvertices, special vertices q_j for $j \in [p \cdot (k-1)]$, with list $L(q_j) = \{\beta, \gamma\}$ and such that:

- for every list homomorphism $\varphi : (Q, L) \to H$, if q is mapped to γ, then for every $j \in [p \cdot (k-1)]$ the vertex q_j is mapped to γ.
- there exists a list homomorphism $\varphi : (Q, L) \to H$ such that $\varphi(q) = \gamma$ and $\varphi(q_j) = \gamma$ for every $j \in [p \cdot (k-1)]$.
- for every $c \in \{\alpha, \beta\}$ there exists a list homomorphism $\varphi : (Q, L) \to H$ such that q is mapped to c and for every $j \in [p \cdot (k-1)]$ the vertex q_j is mapped to β.

The path Q is constructed using the walks from Lemma 7, similarly as we did in Lemma 12 and Lemma 13 (♠). Again, for each neighbor a_j of q (the neighbors of $q_C^{i,f}$ are ordered as in the previous case) we add an edge $q_j a_j$ and remove the edge qa_j (see Figure 2).

This completes the Step 1. We will refer to the newly introduced vertices q_j as q-vertices.

Step 2. Splitting x-vertices. The only vertices that might still have large degree are vertices from $\{x_j^i \mid i \in [t], j \in [p]\}$. More precisely, the degree of the x-vertex in an assignment gadget is $(k-1)^2$, and thus the degree of an x-vertex x is $d = d(x) \cdot (k-1)^2$, where $d(x)$

is the number of the assignment gadgets, whose x-vertex is x. We replace the vertex x with d vertices $x_1, \ldots, x_d$, each with list S. For every $s \in [d-1]$ we introduce a path X_s of length 2, lists of consecutive vertices S, S', S, and we identify its endvertices with x_s and x_{s+1}, respectively. We fix an ordering $b_1, \ldots, b_d$ of neighbors of x, such that if b_i and b_j belong, respectively, to assignment gadgets A_i and A_j, and A_i precedes A_j in the fixed order of the assignment gadgets, then b_i precedes b_j. The order of the neighbors from the same assignment gadget is arbitrary. For every $s \in [d]$ we add an edge $b_s x_s$. We will refer to the new vertices x_j introduced in this step also as x-vertices.

It can be verified that $(\widetilde{G}, \widetilde{L}) \to H$ if and only if ϕ is satisfiable (♠). Furthermore, we can specify a linear layout π of $\widetilde{G}$ with width at most $w := t \cdot p + f(H)$, for some function f, as follows (♠). We order the vertices of the original paths P_C (those from graph G), such that the vertices from P_{C_j} precede vertices of $P_{C_{j+1}}$, and the vertices from one path P_C are ordered in a natural way (the vertex x_C is the first one and the vertex y_C is the last one). Then, if Case 1. in Step 1. was applied, we replace each q-vertex q with vertices q_j and vertices of paths Q_j in the following order: $Q_0, q_1, Q_1, \ldots, q_{p\cdot(k-1)}, Q_{p\cdot(k-1)}$. If Case 2. was applied, we insert the vertices from the path Q just after q, in the natural order with q being the first one.

Now we need to place the vertices of assignment gadgets and of paths X_s. We insert the vertices of an assignment gadget A_v, whose y-vertex was q, just after q-vertices adjacent to A_v, which were introduced for q in Step 1. We also insert there the vertices from those paths X_s, whose endvertices are adjacent to A_v.

To see that the width of π is at most $t \cdot p + f(H)$ observe that we placed vertices from each assignment gadget close to each other and to the vertices adjacent to that gadget. The number of the edges with at least one endpoint in a fixed assignment gadget is bounded by some constant $f(H)$. The only edges between vertices that are possibly "far" are edges from paths X connecting x-vertices adjacent to different assignment gadgets and in each cut their number is bounded by the number of original x-vertices, i.e., $t \cdot p$.

Now suppose there is an algorithm that solves every instance (G, L) of LHom(H) in time $(k-\varepsilon)^w \cdot |V(G)|^{\mathcal{O}(1)}$. Then for an instance ϕ of CNF-Sat we can construct the instance $(\widetilde{G}, \widetilde{L})$ as above and we can solve $(\widetilde{G}, \widetilde{L})$ in time $(k-\varepsilon)^{t\cdot p + f(H)} \cdot |V(G)|^{\mathcal{O}(1)}$, which is equivalent to solving the instance ϕ. As in the proof of Theorem 14, we conclude that this implies that CNF-Sat with n variables and m clauses can be solved in time $(2-\delta)^n \cdot (n+m)^{\mathcal{O}(1)}$ for some $\delta > 0$, which contradicts the SETH. ◀

5 Hardness for general target graphs

For a graph H, the *associated bipartite graph* H^* is the graph with vertex set $V(H^*) = \{v', v'' \mid v \in V(H)\}$, whose edge set contains those pairs $u'v''$, for which $uv \in E(H)$.

Recall that for general graphs H, Feder et al. [13] showed that the LHom(H) problem is polynomial-time solvable if H is a bi-arc graph, and NP-complete otherwise. They also observed that H is a bi-arc graph if and only if H^* is the complement of a circular-arc graph. Furthermore, an irreflexive graph is bi-arc if and only if it is bipartite and its complement is a circular-arc graph. Thus "hard" cases of LHom(H) correspond to the "hard" cases of LHom(H^*).

Observe that if H is bipartite, then H^* consists of two disjoint copies of H. Thus for bipartite H it holds that $i^*(H^*) = i^*(H)$ and $mim^*(H^*) = mim^*(H)$. On the other hand, if H is non-bipartite and additionally connected, then H^* is connected. This motivates the following extension of the definition of i^* and mim^* to non-bipartite H.

▶ **Definition 16** ($i^*(H)$ and $mim^*(H)$)**.** *Let H be a non-bi-arc graph. Define:*

$$i^*(H) := i^*(H^*) \qquad \textit{and} \qquad mim^*(H) := mim^*(H^*).$$

Observe that that the instances (G, L) constructed in the proofs of Theorem 14 or Theorem 15 are bipartite. Indeed, the instance constructed in Theorem 14 consists of paths P_C and assignment gadgets, whose vertices were appropriately identified. More precisely, we identify some q-vertices (from switching gadgets belonging to paths P_C) with y-vertices (from addignment gadgets), and x-vertices with another x-vertices (from assignment gadgets). Recall that each assignment gadget is bipartite by property (A3.) of Definition 10. Moreover, for each assignment gadget, the x-vertex is in the same bipartition class as the y-vertex. Similarly, for each path P_C, all q-vertices are in the same bipartition class. Therefore, the instance (G, L) constructed in Theorem 14 is bipartite. The instance from Theorem 15 was obtained from the instance (G, L) from Theorem 14 by splitting some vertices into a set of vertices joined by paths of even lenght, so the instance remains bipartite.

Furthermore, if the bipartition classes of G are X and Y, then $L(X)$ is contained in one bipartition class of H, and $L(Y)$ is contained in the other one. Finally, without loss of generality we can assume that for each $v \in V(G)$, the set $L(v)$ is incomparable. Indeed, if $L(v)$ contains two distincts vertices x, y, such that $N_H(x) \subseteq N_H(y)$, we can safely remove x from $L(v)$, obtaining an equivalent instance. Instances satisfying these three conditions are called *consistent*.

▶ **Proposition 17** ([32, 33])**.** *Let H be a graph and let (G, L) be a consistent instance of* LHom*(H^*). Define L' as $L'(x) := \{u \colon \{u', u''\} \cap L(x) \neq \emptyset\}$. Then $(G, L) \to H^*$ if and only if $(G, L') \to H$.*

Now we can prove Theorem 3 b) and Theorem 4 (♠).

Sketch of proof of Theorem 3 b) and Theorem 4. If H is bipartite, we are done by Theorem 14 or Theorem 15. Otherwise H^* is connected. Let (G, L) be an instance of LHom(H^*) constructed in the proof of Theorem 14 or Theorem 15. Let (G, L') be an equivalent instance of LHom(H) given by Proposition 17. As the instance graph remains the same, the lower bound holds. ◀

Note that the statement of Theorem 15 actually implies the following, slightly stronger result.

▶ **Corollary 18.** *For every connected non-bi-arc graph H, there is no algorithm that solves every instance (G, L) of* LHom*(H), given with a linear layout of width at most w, in time $(mim^*(H) - \varepsilon)^w \cdot |V(G)|^{\mathcal{O}(1)}$ for any $\varepsilon > 0$, unless the SETH fails.*

6 Hardness of Hom(H)

In this section we extend Theorem 4 to the non-list case, i.e., we prove Theorem 5. Let us first discuss the graph class mentioned in the statement. Recall that Hom(H) is NP-hard if H is non-bipartite and has no loops [18]. In particular, this implies that H has at least three vertices. We say that a graph H is a *core* if every homomorphism $\varphi : H \to H$ is an automorphism, i.e., is injective and surjective. We also need the following characterization of projective graphs.

▶ **Theorem 19** (Larose, Tardif [29]). *Let H be graph with at least three vertices. Then H is projective if and only if for every $L \subseteq V(H)$ there exist a tuple $(x_1, \ldots, x_\ell)$ of vertices in H and a graph F_L with a tuple of its vertices $(y_0, y_1, \ldots, y_\ell)$ such that*

$$L = \{\varphi(y_0) \mid \varphi : F_L \to H, \text{such that } \varphi(y_1) = x_1, \ldots, \varphi(y_\ell) = x_\ell\}.$$

Now we are ready to prove Theorem 5.

▶ **Theorem 5.** *For every connected non-bipartite, irreflexive projective core H, there is no algorithm that solves every instance G of* Hom*(H) in time $(mim^*(H) - \varepsilon)^{\mathrm{ctw}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$ for any $\varepsilon > 0$, unless the SETH fails.*

Sketch of proof. Let H be as in the statement. As non-bipartite irreflexive graphs are not bi-arc graphs [13], we can use Corollary 18. Let (G, L) be an instance of LHom(H), and let $\pi = (v_1, \ldots, v_{|V(G)|})$ be a linear layout of G of width w. Consider an instance $\widetilde{G}$ of Hom(H) constructed as follows. For every $v_i \in V(G)$ we call Theorem 19 to obtain the tuple $(x_1^{(i)}, \ldots, x_{\ell_i}^{(i)})$ of vertices in H and a graph $F_{L(v_i)}$ with special vertices $y_0^{(i)}, \ldots, y_{\ell_i}^{(i)}$. For every v_i we introduce a copy $H^{(i)}$ of the graph H and identify vertices $y_1^{(i)}, \ldots, y_{\ell_i}^{(i)}$, respectively with $x_1^{(i)}, \ldots, x_{\ell_i}^{(i)}$ in the copy $H^{(i)}$. Moreover, we identify $y_0^{(i)}$ with v_i. Finally, for every $i \in [|V(G)| - 1]$ we add edges between the copies $H^{(i)}$ and $H^{(i+1)}$ as follows. For every vertex $z^{(i)}$ in $H^{(i)}$ and its corresponding copy $z^{(i+1)}$ in $H^{(i+1)}$ we add all edges between $z^{(i)}$ and $N_{H^{(i+1)}}(z^{(i+1)})$. This completes the construction of $\widetilde{G}$.

Theorem 19 implies that $(G, L) \to H$ if and only if $\widetilde{G} \to H$: every graph $F_{L(v)}$ together with a copy of H forces that a vertex v can be colored only with vertices from $L(v)$ and thus it imitates the list of v (♠). Furthermore, $\mathrm{ctw}(\widetilde{G}) \leqslant w + g(H)$ for some function g of H: the copies of H are connected in the appropriate order and thus the linear layout π of G can be easily modified to a linear layout $\widetilde{\pi}$ of $\widetilde{G}$ with width larger than w only by a constant depending on H (♠).

Now suppose that Hom(H) can be solved for every instance G' in time $(mim^*(H) - \varepsilon)^{\mathrm{ctw}(G')} \cdot |V(G')|^{\mathcal{O}(1)}$ for some $\varepsilon > 0$. Then, for an instance (G, L) of LHom(H) with a linear layout π of width w, we can construct in polynomial time the instance $\widetilde{G}$ of Hom(H) as above. We solve the instance $\widetilde{G}$ in time $(mim^*(H) - \varepsilon)^{\mathrm{ctw}(\widetilde{G})} \cdot |V(\widetilde{G})|^{\mathcal{O}(1)}$, which is equivalent to solving the instance (G, L) in time $(mim^*(H) - \varepsilon)^w \cdot |V(G)|^{\mathcal{O}(1)}$. By Corollary 18, this contradicts the SETH. ◀

7 Conclusion

A natural open question is to close the gap between lower and upper bounds for the complexity of LHom(H), parameterized by the cutwidth. As a concrete problem, we believe that a good starting point is to understand the complexity of LHom(C_k), where $k \geqslant 5$. Recall that we have a lower bound $mim^*(C_k)^{\mathrm{ctw}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$ and an upper bound $i^*(C_k)^{\mathrm{ctw}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$. The value of $mim^*(C_k)$ is $\lfloor k/3 \rfloor$ if k is even, and $\lfloor 2k/3 \rfloor$ if k is odd. On the other hand, $i^*(C_k)$ is $k/2$ if k is even, and k if k is odd. Where does the truth lie? To be even more specific, what is the complexity of LHom(C_6)?

Another research direction that we find exciting is to study the complexity of Hom(H) and LHom(H), depending on different parameters of the instance graph. In particular, Lampis [27] showed that k-Coloring on a graph G can be solved in time $\mathcal{O}^*((2^k - 2)^{\mathrm{cw}(G)})$, where $\mathrm{cw}(G)$ is the *clique-width* of G. Furthermore, an algorithm with a running time $\mathcal{O}^*((2^k - 2 - \varepsilon)^{\mathrm{cw}(G)})$, for any $\varepsilon > 0$, would contradict the SETH. We believe it is exciting to investigate how these results generalize to non-complete target graphs H.

References

1 Noga Alon and Joel H. Spencer. *The Probabilistic Method, Third Edition.* Wiley-Interscience series in discrete mathematics and optimization. Wiley, 2008.

2 Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discret. Appl. Math.*, 23(1):11–24, 1989. doi:10.1016/0166-218X(89)90031-0.

3 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009. doi:10.1137/070683933.

4 H. L. Bodlaender. Classes of graphs with bounded tree-width. *Bulletin of EATCS*, pages 116–128, 1988.

5 Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Comput. J.*, 51(3):255–269, May 2008. doi:10.1093/comjnl/bxm037.

6 Fan R. K. Chung and Paul D. Seymour. Graphs with small bandwidth and cutwidth. *Discret. Math.*, 75(1-3):113–119, 1989. doi:10.1016/0012-365X(89)90083-6.

7 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990. doi:10.1016/S0747-7171(08)80013-2.

8 Marek Cygan, Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, Ivan Mihajlin, Jakub Pachocki, and Arkadiusz Socala. Tight lower bounds on graph embedding problems. *J. ACM*, 64(3):18:1–18:22, 2017. doi:10.1145/3051094.

9 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015. doi:10.1007/978-3-319-21275-3.

10 László Egri, Dániel Marx, and Paweł Rzążewski. Finding list homomorphisms from bounded-treewidth graphs to reflexive graphs: a complete complexity characterization. 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 27:1–27:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.27.

11 Tomás Feder and Pavol Hell. List homomorphisms to reflexive graphs. *Journal of Combinatorial Theory, Series B*, 72(2):236–250, 1998. doi:10.1006/jctb.1997.1812.

12 Tomás Feder, Pavol Hell, and Jing Huang. List homomorphisms and circular arc graphs. *Combinatorica*, 19(4):487–505, 1999. doi:10.1007/s004939970003.

13 Tomás Feder, Pavol Hell, and Jing Huang. Bi-arc graphs and the complexity of list homomorphisms. *Journal of Graph Theory*, 42(1):61–80, 2003. doi:10.1002/jgt.10073.

14 Fedor V. Fomin, Pinar Heggernes, and Dieter Kratsch. Exact algorithms for graph homomorphisms. *Theory Comput. Syst.*, 41(2):381–393, 2007. doi:10.1007/s00224-007-2007-x.

15 M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976. doi:10.1016/0304-3975(76)90059-1.

16 Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Cliquewidth III: the odd case of graph coloring parameterized by cliquewidth. 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 262–273. SIAM, 2018. doi:10.1137/1.9781611975031.19.

17 Pavol Hell and Jaroslav Nešetřil. *Graphs and homomorphisms.* Oxford University Press, 2004.

18 Pavol Hell and Jaroslav Nešetřil. On the complexity of H-coloring. *J. Comb. Theory, Ser. B*, 48(1):92–110, 1990. doi:10.1016/0095-8956(90)90132-J.

19 Ian Holyer. The NP-completeness of edge-coloring. *SIAM J. Comput.*, 10(4):718–720, 1981. doi:10.1137/0210055.

20 Shenwei Huang. Improved complexity results on k-coloring P_t-free graphs. *Eur. J. Comb.*, 51:336–346, 2016. doi:10.1016/j.ejc.2015.06.005.

21 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

22 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

23 Lars Jaffke and Bart M. P. Jansen. Fine-grained parameterized complexity analysis of graph coloring problems. In Dimitris Fotakis, Aris Pagourtzis, and Vangelis Th. Paschos, editors, *Algorithms and Complexity - 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*, volume 10236 of *Lecture Notes in Computer Science*, pages 345–356, 2017. doi:10.1007/978-3-319-57586-5_29.

24 Bart M. P. Jansen. Personal communication.

25 Bart M. P. Jansen and Jesper Nederlof. Computing the chromatic number using graph decompositions via matrix rank. *Theor. Comput. Sci.*, 795:520–539, 2019. doi:10.1016/j.tcs.2019.08.006.

26 Sanjeev Khanna, Nathan Linial, and Shmuel Safra. On the hardness of approximating the chromatic number. *Combinatorica*, 20(3):393–415, 2000. doi:10.1007/s004930070013.

27 Michael Lampis. Finer tight bounds for coloring on clique-width. 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 86:1–86:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.86.

28 Benoît Larose. Families of strongly projective graphs. *Discuss. Math. Graph Theory*, 22(2):271–292, 2002. doi:10.7151/dmgt.1175.

29 Benoit Larose and Claude Tardif. Strongly rigid graphs and projectivity. *Multiple-Valued Logic*, 7:339–361, 2001.

30 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.

31 Tomasz Łuczak and Jaroslav Nešetřil. Note on projective graphs. *Journal of Graph Theory*, 47(2):81–86, 2004.

32 Karolina Okrasa, Marta Piecyk, and Paweł Rzążewski. Full complexity classification of the list homomorphism problem for bounded-treewidth graphs. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 74:1–74:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.74.

33 Karolina Okrasa, Marta Piecyk, and Paweł Rzążewski. Full complexity classification of the list homomorphism problem for bounded-treewidth graphs. *CoRR*, abs/2006.11155, 2020. arXiv:2006.11155.

34 Karolina Okrasa and Paweł Rzążewski. Fine-grained complexity of graph homomorphism problem for bounded-treewidth graphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms*, pages 1578–1590, 2020. doi:10.1137/1.9781611975994.97.

35 Marta Piecyk and Paweł Rzążewski. Fine-grained complexity of the list homomorphism problem: feedback vertex set and cutwidth. *CoRR*, abs/2009.11642, 2020. arXiv:2009.11642.

36 Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

37 Paweł Rzążewski. Exact algorithm for graph homomorphism and locally injective graph homomorphism. *Inf. Process. Lett.*, 114(7):387–391, 2014. doi:10.1016/j.ipl.2014.02.012.

38 Magnus Wahlström. New plain-exponential time classes for graph homomorphism. *Theory Comput. Syst.*, 49(2):273–282, 2011. doi:10.1007/s00224-010-9261-z.

39 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898. ACM, 2012. doi:10.1145/2213977.2214056.

6-Uniform Maker-Breaker Game Is PSPACE-Complete

Md Lutfar Rahman ✉
University of Memphis, TN, USA

Thomas Watson ✉
University of Memphis, TN, USA

Abstract

In a STOC 1976 paper, Schaefer proved that it is PSPACE-complete to determine the winner of the so-called Maker-Breaker game on a given set system, even when every set has size at most 11. Since then, there has been no improvement on this result. We prove that the game remains PSPACE-complete even when every set has size 6.

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

Keywords and phrases Game, Maker-Breaker, Complexity, Reduction, PSPACE-complete, NL-hard

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.57

Related Version *Full Version*: https://eccc.weizmann.ac.il/report/2020/097/

Funding This work was supported by NSF grants CCF-1657377 and CCF-1942742.

1 Introduction

The Maker-Breaker game is a perfect-information game played on a set system – a collection of subsets of some finite universe. The two players, called Maker and Breaker, alternate turns. In each turn, the current player claims a previously-unclaimed element of the universe as his own. Maker wins if he claims every element in at least one subset. Breaker wins if he claims at least one element in every subset. There are no draws, and for every set system, one of the players has a strategy that guarantees that he wins. The popular game of Hex can be viewed as a Maker-Breaker game.

Maker-Breaker games were introduced in the influential paper [9], which provided a sufficient condition for Breaker to win (and is often considered the forerunner to the method of conditional probabilities). There is a very substantial literature on determining which player has a winning strategy, for various kinds of set systems (and for many generalizations and variants of Maker-Breaker games). We refer to [14] for a survey. Some cornerstones of this literature are:

- When the universe is the set of edges of an undirected graph with distinguished nodes s and t, and the subsets are s-t paths (this special case is called the "Shannon switching game"), Lehman [16] characterized which player can win, in terms of combinatorial properties of the graph.
- When the universe is the set of edges of a sufficiently large complete undirected graph, and the subsets are Hamiltonian cycles, Chvátal and Erdös [5] proved that Maker can win.

Given the effort that has gone into determining the winner for various set systems, it is natural to consider the possibility of automating this process. In other words, let us view this as a computational problem and investigate how efficiently it can be solved.

> *What is the computational complexity of determining which player has a winning strategy in the Maker-Breaker game on a given set system?*

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 57; pp. 57:1–57:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In a seminal paper, Schaefer [19, 20] proved that the problem is PSPACE-complete, even when the set system has *width* 11, which means each subset in the system has size at most 11. (A simplified proof of PSPACE-completeness for unbounded width was given in [4].) Reductions from this theorem have been used for many other PSPACE-completeness results [1, 3, 4, 6, 7, 8, 10, 11, 12, 13, 17, 21, 22, 24, 25].

Since Schaefer's PSPACE-completeness result first appeared in 1976, there has been no improvement on the width 11. We make the first progress in 44 years: Determining the winner of the Maker-Breaker game remains PSPACE-complete even for set systems of width 6. As we note later, this also implies PSPACE-completeness of Maker-Breaker for set systems that are 6-uniform, meaning that every subset has size exactly 6.

1.1 CNF games

In this section, we introduce "CNF games," a broader sense of games that includes Maker-Breaker as a special case.

- In the *ordered* game, the input consists of a conjunctive normal form (CNF) formula φ and an ordered list of variables $\{x_{2n}, x_{2n-1}, \dots, x_2, x_1\}$ that contains all variables of φ. Player 1 is called T because his goal is to make φ true, and player 2 is called F because his goal is to make φ false. In the first round, T assigns a bit value for x_{2n}, then F assigns a bit value for x_{2n-1}. In the next round, T assigns x_{2n-2}, then F assigns x_{2n-3}, and so on for n rounds. The winner depends on whether φ is satisfied by the resulting assignment. In other words, which player has a winning strategy is determined by whether the following quantified boolean formula is true:

  $$(\exists x_{2n})(\forall x_{2n-1}) \cdots (\exists x_2)(\forall x_1) : \varphi(x_1, \dots, x_{2n})$$

  The problem w-TQBF is to determine which player has a winning strategy, under the restriction that φ has width w (every clause has at most w literals). It is known that 2-TQBF is NL-complete [2] and 3-TQBF is PSPACE-complete [23].
- In the *unordered* game, the input consists of a CNF φ, a set X of variables that contains all variables of φ (and possibly more), and an indication of which player (T or F) gets the first move. Again, T and F alternate turns assigning bit values to variables, and the winner depends on whether φ is satisfied by the resulting assignment. But now, each turn consists of picking which remaining variable to assign, as well as which bit to assign it. The unordered game more closely resembles real-world games in which the same moves are available to both players. The problem G_w is to determine which player has a winning strategy, under the restriction that φ has width w. The paper [17] originated the G_w notation and showed that G_2 is in L and G_5 is PSPACE-complete.
- The *unordered positive* game is just the unordered game under the restriction that φ must be a positive (a.k.a. monotone) CNF – it only has unnegated literals. In this game, it would never be advantageous for T to assign 0 to a variable, or for F to assign 1 to a variable. Thus we can assume each move consists of T picking a remaining variable and assigning it 1, or F picking a remaining variable and assigning it 0. If we view each clause of φ as a subset of X (the set of variables), then the unordered positive game is equivalent to the Maker-Breaker game on the set system corresponding to (φ, X), where F is Maker (he wants to assign every variable in at least one clause) and T is Breaker (he wants to assign at least one variable in every clause). The problem G_w^+ is the restriction of G_w to positive w-CNFs, i.e., determining whether Maker or Breaker has a winning strategy on a given set system of width w. Thus, Schaefer's theorem [19, 20] can be stated as: G_{11}^+ is PSPACE-complete.

Previously, [18] conjectured that G_3^+, and perhaps even G_3, might actually be tractable. These problems have been shown to be tractable – indeed, in L– under various restrictions on the 3-CNF [15, 18]. The unordered CNF game seems qualitatively very different from its ordered counterpart. Width 6 might not be optimal for PSPACE-completeness of Maker-Breaker (though it appears to be a barrier for our proof technique), but it is unclear what the optimal width ought to be.

In this paper, we prove the following three results:

▶ **Theorem 1.** *G_6^+ is PSPACE-complete.*

▶ **Theorem 2.** *G_5^+ is NL-hard.*

▶ **Theorem 3.** *G_4 is NL-hard.*

In Table 1 we summarize the state-of-the-art for the ordered, unordered, and unordered positive CNF games.

Table 1 Results.

$w \to$	2	3	4	5	6
w-TQBF	NL-complete [2]	PSPACE-complete [23]			
G_w	L [17]	L under restrictions [18]	NL-hard [Theorem 3]	PSPACE-complete [17]	
G_w^+		L under restrictions [15]	Unknown	NL-hard [Theorem 2]	PSPACE-complete [Theorem 1]

Each game has four different patterns for "who has the first move" and "who has the last move." For $a, b \in \{T, F\}$ we use the subscript $a \cdots b$ to indicate that player a goes first and b goes last. For example, $G_{6,T\cdots F}^+$ is G_6^+ restricted to instances where T has the first move and F has the last move (which necessitates $|X|$ being even). With no such subscript, an instance of G_6^+ must specify which player goes first (and then the parity of $|X|$ determines who goes last). We prove that G_6^+ is PSPACE-complete for each of the four possible patterns, and similarly for G_5^+ being NL-hard, but we are only able to show NL-hardness of G_4 for the patterns $T \cdots F$ and $F \cdots F$.

Our proof of Theorem 1 follows a similar high-level outline as the proof that G_{11}^+ is PSPACE-complete from [19, 20], using a reduction from 3-TQBF. The key is to trade size for width – we develop a gadget for simulating a round of the ordered game, using more variables and clauses but lower width than the gadget from [19, 20]. Our correctness analysis also uses a new perspective on the case where T is supposed to win (which is much trickier than the case where F is supposed to win, since T must satisfy every clause whereas F only needs to falsify one clause). To frame T's winning strategy in the event that F "misbehaves," we make use of ideas from the recent paper [18].

The proof of Theorem 1 also yields Theorem 2. Theorem 3 holds by an elementary but new reduction from 2-SAT, which appears in the full version of this paper.

2 Proof of Theorem 1 (and Theorem 2)

We prove Theorem 1 in Subsection 2.1. In Subsection 2.2 we provide a streamlined proof of a special case of a lemma from [18], which is needed for the proof of Theorem 1. Then we prove a series of corollaries in Section 3, which cover all the patterns for both Theorem 1 and Theorem 2.

2.1 Proof of Theorem 1

We show 3-TQBF $\leqslant \mathrm{G}^+_{6,\mathrm{T}\cdots\mathrm{F}}$. Suppose an instance of 3-TQBF is given by

$$(\exists x_{2n})(\forall x_{2n-1})\cdots(\exists x_2)(\forall x_1) : F_1 \wedge F_2 \wedge \cdots \wedge F_m$$

where each F_k is a clause with width $\leqslant 3$. We construct an instance of $\mathrm{G}^+_{6,\mathrm{T}\cdots\mathrm{F}}$ as (φ^+, X) where φ^+ is a positive 6-CNF and X is the set of variables in it, such that T has a winning strategy in the 3-TQBF game iff T has a winning strategy in the $\mathrm{G}^+_{6,\mathrm{T}\cdots\mathrm{F}}$ game on (φ^+, X).

A 3-TQBF round $(\exists x_i)(\forall x_{i-1})$, where $i \in \{2, 4, 6, \ldots, 2n\}$, will correspond to 16 variables in X and 14 clauses in φ^+. Four of the 16 variables are $\{x_i, \overline{x}_i, x_{i-1}, \overline{x}_{i-1}\}$. Here, $\overline{x}_i$ is the name of an unnegated variable, distinct from the variable x_i. The variables x_i and $\overline{x}_i$ do not necessarily get assigned opposite values. Similarly for x_{i-1} and $\overline{x}_{i-1}$. The other 12 variables associated with a 3-TQBF round $(\exists x_i)(\forall x_{i-1})$ are $\{u_{6i}, u_{6i-1}, \ldots, u_{6i-11}\}$. (This variable naming scheme is borrowed from [19, 20].) In the $\mathrm{G}^+_{6,\mathrm{T}\cdots\mathrm{F}}$ game, we define "legitimate" gameplay corresponding to a 3-TQBF round $(\exists x_i)(\forall x_{i-1})$ as follows:

1. T plays one of x_i, $\overline{x}_i$
2. F plays the remaining variable in the pair x_i, $\overline{x}_i$
3. T plays u_{6i}
4. F plays u_{6i-1}
5. T plays u_{6i-2}
6. F plays u_{6i-3}
7. T plays u_{6i-4}
8. F plays one of x_{i-1}, $\overline{x}_{i-1}$
9. T plays the remaining variable in the pair x_{i-1}, $\overline{x}_{i-1}$
10. F plays u_{6i-5}
11. T plays u_{6i-6}
12. F plays u_{6i-7}
13. T plays u_{6i-8}
14. F plays u_{6i-9}
15. T plays u_{6i-10}
16. F plays u_{6i-11}

In the $\mathrm{G}^+_{6,\mathrm{T}\cdots\mathrm{F}}$ game, T always assigns 1 and F always assigns 0 to variables. In a legitimate gameplay, T choosing x_i or $\overline{x}_i$ to assign 1 is like T choosing to assign $x_i = 1$ or $x_i = 0$ (respectively) in the 3-TQBF game. Similarly, F choosing x_{i-1} or $\overline{x}_{i-1}$ to assign 0 is like F choosing to assign $x_{i-1} = 0$ or $x_{i-1} = 1$ (respectively) in the 3-TQBF game.

We say the gameplay for the entire $\mathrm{G}^+_{6,\mathrm{T}\cdots\mathrm{F}}$ game is legitimate when it consists of legitimate gameplay for the $(\exists x_{2n})(\forall x_{2n-1})$ round, followed by legitimate gameplay for the $(\exists x_{2n-2})(\forall x_{2n-3})$ round, followed by legitimate gameplay for the $(\exists x_{2n-4})(\forall x_{2n-5})$ round, and so on. Legitimate gameplay mimics the 3-TQBF gameplay in a natural way. We will design the clauses so that any player who plays illegitimately either outright loses, or at least gains no advantage by deviating from legitimate gameplay.

The 14 clauses associated with the 3-TQBF round $(\exists x_i)(\forall x_{i-1})$ are:

$$\begin{aligned}
A_i &= x_i \vee \overline{x}_i \vee u_{6i+1} \vee u_{6i+3} \vee u_{6i+5} \\
C_{6i} &= u_{6i} \vee u_{6i+1} \vee u_{6i+3} \vee u_{6i+5} \vee (x_i \wedge \overline{x}_i) \\
C_{6i-2} &= u_{6i-2} \vee u_{6i-1} \vee u_{6i+1} \vee u_{6i+3} \vee (x_i \wedge \overline{x}_i) \\
C_{6i-4} &= u_{6i-4} \vee u_{6i-3} \vee u_{6i-1} \vee u_{6i+1} \vee (x_i \wedge \overline{x}_i) \\
B_i &= x_{i-1} \vee \overline{x}_{i-1} \vee u_{6i-3} \vee u_{6i-1} \\
C_{6i-6} &= u_{6i-6} \vee u_{6i-5} \vee u_{6i-3} \vee u_{6i-1} \vee (x_{i-1} \wedge \overline{x}_{i-1}) \\
C_{6i-8} &= u_{6i-8} \vee u_{6i-7} \vee u_{6i-5} \vee u_{6i-3} \vee (x_{i-1} \wedge \overline{x}_{i-1}) \\
C_{6i-10} &= u_{6i-10} \vee u_{6i-9} \vee u_{6i-7} \vee u_{6i-5} \vee (x_{i-1} \wedge \overline{x}_{i-1})
\end{aligned}$$

As we note later, each C_j is not really a clause, since it contains a conjunction, but it is equivalent to a pair of clauses. Thus the six C_j's correspond to 12 clauses, but we often refer to C_j as "a clause" anyway. Note that each C_j contains one even-index u variable and the three previous odd-index u variables. For any clause that appears to contain some u_j variable where $j > 12n$, that non-existent variable is actually not present in the clause. Intuitively, the variables x_i and $\overline{x}_i$ in A_i, and x_{i-1} and $\overline{x}_{i-1}$ in B_i, and u_j in C_j (which we wrote first in the clauses) enable F to threaten T with defeat if T plays illegitimately, and the other variables in the clauses enable T to threaten F with defeat if F plays illegitimately.

For each clause F_k in the 3-TQBF game we introduce a clause

$$D_k = F'_k \vee u_1 \vee u_3 \vee u_5$$

where F'_k is the clause which results from replacing each negated variable $\neg x_i$ by the unnegated variable $\overline{x}_i$ throughout the clause F_k. For example, if $F_k = (x_1 \vee \neg x_2 \vee \neg x_3)$ then $F'_k = (x_1 \vee \overline{x}_2 \vee \overline{x}_3)$, where x_2, $\overline{x}_2$, x_3, $\overline{x}_3$ are separate variables.

In summary, the formal construction is as follows:

$$\begin{aligned}
X &= \{x_1, \overline{x}_1, x_2, \overline{x}_2, \ldots, x_{2n}, \overline{x}_{2n}\} \cup \{u_1, u_2, \ldots, u_{12n}\} \\
&= \bigcup_{i=2,4,6,\ldots,2n} \{x_i, \overline{x}_i, x_{i-1}, \overline{x}_{i-1}, u_{6i}, u_{6i-1}, \ldots, u_{6i-11}\} \\
\varphi^+ &= \bigwedge_{i=2,4,6,\ldots,2n} (A_i \wedge B_i) \wedge \bigwedge_{j=2,4,6,\ldots,12n} (C_j) \wedge \bigwedge_{k=1,2,3,\ldots,m} (D_k)
\end{aligned}$$

where

$$\begin{aligned}
A_i &= x_i \vee \overline{x}_i \vee u_{6i+1} \vee u_{6i+3} \vee u_{6i+5} \\
B_i &= x_{i-1} \vee \overline{x}_{i-1} \vee u_{6i-3} \vee u_{6i-1} \\
C_j &= u_j \vee u_{j+1} \vee u_{j+3} \vee u_{j+5} \vee (x_{\lceil j/6 \rceil} \wedge \overline{x}_{\lceil j/6 \rceil}) \\
D_k &= F'_k \vee u_1 \vee u_3 \vee u_5
\end{aligned}$$

Any occurrence of a non-existent variable u_j (where $j > 12n$) is omitted from the clauses. For example, A_{2n} is simply the clause $x_{2n} \vee \overline{x}_{2n}$. Now:

$$C_j = (u_j \vee u_{j+1} \vee u_{j+3} \vee u_{j+5} \vee x_{\lceil j/6 \rceil}) \wedge (u_j \vee u_{j+1} \vee u_{j+3} \vee u_{j+5} \vee \overline{x}_{\lceil j/6 \rceil})$$

So C_j contains two clauses with width $\leqslant 5$, and A_i, B_i, and D_k are individual clauses with widths $\leqslant 5$, $\leqslant 4$, and $\leqslant 6$ respectively. Therefore, φ^+ is a positive 6-CNF with $16n$ variables and $14n + m$ clauses. Though C_j contains two clauses we often treat C_j as a clause in the proof. The construction is now complete. Furthermore, (φ^+, X) can be constructed in logarithmic space.

Now we claim T has a winning strategy in the 3-TQBF game iff T has a winning strategy in the $\mathrm{G}^{+}_{6,\mathrm{T}\cdots\mathrm{F}}$ game (φ^+, X).

First we prove in Lemma 4 that the claim holds if the gameplay is restricted to be legitimate. Then we prove that the claim still holds even if the gameplay is not legitimate. In Lemma 5 we show if T plays illegitimately then either the game will be restored to a legitimate situation with no advantage to T, or F will win immediately. In Lemma 6 we show if F plays illegitimately then either the game will be restored to a legitimate situation with no advantage to F, or a chain reaction will be started that enables T to win eventually.

▶ **Lemma 4.** T *has a winning strategy in the* 3-TQBF *game iff* T *has a winning strategy in the* $\mathrm{G}^{+}_{6,\mathrm{T}\cdots\mathrm{F}}$ *game* (φ^+, X) *when gameplay is restricted to be legitimate.*

Proof. A legitimate gameplay satisfies all A_i, B_i, C_j since A_i is satisfied by one of x_i or $\overline{x}_i$, B_i is satisfied by one of x_{i-1} or $\overline{x}_{i-1}$, and C_j is satisfied by u_j where j is even because they have been played by T. Since F plays u_1, u_3, u_5 we know that D_k gets satisfied iff F'_k gets satisfied. Furthermore, F'_k gets satisfied iff F_k gets satisfied by the assignment to the x_i variables (ignoring the $\overline{x}_i$ variables), because of the definition of F'_k and the fact that x_i and $\overline{x}_i$ get opposite values. In summary, a legitimate gameplay satisfies φ^+ iff $F_1 \wedge F_2 \wedge \cdots \wedge F_m$ gets satisfied by the assignment to the x_i variables.

Suppose F has a winning strategy in the 3-TQBF game. We describe F's winning strategy in (φ^+, X). F can use the same strategy to pick one from $x_{i-1}, \overline{x}_{i-1}$ where F picking x_{i-1} or $\overline{x}_{i-1}$ is equivalent to assigning $x_{i-1} = 0$ or $x_{i-1} = 1$ respectively in the 3-TQBF game. F wins since this strategy makes the assignment to all the x_i variables match F's strategy in the 3-TQBF game, which ensures $F_1 \wedge \cdots \wedge F_m$ is unsatisfied and hence φ^+ is unsatisfied.

Suppose T has a winning strategy in the 3-TQBF game. We describe T's winning strategy in (φ^+, X). T can use the same strategy to pick one from $x_i, \overline{x}_i$ where T picking x_i or $\overline{x}_i$ is equivalent to assigning $x_i = 1$ or $x_i = 0$ in the 3-TQBF game respectively. T wins since this strategy makes the assignment to all the x_i variables match T's strategy in the 3-TQBF game, which ensures $F_1 \wedge \cdots \wedge F_m$ is satisfied and hence φ^+ is satisfied. ◀

▶ **Lemma 5.** *If* F *has a winning strategy in the* 3-TQBF *game then* F *has a winning strategy in the* $\mathrm{G}^{+}_{6,\mathrm{T}\cdots\mathrm{F}}$ *game* (φ^+, X) *even if the gameplay does not progress legitimately.*

Proof. Suppose F has a winning strategy in the 3-TQBF game. In the game (φ^+, X), F can follow his strategy from Lemma 4 until T plays illegitimately on move p (p is odd and $1 \leqslant p \leqslant 16$) at round $(\exists x_i)(\forall x_{i-1})$. We consider all the different cases of p:

- $p = 1$: F already played u_{6i+1}, u_{6i+3}, u_{6i+5} (or these variables do not exist if $i = 2n$) due to legitimate gameplay before this move. T was supposed to play x_i or $\overline{x}_i$ but T did not do so. There are two possibilities:
  - If T also did not play u_{6i}, then F plays u_{6i}. Then whatever T plays, F plays one of x_i, $\overline{x}_i$. F wins since C_{6i} is unsatisfied.
  - If T played u_{6i}, then F plays one of x_i or $\overline{x}_i$ (it does not matter which one). Now it is T's move. If T plays the other from x_i, $\overline{x}_i$ then the game comes back to a legitimate situation at move 4, where F has no disadvantage since T effectively let F make the choice of x_i or $\overline{x}_i$ for him. If T does not play the other from x_i, $\overline{x}_i$ then F plays it and wins since A_i is unsatisfied.
- $p = 9$: F already played u_{6i-3}, u_{6i-1} and one of x_{i-1}, $\overline{x}_{i-1}$ due to legitimate gameplay before this move. T was supposed to play the other one from x_{i-1}, $\overline{x}_{i-1}$ but T did not do so. F plays it and wins since B_i is unsatisfied.

- Other p: T was supposed to play u_j where j is even, but T did not do so. F already played u_{j+1}, u_{j+3}, u_{j+5} and one of $x_{\lceil j/6\rceil}$, $\overline{x}_{\lceil j/6\rceil}$ due to legitimate gameplay before this move. Then F plays u_j and wins since C_j is unsatisfied. ◀

▶ **Lemma 6.** *If* T *has a winning strategy in the* 3-TQBF *game then* T *has a winning strategy in the* $\mathrm{G}^{+}_{6,\mathrm{T}\cdots\mathrm{F}}$ *game* (φ^+, X) *even if the gameplay does not progress legitimately.*

▶ **Definition 7.** *We define an* ***order on all the clauses****:* A_i*,* C_{6i}*,* C_{6i-2}*,* C_{6i-4}*,* B_i*,* C_{6i-6}*,* C_{6i-8}*,* C_{6i-10} *for* $i = 2n$ *then the same for* $i = 2n-2$*, and so on. Finally all* D_k *are at the end ordered by* k *increasing. To represent an* ***interval of clauses*** *from this order, we use analogous mathematical notations "(", ")", "[", "]". For example,* $[A_{2n}, C_t)$ *means all the clauses from* A_{2n} *(inclusive) to* C_t *(exclusive). Let* V_t *be all the variables that occur at least once in* $(C_t, C_2]$ *along with* $\{u_1, u_3, u_5\}$*. For example,* $V_2 = \{u_1, u_3, u_5\}$ *and* $V_4 = \{u_1, u_2, u_3, u_5, u_7, x_1, \overline{x}_1\}$*.*

▶ **Lemma 8.** *If* $[A_{2n}, C_t]$ *are already satisfied where* $t \leqslant 12n-4$ *and* F *has already played at most one variable in* V_t*, then* T *has a strategy to satisfy* $(C_t, D_m]$ *even if it is* F*'s turn.*

Before proving Lemma 8, we use it to prove Lemma 6.

Proof of Lemma 6. Suppose T has a winning strategy in the 3-TQBF game. In the game (φ^+, X), T can follow his strategy from Lemma 4 until F plays illegitimately on move p (p is even and $1 \leqslant p \leqslant 16$) at round $(\exists x_i)(\forall x_{i-1})$. The outline of the argument is: The legitimate gameplay so far will have satisfied an interval of clauses, from A_{2n} through some clause associated with round $(\exists x_i)(\forall x_{i-1})$. After the illegitimate move by F, there might be another opportunity for F to restore the gameplay to a legitimate situation with no disadvantage to T. If that opportunity does not exist, or if F fails to get the gameplay "back on track," then T will have a move that satisfies the next few clauses. Then for some t (t stands for "threshold"), $[A_{2n}, C_t]$ will be satisfied, and it will be F's turn and T will satisfy the rest of the clauses (and hence win) by Lemma 8. The illegitimate move by F could have happened in V_t or somewhere else, and none of the other prior moves happened in V_t.

We consider all the different cases of p:

- $p = 2$: $[A_{2n,}C_{6i})$ are already satisfied due to legitimate gameplay before this move. F was supposed to play the other one from x_i, $\overline{x}_i$ but F did not do so. Then T plays that and that satisfies $[C_{6i,}C_{6i-4}]$. Now it is F's turn and T wins by Lemma 8 with $t = 6i-4$.
- $p = 8$: $[A_{2n,}B_i)$ are already satisfied due to legitimate gameplay before this move. F was supposed to play one from x_{i-1}, $\overline{x}_{i-1}$ but F did not do so. There are two possibilities:
  - If F played u_{6i-5}, then T plays one of x_{i-1} or $\overline{x}_{i-1}$ (it does not matter which one). Now it is F's move. If F plays the other from x_{i-1}, $\overline{x}_{i-1}$ then the game comes back to a legitimate situation at move 11, where T has no disadvantage since F effectively let T make the choice of x_{i-1} or $\overline{x}_{i-1}$ for him. If F does not play the other from x_{i-1}, $\overline{x}_{i-1}$ then T plays it and that satisfies $[B_i, C_{6i-10}]$, so now it is F's turn and T wins by Lemma 8 with $t = 6i-10$.
  - If F did not play u_{6i-5}, then T plays u_{6i-5} and that satisfies $[C_{6i-6,}C_{6i-10}]$. Let us pretend, for a moment, that one of x_{i-1} or $\overline{x}_{i-1}$ has already been played by T and the other has already been played by F (though in reality, neither has been played yet). Then B_i and hence all of $[A_{2n}, C_{6i-10}]$ are satisfied, and F's illegitimate move was the only variable that may have been played so far among V_{6i-10}, and it is F's turn, so T would win by Lemma 8 with $t = 6i-10$. In reality, T can use that strategy from Lemma 8, and whenever F plays one of x_{i-1} or $\overline{x}_{i-1}$, T responds by playing the other, then resumes the strategy from Lemma 8. (Or, if F never plays x_{i-1} or $\overline{x}_{i-1}$, then T

will play one of them after concluding his strategy from Lemma 8, and F will have to play the other as the final move.) Then B_i gets satisfied along with $(C_{6i-10}, D_m]$, so T wins.

- $p = 16$: $[A_{2n}, C_{6i-10}]$ are already satisfied due to legitimate gameplay before this move. F was supposed to play u_{6i-11} but F did not do so. Here $i > 2$ since if $i = 2$ then $u_{6i-11} = u_1$, which will be the only leftover variable to play and F must play it. So we only consider $i > 2$. Then T plays u_{6i-11} (which is $u_{6(i-2)+1}$) and that satisfies $[A_{i-2}, C_{6(i-2)-4}]$. Now it is F's turn and T wins by Lemma 8 with $t = 6(i-2) - 4$.
- Other p: F was supposed to play u_{j+1} (2nd variable in C_j and j is even) but F did not do so. $[A_{2n}, C_j)$ are already satisfied due to legitimate gameplay before this move. Then T plays u_{j+1}. There are two possibilities of j:
  - $j \leqslant 4$: T's move u_{j+1} satisfies $[C_j, D_m]$ since all D_k are satisfied by u_{j+1} (which is either u_3 or u_5). Therefore T wins.
  - $j > 4$: T's move u_{j+1} satisfies $[C_j, C_{j-4}]$. Now it is F's turn and T wins by Lemma 8 with $t = j - 4$. ◀

To prove Lemma 8, we need Lemma 10, which concerns "tree-like" positive 3-CNFs. Lemma 10 follows from [18], but for completeness we provide a streamlined, self-contained proof in Subsection 2.2.

▶ **Definition 9.** *A positive* 3*-CNF is a* **tree** *if each of the following holds:*

(1) *Each clause has width exactly* 3*, so the formula can be viewed as a* 3*-uniform hypergraph where variables are nodes and clauses are hyperedges.*

(2) *Each clause has at least one "spare variable" that occurs in no other clauses.*

(3) *Any two clauses share at most one variable.*

(4) *If we delete a spare variable from every clause, the resulting graph (*2*-uniform hypergraph) would be a tree (i.e., connected and no cycles).*

When we say F can use pass moves, this means F has the option of forgoing any turn, thus forcing T to play multiple variables in a row.

▶ **Lemma 10.** *For every tree,* T *has a winning strategy even if* F *gets to play the first two moves and* F *can use pass moves.*

Proof of Lemma 8. Shrink the clauses $(C_t, D_m]$ by removing some variables from them as follows:

$$\begin{aligned}
A'_i &= x_i \vee \overline{x}_i \vee u_{6i+3} \\
B'_i &= x_{i-1} \vee \overline{x}_{i-1} \vee u_{6i-3} \\
C'_j &= u_j \vee u_{j+3} \vee u_{j+5} && \text{(previously two clauses, now only one)} \\
D' &= u_1 \vee u_3 \vee u_5 && \text{(all } D'_k \text{ are the same, we call it just } D'\text{)}
\end{aligned}$$

All these clauses form a positive 3-CNF ψ. The hypergraph for ψ has been illustrated in Figure 1. We argue that ψ is a tree. We show it satisfies each of the four properties of a tree as described in Definition 9.

- Tree property (1) holds since each of A'_i, B'_i, C'_j, D' has exactly 3 variables. The variables u_{6i+3} in A'_i, and u_{j+3} and u_{j+5} in C'_j, are guaranteed to exist since $t \leqslant 12n - 4$.
- Tree property (2) holds since x_i, x_{i-1}, u_j, u_1 only occur in A'_i, B'_i, C'_j, D' respectively.
- Tree property (3) holds since:
  - C'_j and A'_i share only u_{6i+3} if $j = 6i$ or $j = 6i - 2$.
  - C'_j and B'_i share only u_{6i-3} if $j = 6i - 6$ or $j = 6i - 8$.
  - C'_j and C'_{j-2} share only u_{j+3}.

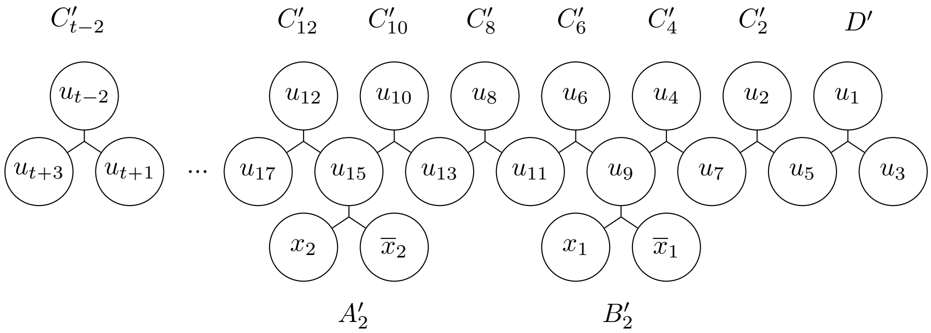


Figure 1 Hypergraph for ψ.

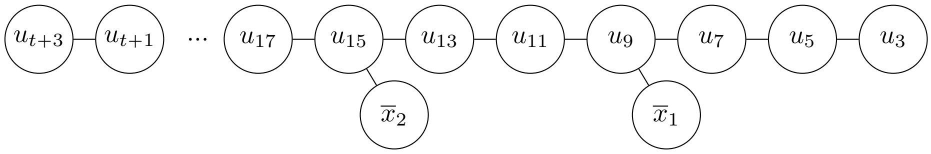


Figure 2 Hypergraph after deleting a spare variable from each clause in ψ.

  - C'_2 and D' share only u_5.
  - Other pairs do not share a variable.
- Tree property (4) holds since deleting x_i, x_{i-1}, u_j, u_1 (which are spare variables) from A'_i, B'_i, C'_j, D' respectively creates a 2-uniform hypergraph as shown in Figure 2 which is clearly a tree.

Therefore ψ is a tree.

By Lemma 10, T has a winning strategy on the tree ψ even if F has the first two moves (and subsequently T and F play alternately) and F can use pass moves. Now we claim that T has a strategy to satisfy $(C_t, D_m]$ in φ^+ assuming F has already played at most one variable in V_t and it is F's turn (and F cannot use pass moves). Because every variable in ψ is also in V_t, we can say F has already played at most one variable of ψ. Because it is F's turn in φ^+, that's like allowing F to have the second move in ψ as well. After that, T's strategy for φ^+ is the same as T's winning strategy for ψ, except that whenever F plays a variable of φ^+ that's not in ψ, T interprets it as a pass move by F and continues with his strategy for ψ. Since this strategy ensures that ψ gets satisfied, it also ensures that $(C_t, D_m]$ and hence all of φ^+ gets satisfied. ◀

2.2 Trees

In order to prove Lemma 10, we need Lemma 12 and Lemma 13. First we outline some definitions.

▶ **Definition 11.** *We henceforth refer to a tree as a* ***single tree****. A* ***married tree*** *is a formula consisting of two disjoint single trees ("spouses") and a width-2 clause with one endpoint in each spouse (and every width-3 clause has a spare variable even after the inclusion of the width-2 clause). The endpoints of the width-2 clause in a married tree are considered roots of the spouses. A* ***win-forest*** *is a formula where each connected component is either a single tree or a married tree.*

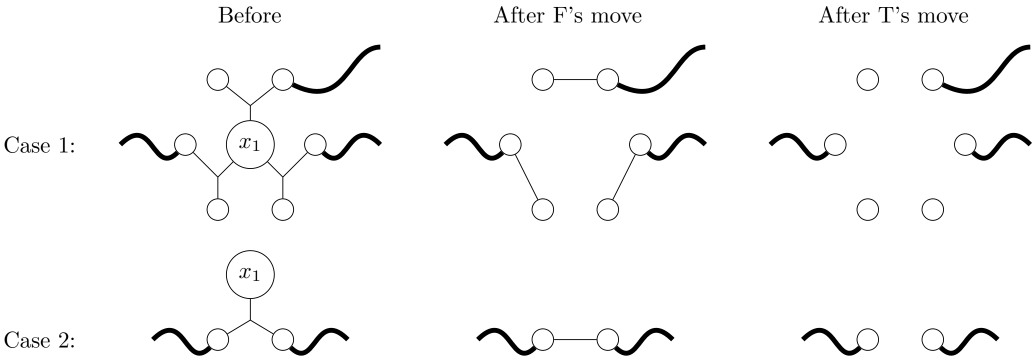


Figure 3 F's move and T's move on x_1 and its effect on formulas.

After any move by T or F, a formula changes to a residual formula where the variable that got played is removed, and if T played then any clause containing the variable disappears (since it is satisfied), and if F played then any clause containing the variable shrinks (since a false literal might as well not be there).

▶ **Lemma 12.** *Any move by* F *on a single tree results in a win-forest.*

▶ **Lemma 13.** T *can ensure that a win-forest remains a win-forest after an* F-T *round even if* F *can use pass moves.*

Before proving Lemma 12 and Lemma 13, we use them to prove Lemma 10.

Proof of Lemma 10. The tree ψ is a single tree. By Lemma 12, F's first move on ψ results in a win-forest. Then we prove T can win a $G^{+}_{3,F\ldots}$ game on that win-forest even if F can use pass moves. We prove this by induction on the number of variables.

Base case: The formula is a win-forest with one or two variables. In case of one variable the only possibility is an isolated variable with no clauses. T has already won in this case. In case of two variables there exists either two isolated variables where T has already won or a width-2 clause which T can satisfy in one move.

Induction step: The formula is a win-forest with at least three variables. Whatever F plays, T has a response to ensure the residual formula is again a win-forest by Lemma 13. By the induction hypothesis, T can win the rest of the game. ◀

Any move by T or F can occur in two different ways as illustrated in Figure 3. Specifically, Case 1 is a move on a non-spare variable, and Case 2 is a move on a spare variable.

Proof of Lemma 12. The formula is a single tree. If F's move is a pass move then that results in a win-forest with only one single tree. If F's move is an actual move then it creates some married trees in which one spouse is just a single variable (Case 1 with F) or only one married tree (Case 2 with F). Then that results in a win-forest with only married trees. ◀

Proof of Lemma 13. The argument will show that whatever F plays, whether a pass move or an actual move in a single tree or married tree, T has a response such that each component of the residual formula is again either a single tree or a married tree; therefore the residual formula is again a win-forest.

Suppose F played a pass move. T can play any remaining variable in the win-forest. If that variable is an isolated variable then it just removes the isolated variable. Otherwise it satisfies some clauses in a component by Case 1 or Case 2 with T. Consequently the component is broken down into some single trees and possibly one married tree (if the component was a married tree). This preserves the win-forest property.

Suppose F played in a single tree. Then by Lemma 12 the residual formula is a win-forest. Then T can pretend F just played a pass move on this win-forest, and T can respond as explained in the previous paragraph. This preserves the win-forest property.

Suppose F played in a married tree. F's move happened in one of the two single trees that got married. T can play the root of the other spouse (where F has not played) and satisfy the width-2 clause. This means the two single trees get separated by T's move and it also breaks T's single tree at the root by Case 1 with T. Furthermore, F's move in his single tree also preserves the win-forest property by Lemma 12. This preserves the win-forest property. ◀

3 Corollaries

In this section, we investigate corollaries for G_6^+ in Subsection 3.1, G_6 in Subsection 3.2, G_5^+ in Subsection 3.3, and G_5 in Subsection 3.4.

3.1 G_6^+

Our proof of Theorem 1 in Subsection 2.1 showed that $\mathrm{G}^+_{6,\mathrm{T}\cdots\mathrm{F}}$ is PSPACE-complete. Now we show that $\mathrm{G}^+_{6,\mathrm{F}\cdots\mathrm{F}}$, $\mathrm{G}^+_{6,\mathrm{T}\cdots\mathrm{T}}$, and $\mathrm{G}^+_{6,\mathrm{F}\cdots\mathrm{T}}$ are also PSPACE-complete.

▶ **Corollary 14.** $\mathrm{G}^+_{6,\mathrm{F}\cdots\mathrm{F}}$ *is* PSPACE*-complete.*

Proof. The reduction is 3-TQBF $\leqslant \mathrm{G}^+_{6,\mathrm{F}\cdots\mathrm{F}}$. The idea is similar to 3-TQBF $\leqslant \mathrm{G}^+_{6,\mathrm{T}\cdots\mathrm{F}}$ from the proof of Theorem 1 in Subsection 2.1. We introduce one more variable z to X and add z to the first four clauses of φ^+: A_{2n}, C_{12n}, C_{12n-2}, and C_{12n-4}, increasing their widths by one, from $2, 2, 3, 4$ to $3, 3, 4, 5$ respectively. So φ^+ is a 6-CNF.

Now the claim is that T has a winning strategy in the 3-TQBF game iff T has a winning strategy in the $\mathrm{G}^+_{6,\mathrm{F}\cdots\mathrm{F}}$ game (φ^+, X).

Suppose F has a winning strategy in the 3-TQBF game. Then F can play z as the first move. Then F wins by the same argument as in Subsection 2.1.

Suppose T has a winning strategy in the 3-TQBF game. If F plays z as the first move then T wins by the same argument as in Subsection 2.1. If F does not play z as the first move then T plays z and satisfies A_{2n}, C_{12n}, C_{12n-2}, and C_{12n-4}. Then T wins by Lemma 8 with $t = 12n - 4$. ◀

▶ **Corollary 15.** $\mathrm{G}^+_{6,\mathrm{T}\cdots\mathrm{T}}$ *is* PSPACE*-complete.*

Proof. The reduction is $\mathrm{G}^+_{6,\mathrm{T}\cdots\mathrm{F}} \leqslant \mathrm{G}^+_{6,\mathrm{T}\cdots\mathrm{T}}$. Suppose an instance of $\mathrm{G}^+_{6,\mathrm{T}\cdots\mathrm{F}}$ is (φ^+, X). We simply introduce a dummy variable z that does not appear in φ^+ and use $Y = X \cup \{z\}$. We claim that T has a winning strategy in the $\mathrm{G}^+_{6,\mathrm{T}\cdots\mathrm{F}}$ game (φ^+, X) iff T has a winning strategy in the $\mathrm{G}^+_{6,\mathrm{T}\cdots\mathrm{T}}$ game (φ^+, Y). We repeat an argument from [17] that shows this.

Suppose T has a winning strategy on (φ^+, X). We show T's winning strategy on (φ^+, Y). T can start by the same strategy as in (φ^+, X) and continue as long as F does not play z. If F never plays z, then T plays z at the end and wins as in (φ^+, X). If F plays z then T can respond by playing any remaining variable $x_i = 1$, then T resumes his strategy from (φ^+, X) until that strategy tells him to play x_i. At this time, T again picks any other remaining variable and assigns it 1. Then T again resumes his strategy from (φ^+, X). The game goes on like this in phases. At the end, T has played all the variables he would have played in the (φ^+, X) game and possibly one more. Since φ^+ is positive, it must still be satisfied when one of the variables is 1 instead of 0.

Suppose F has a winning strategy on (φ^+, X). Then F's winning strategy on (φ^+, Y) is analogous to T's strategy in the previous paragraph. ◀

▶ **Corollary 16.** $\mathrm{G}^+_{6,\mathrm{F}\cdots\mathrm{T}}$ *is* PSPACE*-complete.*

Proof. $\mathrm{G}^+_{6,\mathrm{F}\cdots\mathrm{F}}$ is PSPACE-complete by Corollary 14. The reduction is $\mathrm{G}^+_{6,\mathrm{F}\cdots\mathrm{F}} \leqslant \mathrm{G}^+_{6,\mathrm{F}\cdots\mathrm{T}}$. The technique is identical to Corollary 15. ◀

Therefore we found PSPACE-completeness of all patterns of G^+_6 games.

▶ **Corollary 17.** $\mathrm{G}^+_{6,\mathrm{T}\cdots\mathrm{F}}, \mathrm{G}^+_{6,\mathrm{F}\cdots\mathrm{F}}, \mathrm{G}^+_{6,\mathrm{T}\cdots\mathrm{T}}, \mathrm{G}^+_{6,\mathrm{F}\cdots\mathrm{T}}$ *remain* PSPACE*-complete even when every clause has exactly* 6 *variables.*

Proof. For any pattern $a \cdots b$ where $a, b \in \{\mathrm{T}, \mathrm{F}\}$, we reduce from $\mathrm{G}^+_{6,a\cdots b}$ to the restricted version where every clause has exactly 6 variables. We argue that any clause C with width < 6 can be resized to a set of width-6 clauses without changing the outcome. We introduce two variables x, x' and clause C is written as $(C \vee x) \wedge (C \vee x')$, thus increasing C's width by 1. Whichever player has a winning strategy in the original formula, they can follow the same strategy in the modified formula until the other player plays x or x' and then respond by playing the other. (Or, if the other player never plays x or x', then it does not matter which one the winning player plays as the 2nd-to-last move in the game.) So it is possible to increase any clause's width without changing the outcome. We can repeatedly do this process until all clauses have width exactly 6. This increases the size of the formula by at most a constant factor. ◀

3.2 G_6

We already know that $\mathrm{G}_{5,\mathrm{T}\cdots\mathrm{F}}$ and $\mathrm{G}_{5,\mathrm{F}\cdots\mathrm{F}}$ are PSPACE-complete [17]. But any completeness result for $\mathrm{G}_{5,\mathrm{T}\cdots\mathrm{T}}$ and $\mathrm{G}_{5,\mathrm{F}\cdots\mathrm{T}}$ is unknown. Not only that, but also the complexities of $\mathrm{G}_{6,\mathrm{T}\cdots\mathrm{T}}$ and $\mathrm{G}_{6,\mathrm{F}\cdots\mathrm{T}}$ were unknown. Due to Corollary 15 and Corollary 16 we now know that $\mathrm{G}_{6,\mathrm{T}\cdots\mathrm{T}}$ and $\mathrm{G}_{6,\mathrm{F}\cdots\mathrm{T}}$ are also PSPACE-complete.

3.3 G^+_5

Now we show that $\mathrm{G}^+_{5,\mathrm{T}\cdots\mathrm{F}}$, $\mathrm{G}^+_{5,\mathrm{F}\cdots\mathrm{F}}$, $\mathrm{G}^+_{5,\mathrm{T}\cdots\mathrm{T}}$, and $\mathrm{G}^+_{5,\mathrm{F}\cdots\mathrm{T}}$ are all NL-hard. Each of these results implies Theorem 2.

▶ **Corollary 18.** $\mathrm{G}^+_{5,\mathrm{T}\cdots\mathrm{F}}$ *is* NL*-hard.*

Proof. It is well-known that 2-SAT is NL-complete, and trivially 2-SAT $\leqslant$ 2-TQBF. The reduction is 2-TQBF $\leqslant \mathrm{G}^+_{5,\mathrm{T}\cdots\mathrm{F}}$. The technique is identical to 3-TQBF $\leqslant \mathrm{G}^+_{6,\mathrm{T}\cdots\mathrm{F}}$ in Theorem 1 where the widths of A_i, B_i, C_j, D_k were $5, 4, 5, 6$ respectively. Since each F_k is now a width-2 clause, D_k becomes a width-5 clause. Therefore φ^+ becomes a 5-CNF. ◀

▶ **Corollary 19.** $\mathrm{G}^+_{5,\mathrm{F}\cdots\mathrm{F}}$ *is* NL*-hard.*

Proof. The reduction is 2-TQBF $\leqslant \mathrm{G}^+_{5,\mathrm{F}\cdots\mathrm{F}}$. The technique is identical to Corollary 14. ◀

▶ **Corollary 20.** $\mathrm{G}^+_{5,\mathrm{T}\cdots\mathrm{T}}$ *is* NL*-hard.*

Proof. $\mathrm{G}^+_{5,\mathrm{T}\cdots\mathrm{F}}$ is NL-hard by Corollary 18. The reduction is $\mathrm{G}^+_{5,\mathrm{T}\cdots\mathrm{F}} \leqslant \mathrm{G}^+_{5,\mathrm{T}\cdots\mathrm{T}}$. The technique is identical to Corollary 15. ◀

Table 2 G_w^+ results.

$w \rightarrow$	2	3	4	5	6
$\mathrm{T}\cdots\mathrm{F}$	L [17]	L under restrictions [15]	Unknown	NL-hard [Corollary 18]	PSPACE-complete [Theorem 1]
$\mathrm{F}\cdots\mathrm{F}$				NL-hard [Corollary 19]	PSPACE-complete [Corollary 14]
$\mathrm{T}\cdots\mathrm{T}$				NL-hard [Corollary 20]	PSPACE-complete [Corollary 15]
$\mathrm{F}\cdots\mathrm{T}$				NL-hard [Corollary 21]	PSPACE-complete [Corollary 16]

▶ **Corollary 21.** $\mathrm{G}^+_{5,\mathrm{F}\ldots\mathrm{T}}$ *is* NL*-hard.*

Proof. $\mathrm{G}^+_{5,\mathrm{F}\ldots\mathrm{F}}$ is NL-hard by Corollary 19. The reduction is $\mathrm{G}^+_{5,\mathrm{F}\ldots\mathrm{F}} \leqslant \mathrm{G}^+_{5,\mathrm{F}\ldots\mathrm{T}}$. The technique is identical to Corollary 15. ◀

Therefore we found NL-hardness of all patterns of G_5^+ games. But any completeness result for any pattern still remains open.

▶ **Corollary 22.** $\mathrm{G}^+_{5,\mathrm{T}\ldots\mathrm{F}}, \mathrm{G}^+_{5,\mathrm{F}\ldots\mathrm{F}}, \mathrm{G}^+_{5,\mathrm{T}\ldots\mathrm{T}}, \mathrm{G}^+_{5,\mathrm{F}\ldots\mathrm{T}}$ *remain* NL*-hard even when every clause has exactly* 5 *variables.*

Proof. The technique is identical to Corollary 17. ◀

3.4 G_5

We already know that $\mathrm{G}_{5,\mathrm{T}\ldots\mathrm{F}}$ and $\mathrm{G}_{5,\mathrm{F}\ldots\mathrm{F}}$ are PSPACE-complete [17]. But nothing was known for $\mathrm{G}_{5,\mathrm{T}\ldots\mathrm{T}}$ and $\mathrm{G}_{5,\mathrm{F}\ldots\mathrm{T}}$. Due to Corollary 20 and Corollary 21 we now know that $\mathrm{G}_{5,\mathrm{T}\ldots\mathrm{T}}$ and $\mathrm{G}_{5,\mathrm{F}\ldots\mathrm{T}}$ are also NL-hard. But any completeness result for $\mathrm{G}_{5,\mathrm{T}\ldots\mathrm{T}}$ and $\mathrm{G}_{5,\mathrm{F}\ldots\mathrm{T}}$ still remains open.

4 Summary

In Table 2 we summarize the status of the complexity of G_w^+ for all widths w and all patterns. We conjecture that G_3^+ may be tractable, but the only known general upper bound is PSPACE. For G_5^+, it would be interesting to improve the NL-hardness to P-hardness. For G_4^+, any nontrivial result would be interesting (such as NL-hardness, or improving the PSPACE upper bound even under restrictions on the formula).

In Table 3 we summarize the status of the complexity of G_w for all widths w and all patterns. We conjecture that even G_3 might be tractable, but again the only known general upper bound is PSPACE. For $\mathrm{G}_{4,\mathrm{T}\ldots\mathrm{F}}$, $\mathrm{G}_{4,\mathrm{F}\ldots\mathrm{F}}$, $\mathrm{G}_{5,\mathrm{T}\ldots\mathrm{T}}$, and $\mathrm{G}_{5,\mathrm{F}\ldots\mathrm{T}}$, it would be interesting to improve the NL-hardness to P-hardness. For $\mathrm{G}_{4,\mathrm{T}\ldots\mathrm{T}}$ and $\mathrm{G}_{4,\mathrm{F}\ldots\mathrm{T}}$, any nontrivial result would be interesting.

It would also be interesting to see if Theorem 1 can be used to improve any parameters in some of the many PSPACE-completeness results that have been shown by reduction from Schaefer's theorem for width 11.

Table 3 G_w results.

$w \rightarrow$	2	3	4	5	6
T · · · F	L [17]	L under restrictions [18]	NL-hard	PSPACE-complete [17]	
F · · · F			NL-hard		
T · · · T			Unknown	NL-hard [Corollary 20]	PSPACE-complete [Corollary 15]
F · · · T				NL-hard [Corollary 21]	PSPACE-complete [Corollary 16]

References

1 Argimiro Arratia and Iain Stewart. A note on first-order projections and games. *Theoretical Computer Science*, 290(3):2085–2093, 2003.

2 Bengt Aspvall, Michael Plass, and Robert Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.

3 Boštjan Brešar, Paul Dorbec, Sandi Klavžar, Gašper Košmrlj, and Gabriel Renault. Complexity of the game domination problem. *Theoretical Computer Science*, 648:1–7, 2016.

4 Jesper Byskov. Maker-Maker and Maker-Breaker games are PSPACE-complete. Technical Report RS-04-14, BRICS, Department of Computer Science, Aarhus University, 2004.

5 Vasek Chvátal and Paul Erdös. Biased positional games. *Annals of Discrete Mathematics*, 2:221–229, 1978.

6 Eurinardo Costa, Victor Lage Pessoa, Rudini Menezes Sampaio, and Ronan Soares. PSPACE-hardness of two graph coloring games. In *Proceedings of the 10th Latin and American Algorithms, Graphs, and Optimization Symposium (LAGOS)*, pages 333–344. Elsevier, 2019.

7 Erik Demaine and Robert Hearn. Constraint logic: A uniform framework for modeling computation as games. In *Proceedings of the 23rd Conference on Computational Complexity (CCC)*, pages 149–162. IEEE, 2008.

8 Eric Duchene, Valentin Gledel, Aline Parreau, and Gabriel Renault. Maker–breaker domination game. *Discrete Mathematics*, 343(9):111955, 2020.

9 Paul Erdös and John Selfridge. On a combinatorial game. *Journal of Combinatorial Theory, Series A*, 14(3), 1973.

10 Stephen Fenner, Daniel Grier, Jochen Messner, Luke Schaeffer, and Thomas Thierauf. Game values and computational complexity: An analysis via black-white combinatorial games. In *Proceedings of the 26th International Symposium on Algorithms and Computation (ISAAC)*, pages 689–699. Springer, 2015.

11 Aviezri Fraenkel and Elisheva Goldschmidt. PSPACE-hardness of some combinatorial games. *Journal of Combinatorial Theory, Series A*, 46(1):21–38, 1987.

12 Valentin Gledel, Michael A Henning, Vesna Iršič, and Sandi Klavžar. Maker–breaker total domination game. *Discrete Applied Mathematics*, 282:96–107, 2020.

13 Robert Hearn. Amazons, Konane, and Cross Purposes are PSPACE-complete. In *Games of No Chance 3*, Mathematical Sciences Research Institute Publications, pages 287–306. Cambridge University Press, 2009.

14 Dan Hefetz, Michael Krivelevich, Miloš Stojaković, and Tibor Szabó. *Positional Games.* Birkhäuser Basel (Springer), 2014.

15 Martin Kutz. Weak positional games on hypergraphs of rank three. In *Proceedings of the 3rd European Conference on Combinatorics, Graph Theory, and Applications (EuroComb)*, pages 31–36. Discrete Mathematics & Theoretical Computer Science, 2005.

16 Alfred Lehman. A solution of the Shannon switching game. *Journal of the Society for Industrial and Applied Mathematics*, 12(4):687–725, 1964.

17 Md Lutfar Rahman and Thomas Watson. Complexity of unordered CNF games. *ACM Transactions on Computation Theory*, 12(3):18:1–18:18, 2020.

18 Md Lutfar Rahman and Thomas Watson. Tractable unordered 3-CNF games. In *Proceedings of the 14th Latin American Theoretical Informatics Symposium (LATIN)*. Springer, 2020. To appear.

19 Thomas Schaefer. Complexity of decision problems based on finite two-person perfect-information games. In *Proceedings of the 8th Symposium on Theory of Computing (STOC)*, pages 41–49. ACM, 1976.

20 Thomas Schaefer. On the complexity of some two-person perfect-information games. *Journal of Computer and System Sciences*, 16(2):185–225, 1978.

21 Wolfgang Slany. The complexity of graph Ramsey games. In *Proceedings of the 2nd International Conference on Computers and Games (CG)*, pages 186–203. Springer, 2000.

22 Wolfgang Slany. Endgame problems of Sim-like graph Ramsey avoidance games are PSPACE-complete. *Theoretical Computer Science*, 289(1):829–843, 2002.

23 Larry Stockmeyer and Albert Meyer. Word problems requiring exponential time. In *Proceedings of the 5th Symposium on Theory of Computing (STOC)*, pages 1–9. ACM, 1973.

24 Sachio Teramoto, Erik Demaine, and Ryuhei Uehara. The Voronoi game on graphs and its complexity. *Journal of Graph Algorithms and Applications*, 15(4):485–501, 2011.

25 Jan van Rijn and Jonathan Vis. Complexity and retrograde analysis of the game Dou Shou Qi. In *Proceedings of the 25th Benelux Conference on Artificial Intelligence (BNAIC)*, 2013.

Resolution with Symmetry Rule Applied to Linear Equations

Pascal Schweitzer
TU Kaiserslautern, Germany

Constantin Seebach
TU Kaiserslautern, Germany

Abstract

This paper considers the length of resolution proofs when using Krishnamurthy's classic symmetry rules. We show that inconsistent linear equation systems of bounded width over a fixed finite field $\mathbb{F}_p$ with p a prime have, in their standard encoding as CNFs, polynomial length resolutions when using the local symmetry rule (SRC-II).

As a consequence it follows that the multipede instances for the graph isomorphism problem encoded as CNF formula have polynomial length resolution proofs. This contrasts exponential lower bounds for individualization-refinement algorithms on these graphs.

For the Cai-Fürer-Immerman graphs, for which Torán showed exponential lower bounds for resolution proofs (SAT 2013), we also show that already the global symmetry rule (SRC-I) suffices to allow for polynomial length proofs.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases Logical Resolution, Symmetry Rule, Linear Equation System

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.58

Related Version *Full Version*: `https://arxiv.org/abs/2101.05142` [15]

Funding The research leading to these results has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (EngageS: grant agreement No. 820148).

1 Introduction

Refutation via logical resolution is one of the most basic and fundamental methods in theorem proving used to argue the validity of statements in propositional logic. It is famously sound and complete for proving that formulas in conjunctive normal form (CNF) are unsatisfiable. In automated theorem proving, resolution is in particular used for various primitive backtracking algorithms for the satisfiability problem (SAT) such as the DPLL algorithm.

However, resolution is primitive in that we know simple unsatisfiable CNF formulas that admit only resolution refutations of superpolynomial length. This was first proven by Haken [11] who showed that a canonical encoding of the pigeonhole principle into a CNF formula provides formulas whose shortest refutations are superpolynomial in length. Other examples and exponential bounds were given by Chvátal and Szemerédi [5] as well as Urquhart who used formulas based on Tseitin tautologies [19]. Investigating the resolution complexity of the graph non-isomorphism problem, Torán [17] constructed CNF formulas from so-called CFI-graphs (see [4]) and showed the shortest resolution proofs of the arising formulas have exponential length.

As observed by Krishnamurthy, many simple examples without short resolution refutations exhibit symmetries. This prompted the introduction of Krishnamurthy's symmetry rule [12] which intuitively allows the deduction of a clause symmetric to a previously deduced

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 58; pp. 58:1–58:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

clause in one step (formal definitions are given in Section 2). For various formulas, Krishnamurthy argued polynomial bounds when the symmetry-rule is used, leading to exponential improvements. Further examples with this effect, including another analysis for pigeonhole principle formulas, were provided by Urquhart [20].

Krishnamurthy in fact introduced two rules, each of them arises from permutations of the variables. The *global* rule allows only symmetries of the entire original formula, while the *local* one allows us to use symmetries of a subset of the clauses. These rules led to the proof systems SR-I (*symmetric resolution*) and SR-II (*locally symmetric resolution*), respectively. Urquhart [20] introduced *complementation symmetries* in addition to the variable permutations. This allows us to interchange literals with their negations and leads to the proof systems SRC-I and SRC-II. In [20] Urquhart also showed that there are exponential-to-polynomial improvements regarding proof length from the system SR-I to SRC-I. Arai and Urquhart [1] showed exponential-to-polynomial improvements from SR-I to SR-II and also provided exponential lower bounds for SRC-II.

Szeider [16], who actually focuses on homomorphisms, describes another strengthening of the symmetry rule. In his extension we are allowed the use of symmetries within clauses that have been resolved, rather than only allowing clauses of the original formula. This is called resolution with *dynamic* symmetries and leads to the proof systems SR-III and SRC-III, depending on whether complementation is allowed. However, to date it remains an open problem to find superpolynomial lower bounds on proof length in SR-III and SRC-III.

1.1 Contribution

In this paper we are concerned with proof systems obtained by extending resolution with additional symmetry rules. We prove that the CNF formulas arising from the CFI-graphs have refutations polynomially bounded in length in the SR-I calculus. With Torán's exponential lower bounds [17] mentioned above, this gives an exponential-to-polynomial improvement for the resolution complexity of non-isomorphism when introducing the symmetry rule. To those familiar with the details of the CFI-construction this may not come as a surprise, since the CFI-graphs exhibit many global symmetries. However, this is not the case for multipede graphs, these arise from a construction related to the CFI-graphs [10]. Crucially these graphs are asymmetric. That is, they have no symmetries at all. They provide exponential lower bounds for all individualization-refinement algorithms for the graph isomorphism problem. This includes all tools currently viable in practice, such as nauty/traces [13]. The initial intuition might therefore be that the CNF formulas arising from multipedes provide exponential lower bounds for SRC-III. However, this turns out not to be the case. In fact, maybe surprisingly, we show that even when using only local symmetries rather than dynamic symmetries (i.e., in SCR-II rather than SCR-III) there are polynomial bounds on the respective formulas. In some sense this shows that the multipedes have substructures with symmetries that allow them to be distinguished concisely.

To prove this statement, we reduce the statement to one concerning linear equation systems. It is known that isomorphism of CFI and multipede graphs are related to solvability of linear equation systems. (This is also the case for Tseitin tautologies.) We show that this relation can be exploited. Specifically, we show that there is a resolution transforming the CNFs arising from the graph isomorphism instances to CNFs arising from linear equation systems. We then show our main theorem which says that inconsistent linear equation systems with equations of bounded width (i.e., the maximum number of non-zero coefficients in an equation is bounded) have polynomial resolutions using the local symmetry rule.

	none	global	local	dynamic
without complementation	classical resol.[5, 19]	SR-I [20]	SR-II [1]	SR-III (open)
with complementation	–	SRC-I [20]	SRC-II [1]	SRC-III (open)

Figure 1 Resolution calculi with symmetries rules of varying degree of generality and references with formulas proving exponential lower bounds on resolution length.

▶ **Theorem 1.** *Inconsistent linear equation systems of bounded width over a fixed finite field* $\mathbb{F}_p$ *with* p *a prime have, in their standard encoding as CNFs, polynomial length resolutions when using the local symmetry rule (i.e., in SRC-II).*

Structure of the paper. Section 3 shows that the CNF formulas arising from CFI-graph pairs have polynomial length proofs in SR-I. Due to space restrictions, the proof of this result (Theorem 18) was omitted from this version of the paper. It can be found in the full version [15]. Section 4 shows that linear equation systems of bounded width have polynomial length proofs in SRC-II. Section 5 shows that the formulas arising from (bounded degree) multipede graphs can be transformed in the resolution calculus (without using symmetry) to linear equation systems of bounded width.

1.2 Related Work

Figure 1 gives an overview of resolution calculi with symmetry and references to lower bound constructions. A proof system *p-simulates* another proof system if shortest proofs in the latter are polynomially bounded in the length of shortest proofs in the former. We should remark that the extended resolution system introduced by Tseitin [18] can p-simulate proof systems with symmetries [20]. See [2] for an implementation using Krishnamurthy's symmetry rule. Symmetry rules have of course also been introduced for other proof systems [3, 8]. See also [7] for another way to incorporate symmetries into resolution.

Connection to the graph isomorphism problem. The results of our paper are connected to the graph isomorphism problem in two conceptually very different ways. First, finding valid literal permutations (with or without complementation) for the global symmetry rule is equivalent to the graph isomorphism problem itself (e.g., [4]). Therefore isomorphism solvers such as nauty/traces [13], which are highly efficient in practice, can be used to find the symmetries (see [6]). Symmetry detection is one of the standard applications of graph isomorphism solvers, for example there is a tool integrating nauty into Prolog [9] for this purpose.

Second, our results relate to the proof complexity of the graph isomorphism problem itself, which explains why we are interested in CNF formulas arising from non-isomorphism instances. Torán [17] describes a canonical way to encode the isomorphism problem as a CNF formula (see Subsection 2.2). The resolution complexity of graph non-isomorphism is related to the complexity of the graph isomorphism problem. After all isomorphism solvers need to prove, some way or another, that the inputs are non-isomorphic, if they are. A crucial feature of isomorphism solvers is that they are able to exploit already detected symmetries (i.e., automorphisms) of the underlying instances during run-time [13]. Vaguely, this translates into a symmetry rule that they apply already during the process of computing the symmetries of the instance. Current tools basically only exploit local symmetries. Our new insights into the resolution complexity of multipedes thus shows a combinatorial possibility to solve their isomorphism problem. It brings up the question how to exploit local symmetries in graph isomorphism solvers.

It remains unknown whether graph non-isomorphism has polynomial resolution complexity in any of the proof systems with symmetry rule we have discussed.

2 Preliminaries

2.1 Resolution and the Symmetry Rule

We are interested in unsatisfiability proofs of Boolean formulas. The basic resolution proof system works with formulas in conjunctive normal form.

Let Γ be a finite set of variables. $\mathrm{Lit}(\Gamma) := \Gamma \cup \overline{\Gamma}$ is the set of literals, where $\overline{\Gamma} := \{\overline{x} \mid x \in \Gamma\}$. A *clause* is a disjunction of literals. We also represent clauses as sets of literals. A Boolean formula is in *conjunctive normal form (CNF)* if it is a conjunction of clauses. We may treat such a formula as a set of clauses. $\bot$ is the empty clause, i.e. the disjunction of the empty set, which is unsatisfiable. For sets of clauses C_1 and C_2 define $C_1 \sqsubseteq C_2 :\iff \forall c_1 \in C_1 \exists c_2 \in C_2 : c_1 \supseteq c_2$. Since we will treat clauses as sets of literals, we do not care for their order, i.e. we do not differentiate between $x \vee y$ and $y \vee x$. The same applies to CNF formulas, which we interpret as sets of clauses.

▶ **Definition 2.** Resolution *is a proof system in propositional logic. It operates on CNF formulas, employing a single inference rule:*

$$\frac{x \vee A,\ \overline{x} \vee B}{A \vee B}.$$

The clause produced by the resolution rule is called resolvent*.*

Let $A = \{a_1, \dots, a_m\}$ and B be sets of clauses. We write $A \vdash_n B$ if there exists a sequence of clauses $a_1, \dots, a_m, c_1, \dots, c_n$ such that every c_i is a resolvent of two earlier clauses and $B \subseteq A \cup \{c_1, \dots, c_n\}$. Such a sequence is called derivation *of B from A. When the length of the sequence is irrelevant, we write $A \vdash B$, meaning $A \vdash_n B$ for some n. Given a clause b, we also write $A \vdash_n b$ for $A \vdash_n \{b\}$.*

For a CNF formula F with $F \vdash_n \bot$, we say F has a resolution refutation *of size n.*

We write $A \vdash_n^w B$ if there exists a set of clauses B' such that $B \sqsubseteq B'$ and $A \vdash_n B'$. This is a weaker requirement than $A \vdash_n B$.

Resolution is sound and complete, i.e. $F \vdash_n \bot$ if and only if F is unsatisfiable. We examine the proof complexity of formulas in this proof system, i.e., the length of the shortest possible resolution refutation of a given formula, in relation to the formula size. There exist classes of formulas with exponential lower bounds on the resolution proof complexity [5, 17, 19].

In the following we define the symmetry rule, which is an extension to resolution, aiming to reduce the proof complexity of some of these hard formulas.

▶ **Definition 3.** *Let L be a finite set of literals. A bijection $\sigma : L \to L$ is called* renaming *if for every $\ell \in L$ we have $\overline{\sigma(\ell)} = \sigma(\overline{\ell})$.*

A renaming is essentially a permutation of the variables that may also negate some of them. We can apply renamings to clauses (i.e., sets of literals) and CNF formulas (i.e., sets of clauses). In either case we define $\sigma(C) := \{\sigma(x) \mid x \in C\}$.

▶ **Definition 4** (The Symmetry Rule [1] [16]). *Consider a derivation S from a formula F and a subsequence S' of S which derives a clause C from a subset $F' \subseteq F$. If there exists a renaming σ with $\sigma(F') \subseteq F$, then the* local symmetry rule *allows derivation of $\sigma(C)$.*

With the restriction $F = F'$, we obtain the global symmetry rule*. Adding the global or local symmetry rule to the resolution system yields the proof systems* ***SRC-I*** *and* ***SRC-II****, respectively.*

We write $A \vdash_n^{\text{SRC-II}} B$ to indicate that B can be derived from A using resolution and the local symmetry rule, with a derivation of length at most n.

Note that in order to apply σ via the local symmetry rule to some clause C in a derivation, we must look at the entire history of how C was derived, and find out which part $F' \subseteq F$ of the original formula was used. Then we need to check that $\sigma(F') \subseteq F$.

This means that in general we cannot chain derivations that use the symmetry rule together, because such an operation changes the history for some of the clauses. Still, we can combine **SRC-II** derivations in the following ways:

▶ **Lemma 5.** *Let A, B, C and D be sets of clauses and $n, m \in \mathbb{N}$.*
(a) $A \vdash_n^{SRC\text{-}II} B$ *and* $A \subseteq C$ *implies* $C \vdash_n^{SRC\text{-}II} B$
(b) $A \vdash_n^{SRC\text{-}II} B$ *and* $B \vdash_m C$ *implies* $A \vdash_{n+m}^{SRC\text{-}II} C$
(c) $A \vdash_n^{SRC\text{-}II} B$ *and* $C \vdash_m^{SRC\text{-}II} D$ *implies* $A \cup C \vdash_{n+m}^{SRC\text{-}II} B \cup D$

▶ **Lemma 6.** *Let A and B be sets of clauses and d a clause.*
(a) $A \vdash \bot$ *and* $\{c \vee d \mid c \in A\} \sqsubseteq B$ *implies* $B \vdash^w d$

2.2 Encoding Graph Isomorphism

Our interest in the graph isomorphism problem is twofold: First, finding valid literal permutations for the symmetry rule is equivalent to finding certain graph isomorphisms. Secondly, we examine the proof complexity of the problem by translating it into propositional logic and applying resolution with symmetry rule.

A *graph* is a tuple (V, E) of a set of vertices V and edges E. Each edge is a two element subset of V. A *colored graph* is a graph (V, E) together with a function $f : V \to C$, called *coloring*, assigning to every vertex a color from some set C. Let $G = (V, E)$ be a graph and $v \in V$. $\mathcal{E}_G(v) := \{e \in E \mid v \in e\}$ are the edges *incident* with v. $N_G(v) := \{u \in V \mid \{u, v\} \in E\}$ is the *neighborhood* of v. $\deg_G(v) := |N_G(v)| = |\mathcal{E}_G(v)|$ is the *degree* of v.

Given a colored graph $G = (V, E)$ with coloring f and a vertex $v \in V$, we can *individualize* v by creating a new coloring f' such that $f'(v) := (f(v), 1)$ and setting $f'(v') := (f(v'), 0)$ for all $v' \in V \setminus \{v\}$. We write the individualized graph as G_v.

▶ **Definition 7.** *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be graphs.*

- *A* graph isomorphism *from G_1 to G_2 is a bijection $\varphi : V_1 \to V_2$ such that for all $v, v' \in V_1$ we have $\{v, v'\} \in E_1$ if and only if $\{\varphi(v), \varphi(v')\} \in E_2$.*
- *We say G_1 and G_2 are* isomorphic*, written $G_1 \cong G_2$, if there exists a graph isomorphism from G_1 to G_2.*
- *An* automorphism *of a graph G is a graph isomorphism from G to itself.*
- $\mathrm{Aut}(G)$ *is the* automorphism group *of G.*

The automorphisms of a graph constitute its inherent combinatorial symmetries. We will use the terms automorphism and symmetry synonymously.

Given two graphs G_1 and G_2, one can construct a Boolean formula that is satisfiable if and only if there is an isomorphism between G_1 and G_2 [17]. This is commonly done by constraining variables of the form $x_{u,v}$ such that each satisfying assignment corresponds to an isomorphism: 1 is assigned to $x_{u,v}$ if and only if the isomorphism maps u to v.

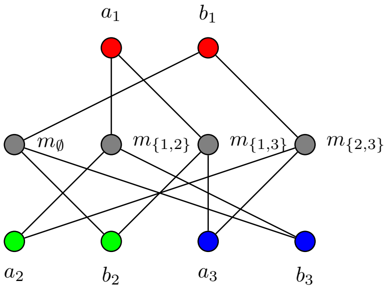


Figure 2 The CFI-gadget $X_{\{1,2,3\}}$.

▶ **Definition 8.** *For a pair of graphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ with $|V_1| = |V_2|$, define*

$$F(G_1, G_2) := T_1 \wedge T_2 \wedge T_3, \text{ where}$$
$$T_1 := \bigwedge_{v_1 \in V_1} \bigvee_{v_2 \in V_2} x_{v_1,v_2},$$
$$T_2 := \bigwedge_{v_2 \in V_2} \bigwedge_{\substack{v_1, v_1' \in V_1 \\ v_1 \neq v_1'}} \left(\overline{x_{v_1,v_2}} \vee \overline{x_{v_1',v_2}}\right),$$
$$T_3 := \bigwedge_{\{u_1,v_1\} \in E_1 \oplus \{u_2,v_2\} \in E_2} \left(\overline{x_{u_1,u_2}} \vee \overline{x_{v_1,v_2}}\right).$$

We refer to the clauses of this CNF formula as being *"of Type i"*, depending on which T_i they come from. The clause types naturally encode the concept of a graph isomorphism in propositional logic. Specifically, Type 1 and Type 2 clauses ensure that we have a bijection from V_1 to V_2; Type 3 clauses make the function preserve edges.

If the graphs G_1 and G_2 are colored by some functions l_1 and l_2 respectively, then an isomorphism between them should respect the colors. To represent this in the formula $F(G_1, G_2)$, we simply assign 0 to all variables $x_{u,v}$ for which $l_1(u) \neq l_2(v)$.

2.3 The CFI Graphs

In this section, we look at the graphs by Cai, Fürer and Immerman [4], which were constructed to prove lower bounds for the Weisfeiler-Lehman method in isomorphism testing. These graphs are also challenging when we use resolution to decide isomorphism. They are built from gadget graphs which are defined as follows (see Figure 2).

▶ **Definition 9** (CFI-gadget [4, 6]). *Given a finite set N, define: $X_N := (V, E, \gamma)$, where $V := A \cup B \cup M$ consists of $A := \{a_w \mid w \in N\}$, $B := \{b_w \mid w \in N\}$ as well as $M := \{m_S \mid S \subseteq N, |S| \text{ even}\}$ and $E := \{\{m_S, a_w\} \mid w \in S\} \cup \{\{m_S, b_w\} \mid w \in N \setminus S\}$. Also define the coloring* $\gamma : V \to C : v \mapsto \gamma(v) := \begin{cases} c_w & \text{if } v \in A \cup B \text{ with } v = a_w \text{ or } v = b_w, \\ m & \text{if } v \in M. \end{cases}$

The most important feature of the CFI-gadgets are their automorphisms:

▶ **Lemma 10** ([4, 6.1]). *There are $2^{|N|-1}$ automorphisms of X_N. Each is uniquely determined by interchanging the vertices a_w and b_w for all w in some subset $S \subseteq N$ of even cardinality.*

▶ **Definition 11** (CFI graph). *From a graph $G = (V, E)$ construct $X(G)$ by connecting the CFI gadgets $\{X^v_{\mathcal{E}_G(v)} \mid v \in V\}$ with edges $E' := \{\{a^u_e, a^v_e\} \mid e = \{u, v\} \in E\} \cup \{\{b^u_e, b^v_e\} \mid e = \{u, v\} \in E\}$.*

▶ **Definition 12.** *Given a graph $G = (V, E)$ with $E \neq \emptyset$, construct $\tilde{X}(G)$ from $X(G)$ by choosing some edge $e = \{u, v\} \in E$ and replacing the edges $\{a_e^u, a_e^v\}, \{b_e^u, b_e^v\}$ with the edges $\{a_e^u, b_e^v\}, \{b_e^u, a_e^v\}$. We say that the edges corresponding to e have been twisted.*

Note that Definition 12 does not specify how to choose the edge which is to be twisted, so there are in fact multiple graphs that we could call $\tilde{X}(G)$. If G is connected however, these graphs are isomorphic. On the other hand, for any graph G with at least one edge, $\tilde{X}(G)$ and $X(G)$ are not isomorphic [[4, see Lemma 6.2]]. The CFI graphs have been used to prove the following lower bound for resolution:

▶ **Theorem 13** ([17, Corollary 5.2])**.** *There exists a family of graphs $\mathcal{G} = (G_n)_{n\in\mathbb{N}}$ such that for every n, G_n has n vertices and the resolution refutation of the formula $F(X(G_n), \tilde{X}(G_n))$ requires size $\exp(\Omega(n))$. The graphs $X(G_n)$ and $\tilde{X}(G_n)$ have color multiplicity at most 4.*

This exponential lower bound motivates the use of a more efficient proof system to prove non-isomorphism of CFI graphs. Because of the symmetric nature of the CFI-gadgets, the symmetry rule is expected to reduce the proof length significantly. With symmetry rule, short proofs exist, as we show in Section 3.

In order to obtain examples for which the symmetry rule is not able to produce short proofs, it is a natural idea to consider asymmetric graphs instead.

2.4 Multipede Graphs

In [10], the so-called Multipedes were defined - a method to construct asymmetric structures. Combining this construction with CFI-gadgets, one obtains a family of asymmetric graphs which provide exponential lower bounds for individualization-refinement algorithms [14].

▶ **Definition 14** (Multipede graph [14])**.** *From a bipartite graph $G = (V, W, E)$, we construct the* Multipede graph $MP(G)$ *as follows: For every $w \in W$ create a pair of vertices a_w, b_w, colored with c_w. We call these pairs* feet*. Then for every $v \in V$ take a CFI-gadget $X^v_{N_G(v)}$ and identify the vertices a^v_w and b^v_w with a_w and b_w respectively.*

▶ **Theorem 15** ([14])**.** *There exists a family of bipartite graphs $\mathcal{G} = (G_n)_{n\in\mathbb{N}}$ such that for each n the graph G_n has $\mathcal{O}(n)$ vertices, $MP(G_n)$ is asymmetric and individualization-refinement algorithms take $\exp(\Omega(n))$ steps to verify $MP(G_n)_{a_w} \ncong MP(G_n)_{b_w}$.*

The Multipede graphs are of particular interest, because they are a generalization of the CFI graphs, and thus also hard for resolution, and additionally they can be constructed to be asymmetric. Hence the global symmetry rule is insufficient to get short proofs concerning Multipedes. However, as we will prove, the local symmetry of the CFI-gadgets can be used by the local symmetry rule.

The automorphism group of a Multipede graph is closely related to the solution set of a linear equation system. As a consequence of Lemma 10, any automorphism φ of a Multipede can be uniquely specified by the set of feet $Y := \{w \in W \mid \varphi(a_w) = b_w\}$ for which the a-b-pairs are swapped. The set Y represents a valid automorphism exactly if for every CFI-gadget in the graph, an even number of incident feet is swapped.

Using linear algebra, we can encode a subset $Y \subseteq W = \{w_1, \dots, w_n\}$ uniquely as a vector $\mathbf{y} \in \mathbb{F}_2^n$, by setting $\mathbf{y}_i = 1$ if and only if $w_i \in Y$ for all i. Then the evenness-condition, which the CFI-gadgets require, can be expressed as a set of linear equations:

$$\text{for all } v \in V\colon \quad \sum_{w_i \in N_G(v)} \mathbf{y}_i = 0\,.$$

We can write the equations in matrix form: Let $G = (\{v_1, \ldots, v_m\}, \{w_1, \ldots, w_n\}, E)$ be a bipartite graph. Define $\mathbf{M}(G) \in \mathbb{F}_2^{m \times n}$ as follows: $\mathbf{M}(G)_{i,j} := \begin{cases} 1 & \text{if } \{v_i, w_j\} \in E \\ 0 & \text{otherwise.} \end{cases}$

The solutions of the linear equation system $\mathbf{M}(G)\mathbf{y} = \mathbf{0}$ correspond to the automorphisms of $MP(G)$. We will show how to apply resolution and the symmetry rule to linear equations, and extend our results to Multipedes.

2.5 Encoding Linear Equations

Linear equations over finite fields have been used to show lower bounds in Proof Complexity. For example, the Tseitin formulas are constructed from graphs, representing a system of linear equations over $\mathbb{F}_2$, and are hard for resolution [19]. In the next section we will show that by adding the symmetry rule to resolution, we get short proofs for linear equations.

To work with linear equations, some basic definitions and notations from linear algebra are needed. Let K be a field and $n, m \in \mathbb{N}$. We write $K^{m \times n}$ for the set of all m by n matrices over K. Symbols for matrices will be written in boldface. Given a matrix $\mathbf{A} \in K^{m \times n}$ and numbers $i \in [1, m]$ and $j \in [1, n]$, we write $\mathbf{A}_{i,j}$ for the element at the i-th row and j-th column of $\mathbf{A}$. We write K^n for the set of all n-element vectors. For our purposes they can be treated like single-column matrices, i.e., $K^n = K^{n \times 1}$.

We write $\mathbf{0}$ for a vector consisting of zeros, where its size is clear from context. Similarly $\mathbf{1}$ is a vector filled with ones.

Applying resolution to a linear equation system means providing a refutation certifying that the system cannot be solved, if that is indeed the case. The following lemma is essential in proving an equation system unsolvable:

▶ **Lemma 16.** *Let $\mathbf{A} \in \mathbb{F}_p^{m \times n}$ and $\mathbf{b} \in \mathbb{F}_p^m$. If the equation system $\mathbf{Ax} = \mathbf{b}$ does not have a solution $\mathbf{x} \in \mathbb{F}_p^n$, then there exists some $\mathbf{v} \in \mathbb{F}_p^m$ such that $\mathbf{vA} = \mathbf{0}$ and $\mathbf{v} \cdot \mathbf{b} = 1$.*

Proof. Since the equation system does not have a solution, applying the Gaussian elimination algorithm yields the equation $0 = 1$. Writing the row operations used by the algorithm as a vector, we get the sought-after $\mathbf{v}$. ◀

We require some notation for standard operations from linear algebra.

Let $\mathbf{r} = (r_1, \ldots, r_n) \in K^n$ and $\mathbf{A} \in K^{m \times n}$. $\operatorname{supp}(\mathbf{r}) := \{i \in \{1, \ldots, n\} \mid r_i \neq 0\}$ is the *support* of $\mathbf{r}$. $\mathbf{A}_{i,*} := (\mathbf{A}_{i,1}, \ldots, \mathbf{A}_{i,n}) \in K^n$ is the i-th row of $\mathbf{A}$. $\operatorname{diag}(\mathbf{r}) \in K^{n \times n}$ is the diagonal matrix with diagonal entries equal to $\mathbf{r}$. $\mathbf{\Sigma A} := \sum_{i=1}^m \mathbf{A}_{i,*} = \mathbf{1} \cdot \mathbf{A}$ is the row sum of $\mathbf{A}$. Let $\mathbf{v} = (v_1, \ldots, v_n) \in K^n$. Then $\mathbf{r}|_\mathbf{v} \in K^n$ is the *restriction* of $\mathbf{r}$ to the support of $\mathbf{v}$, defined by $(\mathbf{r}|_\mathbf{v})_i := \begin{cases} r_i & \text{if } v_i \neq 0 \\ 0 & \text{if } v_i = 0 \end{cases}$ for $i \in \{1, \ldots, n\}$.

To encode linear equations as CNF formulas, we first introduce variables which correspond to the solution vector of the linear equation system: $\text{Vars} := \{\xi_{i,k} \mid i \in [1, n] \text{ and } k \in \mathbb{F}_p\}$. For a given vector $\mathbf{x} \in \mathbb{F}_p^n$, the corresponding assignment to the variables would set $\xi_{i,k}$ to *true* if and only if $\mathbf{x}_i = k$.

Our CNF formula has a clause for every $\mathbf{x}$ with $\mathbf{Ax} \neq \mathbf{b}$, ensuring the forumla is *false* under the assignment corresponding to $\mathbf{x}$. For every row $(\mathbf{a}, b)$ of the equation system, we consider all $\mathbf{x}$ with $\mathbf{a} \cdot \mathbf{x} \neq b$. We can restrict $\mathbf{x}$ to the components for which $\mathbf{a}$ is nonzero.

$$P(\mathbf{a}, b) := \{\mathbf{x} \in \mathbb{F}_p^n \mid \mathbf{a} \cdot \mathbf{x} \neq b \text{ and } \operatorname{supp}(\mathbf{x}) \subseteq \operatorname{supp}(\mathbf{a})\}.$$

The formula for the row $(\mathbf{a}, b)$ is then defined as follows:

$$F(\mathbf{a}, b) := \bigwedge_{\mathbf{x} \in P(\mathbf{a}, b)} C_{\mathbf{a}}(\mathbf{x}), \text{ where } C_{\mathbf{a}}(\mathbf{x}) := \bigvee_{i \in \operatorname{supp}(\mathbf{a})} \overline{\xi_{i,\mathbf{x}_i}}.$$

We extend this definition to whole systems of equations: $F(\mathbf{A}, \mathbf{b}) := \bigwedge_{i=1}^{m} F(\mathbf{A}_{i,*}, \mathbf{b}_i)$. Notice that assigning *false* to every variable satisfies $F(\mathbf{A}, \mathbf{b})$, but this assignment does not represent a vector. For this reason, we additionally need the clauses $V := \bigwedge_{i=1}^{n} \bigvee_{k \in \mathbb{F}_p} \xi_{i,k}$.

▶ **Lemma 17.** *Let $\mathbf{A} \in \mathbb{F}_p^{m \times n}$ and $\mathbf{b} \in \mathbb{F}_p^m$. There exists an $\mathbf{x} \in \mathbb{F}_p^n$ with $\mathbf{A}\mathbf{x} = \mathbf{b}$ if and only if $F(\mathbf{A}, \mathbf{b}) \wedge V$ is satisfiable.*

Proof. $\implies$: Assume $\mathbf{A}\mathbf{x} = \mathbf{b}$. Define an assignment $\varphi : \text{Vars} \to \mathbb{B}$ such that $\varphi(\xi_{i,k}) = 1$ if and only if $\mathbf{x}_i = k$. It is easy to see that $\varphi(V) = 1$. Let $j \in [1, m]$ and $\mathbf{x}' \in P(\mathbf{A}_{j,*}, \mathbf{b}_j)$. Then $\mathbf{A}_{j,*} \cdot \mathbf{x}' \neq \mathbf{b}_j = \mathbf{A}_{j,*} \cdot \mathbf{x}$. Hence there exists $i \in \operatorname{supp}(\mathbf{A}_{j,*})$ such that $\mathbf{x}_i \neq \mathbf{x}'_i$. Therefore $\varphi(\xi_{i,\mathbf{x}'_i}) = 0$, so $\varphi(C_{\mathbf{A}_{j,*}}(\mathbf{x}')) = 1$. Then $\varphi(F(\mathbf{A}_{j,*}, \mathbf{b}_j)) = 1$ and thus $F(\mathbf{A}, \mathbf{b})$ is satisfied by φ.

$\impliedby$: Assume that we have an assignment φ with $\varphi(F(\mathbf{A}, \mathbf{b})) = 1$ and $\varphi(V) = 1$. For all $i \in [1, n]$ there exists a $k \in \mathbb{F}_p$ such that $\varphi(\xi_{i,k}) = 1$. Define $\mathbf{x}_i := k$. Towards a contradiction, assume there exists $j \in [1, m]$ with $\mathbf{b}_j \neq \mathbf{A}_{j,*} \cdot \mathbf{x}$. Then $\mathbf{x}|_{\mathbf{A}_{j,*}} \in P(\mathbf{A}_{j,*}, \mathbf{b}_j)$ and $\varphi(C_{\mathbf{A}_{j,*}}(\mathbf{x})) = 1$. Hence there must exist an $i \in \operatorname{supp}(\mathbf{A}_{j,*})$ such that $\varphi(\xi_{i,\mathbf{x}_i}) = 0$, which contradicts our construction of $\mathbf{x}$. Therefore $\mathbf{A}\mathbf{x} = \mathbf{b}$. ◀

3 Linear-sized Refutations for Non-Isomorphism of CFI graphs

Due to the symmetric nature of the CFI graphs, using the symmetry rule gives us linear-sized resolution proofs of non-isomorphism for a pair of these graphs.

▶ **Theorem 18.** *Let G be a graph with at least one edge. Then*

$$F(X(G), \tilde{X}(G)) \vdash^{\mathit{SRC\text{-}I}}_{\mathcal{O}(|F(X(G), \tilde{X}(G))|)} \bot.$$

For a full proof of Theorem 18, see [15]. Here we only sketch the main ideas of the proof.

Proof sketch. The resolution refutation is created recursively. We remove a special vertex or edge from G to obtain a smaller graph G', get a short proof for $F(X(G'), \tilde{X}(G')) \vdash^{\text{SRC-I}} \bot$, and then use this to build a short proof for $F(X(G), \tilde{X}(G)) \vdash^{\text{SRC-I}} \bot$.

One of the following cases surely holds for G: It has an edge e on a cycle, or it has a vertex v of degree 1. In the latter case, there is only one way to map the CFI-gadgets of $X(G)$ and $\tilde{X}(G)$ around v to each other. This can be proved using standard resolution in a constant number of steps.

In the case of a cycle, we have a certain symmetry in $X(G)$, which the symmetry rule can exploit: Twisting every edge along the cycle is an automorphism. Hence, if $X(G)$ and $\tilde{X}(G)$ are isomorphic, then also this twisted version of $X(G)$ is isomorphic to $\tilde{X}(G)$. To show non-isomorphism, it is then sufficient to prove non-isomorphism for one of these versions, and the rest follows by symmetry - a symmetry which can be written in SRC-I. Choosing one of the versions is equivalent to having a fixed mapping of e.

In both cases, we fix the mapping of a vertex or an edge, which makes it possible to build up the resolution refutation from recursively smaller proofs. Each time we only add a constant number of steps, in sum yielding a proof of linear size. ◀

This result stands in contrast to the exponential lower bound of Theorem 13.

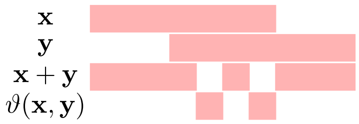


Figure 3 A visualization of $\vartheta(\mathbf{x},\mathbf{y})$.

4 Polynomial-sized Refutations for Linear Equations

4.1 Linear Combinations

The usual approach to showing that a system of linear equations is inconsistent, is to build a linear combination of the equations to derive the obvious contradiction $0 = 1$. This method is complete by Lemma 16. We recreate this process in the resolution proof system. However we need to ensure that the support of equations we create along the way is not excessively large. We will do so by using the the symmetry rule.

Note that the formula $F(\mathbf{a}, b)$ is invariant under linear scaling of the inputs: For any $k \in \mathbb{F}_p \setminus \{0\}$ we have $\operatorname{supp}(\mathbf{a}) = \operatorname{supp}(k\mathbf{a})$ and $P(\mathbf{a}, b) = P(k\mathbf{a}, kb)$. Hence $F(\mathbf{a}, b) = F(k\mathbf{a}, kb)$.

For the computation of linear combinations, we use the following definition. For $\theta \subseteq [1, n]$ define $\Omega(\theta) := \left\{\bigvee_{i\in\theta} \overline{\xi_{i,\mathbf{x}_i}} \mid \mathbf{x} \in \mathbb{F}_p^n \text{ with } \operatorname{supp}(\mathbf{x}) \subseteq \theta\right\}$. Together, the clauses in $\Omega(\theta)$ forbid all possible assignments to the components in range θ. In a sense, $\Omega(\theta)$ is our basic building block for contradictions.

▶ **Lemma 19.** *Let $\theta \subseteq [1, n]$. Then $\Omega(\theta) \wedge V \vdash_{\frac{p^{|\theta|+1}-p}{p-1}} \bot$.*

Proof. Induction over $|\theta|$. If $\theta = \emptyset$ then $\Omega(\theta) = \{\bigvee \emptyset\} = \{\bot\}$, so $\Omega(\theta) \vdash_0 \bot$.

Induction step: $\theta = \theta' \cup \{j\}$. It holds:

$$\begin{aligned}
\Omega(\theta) &= \left\{\bigvee_{i\in\theta} \overline{\xi_{i,\mathbf{x}_i}} \mid \mathbf{x} \in \mathbb{F}_p^n \text{ with } \operatorname{supp}(\mathbf{x}) \subseteq \theta\right\} \\
&= \left\{\overline{\xi_{j,\mathbf{x}_j}} \vee \bigvee_{i\in\theta'} \overline{\xi_{i,\mathbf{x}_i}} \mid \mathbf{x} \in \mathbb{F}_p^n \text{ with } \operatorname{supp}(\mathbf{x}) \subseteq \theta\right\} \\
&= \left\{\overline{\xi_{j,k}} \vee \bigvee_{i\in\theta'} \overline{\xi_{i,\mathbf{x}_i}} \mid \mathbf{x} \in \mathbb{F}_p^n \text{ with } \operatorname{supp}(\mathbf{x}) \subseteq \theta' \text{ and } k \in \mathbb{F}_p\right\} \\
&= \left\{\overline{\xi_{j,k}} \vee c' \mid c' \in \Omega(\theta'), k \in \mathbb{F}_p\right\}
\end{aligned}$$

For each $c' \in \Omega(\theta')$, we can derive the clause c' by resolving $\bigvee_{k\in\mathbb{F}_p} \xi_{j,k}$ from V with the clauses from $\Omega(\theta)$. Doing this for all $c' \in \Omega(\theta')$ takes $p \cdot |\Omega(\theta')| = |\Omega(\theta)| = p^{|\theta|}$ resolution steps. By induction, we can then derive $\bot$ from $\Omega(\theta')$ and V in $\frac{p^{|\theta'|+1}-p}{p-1} = \frac{p^{|\theta|}-p}{p-1}$ steps. The total number of steps taken is $\frac{p^{|\theta|}-p}{p-1} + p^{|\theta|} = \frac{p^{|\theta|+1}-p}{p-1}$. ◀

When we sum two vectors $\mathbf{x}$ and $\mathbf{y}$, some components may become zero which were nonzero before. The following definition captures this phenomenon: $\vartheta(\mathbf{x},\mathbf{y}) := (\operatorname{supp}(\mathbf{x}) \cup \operatorname{supp}(\mathbf{y})) \setminus \operatorname{supp}(\mathbf{x}+\mathbf{y})$. If a coefficient vanishes in a sum, it has to appear in both summands: $\vartheta(\mathbf{x},\mathbf{y}) \subseteq \operatorname{supp}(\mathbf{x}) \cap \operatorname{supp}(\mathbf{y})$ (see Figure 3).

With these ingredients, we can finally explain the process of building sums using resolution.

▶ **Theorem 20** (Sum Resolution). *Let $\mathbf{a} \in \mathbb{F}_p^{2\times n}$ and $\mathbf{b} \in \mathbb{F}_p^2$. Define $\theta := \vartheta(\mathbf{a}_1, \mathbf{a}_2)$, where $\mathbf{a}_i$ is the i-th row of a. For all $c \in F(\boldsymbol{\Sigma}\mathbf{a}, \boldsymbol{\Sigma}\mathbf{b})$ it holds: $F(\mathbf{a}_1, \mathbf{b}_1) \cup F(\mathbf{a}_2, \mathbf{b}_2) \cup V \vdash^{w}_{2(p^{|\theta|}-1)} c$.*

Proof. Let $c \in F(\mathbf{\Sigma a}, \mathbf{\Sigma b})$. By definition of F, there exists some $\mathbf{x} \in P(\mathbf{\Sigma a}, \mathbf{\Sigma b})$ such that $c = \bigvee_{i \in \mathrm{supp}(\mathbf{\Sigma a})} \overline{\xi_{i,\mathbf{x}_i}}$. The resolution derivation will have to get rid of all variables corresponding to components in θ. For $\kappa \in \{1,2\}$ define $R^\kappa := \bigvee_{i \in \mathrm{supp}(\mathbf{a}^\kappa)\setminus\theta} \overline{\xi_{i,\mathbf{x}_i}}$. From this we will build the desired clause c. It holds: $(\mathrm{supp}(\mathbf{a}_1) \cup \mathrm{supp}(\mathbf{a}_2)) \setminus \theta = (\mathrm{supp}(\mathbf{a}_1) \cup \mathrm{supp}(\mathbf{a}_2)) \cap \mathrm{supp}(\mathbf{\Sigma a}) = \mathrm{supp}(\mathbf{\Sigma a})$. Thus

$$\begin{aligned} R^1 \vee R^2 &= \bigvee_{i \in \mathrm{supp}(\mathbf{a}_1)\setminus\theta} \overline{\xi_{i,\mathbf{x}_i}} \vee \bigvee_{i \in \mathrm{supp}(\mathbf{a}_2)\setminus\theta} \overline{\xi_{i,\mathbf{x}_i}} \\ &= \bigvee_{i \in (\mathrm{supp}(\mathbf{a}_1)\cup\mathrm{supp}(\mathbf{a}_2))\setminus\theta} \overline{\xi_{i,\mathbf{x}_i}} \\ &= \bigvee_{i \in \mathrm{supp}(\mathbf{\Sigma a})} \overline{\xi_{i,\mathbf{x}_i}} = c \end{aligned}$$

Consider an arbitrary $\mathbf{y} \in \mathbb{F}_p^n$ with $\mathrm{supp}(\mathbf{y}) \subseteq \theta$. Since $\mathrm{supp}(\mathbf{y}) \cap \mathrm{supp}(\mathbf{\Sigma a}) = \emptyset$, we have $\mathbf{\Sigma a} \cdot \mathbf{y} = 0$. There exists κ with $\mathbf{a}_\kappa \cdot (\mathbf{x}+\mathbf{y}) \neq b_\kappa$, because otherwise we would have $\mathbf{\Sigma b} = \mathbf{b}_1 + \mathbf{b}_2 = \mathbf{a}_1 \cdot (\mathbf{x}+\mathbf{y}) + \mathbf{a}_2 \cdot (\mathbf{x}+\mathbf{y}) = (\mathbf{a}_1 + \mathbf{a}_2) \cdot (\mathbf{x}+\mathbf{y}) = \mathbf{\Sigma a} \cdot \mathbf{x} + \mathbf{\Sigma a} \cdot \mathbf{y} = \mathbf{\Sigma a} \cdot \mathbf{x}$, which contradicts $\mathbf{x} \in P(\mathbf{\Sigma a}, \mathbf{\Sigma b})$.

Hence $(\mathbf{x}+\mathbf{y})|_{\mathbf{a}_\kappa} \in P(\mathbf{a}_\kappa, \mathbf{b}_\kappa)$ and we have a clause $c'(\mathbf{y}) := \bigvee_{i \in \mathrm{supp}(\mathbf{a}_\kappa)} \overline{\xi_{i,(\mathbf{x}+\mathbf{y})_i}} \in F(\mathbf{a}_\kappa, \mathbf{b}_\kappa)$. It holds:

$$\begin{aligned} c'(\mathbf{y}) &= \bigvee_{i \in \mathrm{supp}(\mathbf{a}_\kappa)\cap\theta} \overline{\xi_{i,(\mathbf{x}+\mathbf{y})_i}} \vee \bigvee_{i \in \mathrm{supp}(\mathbf{a}_\kappa)\setminus\theta} \overline{\xi_{i,(\mathbf{x}+\mathbf{y})_i}} \\ &= \bigvee_{i \in \mathrm{supp}(\mathbf{a}_\kappa)\cap\theta} \overline{\xi_{i,\mathbf{y}_i}} \vee \bigvee_{i \in \mathrm{supp}(\mathbf{a}_\kappa)\setminus\theta} \overline{\xi_{i,\mathbf{x}_i}} \\ &= \bigvee_{i \in \theta} \overline{\xi_{i,\mathbf{y}_i}} \vee R^\kappa \end{aligned}$$

Now, looking at the set $C := \{c'(\mathbf{y}) \mid \mathbf{y} \in \mathbb{F}_p^n\} \subseteq F(\mathbf{a}_1, \mathbf{b}_1) \cup F(\mathbf{a}_2, \mathbf{b}_2)$, note that for every clause $d \in \Omega(\theta)$, we have $d \vee R^1 \in C$ or $d \vee R^2 \in C$. By Lemma 19 and Lemma 6 we can resolve the clauses in C together with V to obtain $R^1 \vee R^2 = c$ or stronger. Since $p \geq 2$, this takes at most $\frac{p^{|\theta|+1}-p}{p-1} \leq \frac{p^{|\theta|+1}-p}{p/2} = 2(p^{|\theta|}-1)$ resolution steps. ◀

By applying Theorem 20 iteratively, we can construct the formulas for linear combinations with an arbitrary number of summands. This method, however, is inefficient since the produced intermediate equations may accumulate more and more variables, leading to an exponential growth of the number of required clauses.

We solve this problem by deriving only a single representative clause for intermediate results, and using the local symmetry rule to derive more clauses as necessary.

4.2 Local Symmetry in Equations

We want to understand which symmetries the formulas corresponding to linear equations have. For $\mathbf{d} \in \mathbb{F}_p^n$ define $\Delta_{\mathbf{d}} : \mathrm{Vars} \to \mathrm{Vars} : \xi_{i,k} \mapsto \Delta_{\mathbf{d}}(\xi_{i,k}) := \xi_{i,k+\mathbf{d}_i}$. This bijective map is a translation by $\mathbf{d}$ of the vector corresponding to the variables.

▶ **Lemma 21.** *Let $b \in \mathbb{F}_p$ and $\mathbf{a}, \mathbf{d} \in \mathbb{F}_p^n$. Then $\Delta_{\mathbf{d}} \in \mathrm{Sym}(F(\mathbf{a}, b))$ if and only if $\mathbf{a} \cdot \mathbf{d} = 0$.*

Proof. $\Longleftarrow$: Assume $\mathbf{a} \cdot \mathbf{d} = 0$. Let $c = C_{\mathbf{a}}(\mathbf{x}) = \bigvee_{i \in \mathrm{supp}(\mathbf{a})} \overline{\xi_{i,\mathbf{x}_i}} \in F(\mathbf{a}, b)$ for some $\mathbf{x} \in P(\mathbf{a}, b)$. We have $\mathbf{a} \cdot (\mathbf{x}+\mathbf{d}) = \mathbf{a} \cdot \mathbf{x} \neq b$. Hence $(\mathbf{x}+\mathbf{d})|_{\mathbf{a}} \in P(\mathbf{a}, b)$ and thus $\Delta_{\mathbf{d}}(c) = \bigvee_{i \in \mathrm{supp}(\mathbf{a})} \overline{\xi_{i,\mathbf{x}_i+\mathbf{d}_i}} = C_{\mathbf{a}}(\mathbf{x}+\mathbf{d}) = C_{\mathbf{a}}((\mathbf{x}+\mathbf{d})|_{\mathbf{a}}) \in F(\mathbf{a}, b)$.

$\Longrightarrow:$ Assume $\mathbf{a} \cdot \mathbf{d} \neq 0$. Then $\mathbf{a} \neq \mathbf{0}$ and hence there exists a vector $\mathbf{x} \in P(\mathbf{a}, b)$ with $\mathbf{a} \cdot \mathbf{x} = b - \mathbf{a} \cdot \mathbf{d} \neq b$. But then $\mathbf{a} \cdot (\mathbf{x} + \mathbf{d}) = b$ and thus $\Delta_{\mathbf{d}}(C_{\mathbf{a}}(\mathbf{x})) = C_{\mathbf{a}}(\mathbf{x} + \mathbf{d}) \notin F(\mathbf{a}, b)$. Hence $\Delta_{\mathbf{d}} \notin \mathrm{Sym}(F(\mathbf{a}, b))$. ◀

▶ **Corollary 22.** *Let* $\mathbf{A} \in \mathbb{F}_p^{m \times n}$, $\mathbf{b} \in \mathbb{F}_p^m$ *and* $\mathbf{d} \in \mathbb{F}_p^n$. *If* $\mathbf{A} \cdot \mathbf{d} = \mathbf{0}$, *then* $\Delta_{\mathbf{d}} \in \mathrm{Sym}(F(\mathbf{A}, \mathbf{b}))$.

Note the following: If $\mathbf{d}, \mathbf{d}' \in \mathbb{F}_p^n$ such that $\mathbf{d}|_{\mathbf{a}} = \mathbf{d}'|_{\mathbf{a}}$, then for all $c \in F(\mathbf{a}, b)$ it holds: $\Delta_{\mathbf{d}}(c) = \Delta_{\mathbf{d}'}(c)$. In particular, $\Delta_{\mathbf{d}} \in \mathrm{Sym}(F(\mathbf{a}, b))$ implies $\Delta_{\mathbf{d}'} \in \mathrm{Sym}(F(\mathbf{a}, b))$. The condition $\mathbf{d}|_{\mathbf{a}} = \mathbf{d}'|_{\mathbf{a}}$ can equivalently be expressed using matrix algebra: $\mathrm{diag}(\mathbf{a})\mathbf{d} = \mathrm{diag}(\mathbf{a})\mathbf{d}'$.

To make use of the symmetry rule, we want to apply the symmetries of $F(\mathbf{A}, \mathbf{b})$ to derive clauses of $F(\mathbf{\Sigma A}, \mathbf{\Sigma b})$. From the statements and Corollary 22, we conclude the following relation between $\mathrm{Sym}(F(\mathbf{A}, \mathbf{b}))$ and $\mathrm{Sym}(F(\mathbf{\Sigma A}, \mathbf{\Sigma b}))$.

▶ **Lemma 23.** *Let* $\mathbf{A} \in \mathbb{F}_p^{m \times n}$, $\mathbf{b} \in \mathbb{F}_p^m$ *and* $\mathbf{d} \in \mathbb{F}_p^n$ *with* $\Delta_{\mathbf{d}} \in \mathrm{Sym}(F(\mathbf{\Sigma A}, \mathbf{\Sigma b}))$ *If there exists* $\mathbf{d}' \in \mathbb{F}_p^n$ *such that* $\mathbf{A}\mathbf{d}' = \mathbf{0}$ *and* $\mathrm{diag}(\mathbf{\Sigma A})\mathbf{d}' = \mathrm{diag}(\mathbf{\Sigma A})\mathbf{d}$, *then* $\Delta_{\mathbf{d}'} \in \mathrm{Sym}(F(\mathbf{A}, \mathbf{b}))$ *and* $\Delta_{\mathbf{d}'}(c) = \Delta_{\mathbf{d}}(c)$ *for all* $c \in F(\mathbf{\Sigma A}, \mathbf{\Sigma b})$.

Concerning V, the symmetries are simpler: For any $\mathbf{d} \in \mathbb{F}_p^n$ we have $\Delta_{\mathbf{d}} \in \mathrm{Sym}(V)$.

We will assume that the coefficient matrices $\mathbf{A}$ in the following have at most L nonzero entries in each row. In other words, the width of $\mathbf{A}$ is at most L.

▶ **Theorem 24.** *Let* $\mathbf{A} \in \mathbb{F}_p^{m \times n}$ *and* $\mathbf{b} \in \mathbb{F}_p^m$. *For any* $H \subseteq F(\mathbf{\Sigma A}, \mathbf{\Sigma b})$ *it holds:* $F(\mathbf{A}, \mathbf{b}) \wedge V \vdash^{\mathit{SRC\text{-}II}}_{\mathcal{O}(m^{\Theta(p)} p^{L+1}) + |H|} H$.

Proof. Define $\lambda := \frac{\log(2)}{\log(p/(p-1))}$ and $f(x) := Cp^{L+1}x^{\lambda}$ for some constant C chosen later. Regarding the relationship between λ and p we have $\lambda \sim \log(2)p$ (i.e., $\lim_{n \to \infty} \lambda / \log(2)p = 1$)).

We prove the following by induction over the number of equations m: For any $H \subseteq F(\mathbf{\Sigma A}, \mathbf{\Sigma b})$ it holds: $F(\mathbf{A}, \mathbf{b}) \wedge V \vdash^{\text{SRC-II}}_{f(m) + |H|} H$.

Induction basis: $m = 0$. In this case, we have $\mathbf{\Sigma A} = \mathbf{0}$ and $\mathbf{\Sigma b} = 0$. Then $F(\mathbf{\Sigma A}, \mathbf{\Sigma b}) = \emptyset$; hence $H = \emptyset$ and we have nothing to prove.

Induction step: $m - 1 \to m$. Here we have two cases:

Case 1: Symmetric sum. For this case we assume that for all $\mathbf{d}$ with $\Delta_{\mathbf{d}} \in \mathrm{Sym}(F(\mathbf{\Sigma A}, \mathbf{\Sigma b}))$ we have a $\mathbf{d}'$ with $\mathbf{d}|_{\mathbf{\Sigma A}} = \mathbf{d}'|_{\mathbf{\Sigma A}}$ and $\Delta_{\mathbf{d}'} \in \mathrm{Sym}(F(\mathbf{A}, \mathbf{b}))$. Thanks to this property, all the symmetries of $F(\mathbf{\Sigma A}, \mathbf{\Sigma b})$ are already present in $F(\mathbf{A}, \mathbf{b})$ and can be used by the symmetry rule. So we only need to derive a few clauses of $F(\mathbf{\Sigma A}, \mathbf{\Sigma b})$ to obtain a set allowing us to generate all clauses via symmetries. This which can be done by inductively applying Theorem 20 as follows.

Define $\mathbf{A}'$ and $\mathbf{b}'$ to be the first $m - 1$ rows of $\mathbf{A}$ and $\mathbf{b}$ respectively. Define $\theta := \vartheta(\mathbf{\Sigma A}', \mathbf{A}_{m,*})$. We have $|\theta| \leq |\mathrm{supp}(\mathbf{A}_{m,*})| \leq L$. If $\mathbf{\Sigma A} \neq \mathbf{0}$, then for each $k \neq \mathbf{\Sigma b}$ there exists a vector $\mathbf{z}^k$ with $\mathrm{supp}(\mathbf{z}^k) \subseteq \mathrm{supp}(\mathbf{\Sigma A})$ such that $\mathbf{\Sigma A} \cdot \mathbf{z}^k = k$. Define $G := \{C_{\mathbf{\Sigma A}}(\mathbf{z}^k) \mid k \in \mathbb{F}_p \setminus \{\mathbf{\Sigma b}\}\} \subseteq F(\mathbf{\Sigma A}, \mathbf{\Sigma b})$. Then $|G| = p - 1$.

Using Theorem 20, we get $F(\mathbf{\Sigma A}', \mathbf{\Sigma b}') \wedge F(\mathbf{A}_{m,*}, \mathbf{b}_m) \wedge V \vdash^{w}_{|G| \cdot \mathcal{O}(p^L)} G$. This derivation only uses a subset $H' \subseteq F(\mathbf{\Sigma A}', \mathbf{\Sigma b}')$ of at most $|H'| \leq \mathcal{O}(p^{L+1})$ clauses. By induction, it holds that $F(\mathbf{A}', \mathbf{b}') \wedge V \vdash^{\text{SRC-II}}_{f(m-1) + |H'|} H'$. We can combine these derivations by Lemma 5 to obtain $F(\mathbf{A}, \mathbf{b}) \wedge V \vdash^{\text{SRC-II}}_{f(m-1) + \mathcal{O}(p^{L+1})} G$. Using $\lambda \geq 1$, we take in total $f(m-1) + Cp^{L+1} = Cp^{L+1}(m-1)^{\lambda} + Cp^{L+1} \leq Cp^{L+1}m^{\lambda} = f(m)$ steps, for some constant C.

Now we show that G is a generator for H: Let $c \in F(\mathbf{\Sigma A}, \mathbf{\Sigma b})$. Then $c = C_{\mathbf{\Sigma A}}(\mathbf{x})$ for some $\mathbf{x} \in P(\mathbf{\Sigma A}, \mathbf{\Sigma b})$. Define $\mathbf{d} := \mathbf{x} - \mathbf{z}^{\mathbf{\Sigma A} \cdot \mathbf{x}}$. Then $\mathbf{\Sigma A} \cdot \mathbf{d} = 0$, so $\Delta_{\mathbf{d}} \in \mathrm{Sym}(F(\mathbf{\Sigma A}, \mathbf{\Sigma b}))$. Hence there exists a $\varphi \in \mathrm{Sym}(F(\mathbf{A}, \mathbf{b}))$ such that $\varphi(C_{\mathbf{\Sigma A}}(\mathbf{z}^{\mathbf{\Sigma A} \cdot \mathbf{x}})) = \Delta_{\mathbf{d}}(C_{\mathbf{\Sigma A}}(\mathbf{z}^{\mathbf{\Sigma A} \cdot \mathbf{x}})) = C_{\mathbf{\Sigma A}}(\mathbf{x}) = c$. We can apply the local symmetry rule to derive c from G in a single step, using the symmetries of $F(\mathbf{A}, \mathbf{b})$. Repeating this for every $c \in H$ yields $F(\mathbf{A}, \mathbf{b}) \wedge V \vdash^{\text{SRC-II}}_{f(m)+|H|} H$.

If $\mathbf{\Sigma A} = \mathbf{0}$ and $\mathbf{\Sigma b} \neq 0$ then $F(\mathbf{\Sigma A}, \mathbf{\Sigma b}) = \{C_{\mathbf{0}}(\mathbf{0})\} = \{\bot\} =: G$, which can be derived in at most Cp^{L+1} steps, again using Theorem 20.

If $\mathbf{\Sigma A} = \mathbf{0}$ and $\mathbf{\Sigma b} = 0$ then $F(\mathbf{\Sigma A}, \mathbf{\Sigma b}) = \emptyset$. We treat this the same way as the case $m = 0$.

Case 2: Composite. If Case 1 does not apply, the following must hold by Lemma 23: For some $\mathbf{d}$ with $\mathbf{\Sigma A} \cdot \mathbf{d} = 0$, the equations $\mathbf{A}\mathbf{d}' = \mathbf{0}$ and $\mathrm{diag}(\mathbf{\Sigma A})\mathbf{d}' = \mathrm{diag}(\mathbf{\Sigma A})\mathbf{d}$ have no common solution $\mathbf{d}'$.

Applying Lemma 16 to the combined inconsistent equations, we have $\mathbf{v}, \mathbf{w}$ such that $\mathbf{vA} + \mathbf{w}\mathrm{diag}(\mathbf{\Sigma A}) = \mathbf{0}$ and $\mathbf{w}\mathrm{diag}(\mathbf{\Sigma A})\mathbf{d} \neq 0$. We will use the vector $\mathbf{v}$ to decompose $\mathbf{A}$ into two smaller matrices, each contributing independently to the derivation of H. First we show that $\mathbf{v}$ has special properties which make this *divide and conquer* approach work. Then we need to ensure that the sub-problems are not too large for our proof length bound f.

It holds: $\mathbf{vA} = -\mathbf{w}\mathrm{diag}(\mathbf{\Sigma A})$; thus $\mathbf{vAd} = -\mathbf{w}\mathrm{diag}(\mathbf{\Sigma A})\mathbf{d} \neq 0$. For all $i \in [1, n]$, if $(\mathbf{\Sigma A})_i = 0$, then $(\mathbf{vA})_i = (-\mathbf{w}\mathrm{diag}(\mathbf{\Sigma A}))_i = -\mathbf{w}_i(\mathbf{\Sigma A})_i = 0$. Hence $\mathrm{supp}(\mathbf{vA}) \subseteq \mathrm{supp}(\mathbf{\Sigma A})$. We show that $\mathbf{vA}$ and $\mathbf{\Sigma A}$ are linearly independent: Let $\alpha_1, \alpha_2 \in \mathbb{F}_p$ such that $\alpha_1\mathbf{vA} + \alpha_2\mathbf{\Sigma A} = \mathbf{0}$. Then $0 = \alpha_1\mathbf{vAd} + \alpha_2\mathbf{\Sigma Ad} = \alpha_1\mathbf{vAd}$, which implies $\alpha_1 = 0$. Since $\mathbf{\Sigma A} \neq \mathbf{0}$, we also have $\alpha_2 = 0$.

Let $k_1 \in \operatorname*{arg\,max}_{k \in \mathbb{F}_p} |\{i \mid \mathbf{v}_i = k\}|$ be the most common component of $\mathbf{v}$. Let $k_2 \in \operatorname*{arg\,max}_{k \in \mathbb{F}_p,\, k \neq k_1} |\{i \mid \mathbf{v}_i = k\}|$ be the second most common component of $\mathbf{v}$. Since $\mathbf{vA}$ is linearly independent from $\mathbf{\Sigma A}$, we have $\mathbf{v} \neq k \cdot \mathbf{1}$ for all $k \in \mathbb{F}_p$, so there are at least two different components in $\mathbf{v}$. Hence k_1 and k_2 exist. Define m_i to be the number of times k_i occurs in $\mathbf{v}$. We have $m_i \geq 1$ for $i \in \{1, 2\}$. Furthermore $m_1 \geq m/p$ and $m_2 \geq (m - m_1)/(p-1)$.

Define $\mathbf{v}^1 := \mathbf{v} - k_1\mathbf{1}$ and $\mathbf{v}^2 := k_2\mathbf{1} - \mathbf{v}$. It holds: $\mathbf{v}^1 + \mathbf{v}^2 = (k_2 - k_1)\mathbf{1}$. By subtracting k_i from every component, we get exactly m_i zeros in $\mathbf{v}^i$, i.e. $|\mathrm{supp}(\mathbf{v}^i)| = m - m_i$.

Towards a contradiction, assume there is some $j \in \vartheta(\mathbf{v}^1\mathbf{A}, \mathbf{v}^2\mathbf{A})$. Then $0 = (\mathbf{v}^1\mathbf{A} + \mathbf{v}^2\mathbf{A})_j = ((k_2 - k_1)\mathbf{\Sigma A})_j$, so $0 = (\mathbf{\Sigma A})_j$. Thus $0 = -\mathbf{w}_j(\mathbf{\Sigma A})_j = (\mathbf{vA})_j = (\mathbf{v}^1\mathbf{A} + k_1\mathbf{\Sigma A})_j = (\mathbf{v}^1\mathbf{A})_j + k_1(\mathbf{\Sigma A})_j = (\mathbf{v}^1\mathbf{A})_j$. This contradicts the assumption. Hence $\vartheta(\mathbf{v}^1\mathbf{A}, \mathbf{v}^2\mathbf{A}) = \emptyset$. By Theorem 20 we can derive the sum clauses of $\mathbf{v}^1\mathbf{A} + \mathbf{v}^2\mathbf{A}$ in 0 steps, so they are already implied by the summand clauses: $F((\mathbf{v}^1 + \mathbf{v}^2)\mathbf{A}, (\mathbf{v}^1 + \mathbf{v}^2)\mathbf{b}) \sqsubseteq F(\mathbf{v}^1\mathbf{A}, \mathbf{v}^1\mathbf{b}) \cup F(\mathbf{v}^2\mathbf{A}, \mathbf{v}^2\mathbf{b})$. It follows that

$$\begin{aligned} F(\mathbf{\Sigma A}, \mathbf{\Sigma b}) &= F((k_2 - k_1)\mathbf{\Sigma A}, (k_2 - k_1)\mathbf{\Sigma b}) \\ &= F((\mathbf{v}^1 + \mathbf{v}^2)\mathbf{A}, (\mathbf{v}^1 + \mathbf{v}^2)\mathbf{b}) \\ &\sqsubseteq F(\mathbf{v}^1\mathbf{A}, \mathbf{v}^1\mathbf{b}) \cup F(\mathbf{v}^2\mathbf{A}, \mathbf{v}^2\mathbf{b}). \end{aligned}$$

Hence we can partition $H \subseteq F(\mathbf{\Sigma A}, \mathbf{\Sigma b})$ into H_1 and H_2 such that $H_i \sqsubseteq F(\mathbf{v}^i\mathbf{A}, \mathbf{v}^i\mathbf{b})$ for $i \in \{1, 2\}$.

Note that $F(\mathbf{v}^i\mathbf{A}, \mathbf{v}^i\mathbf{b}) = F(\mathbf{\Sigma}\mathrm{diag}(\mathbf{v}^i)\mathbf{A}, \mathbf{\Sigma}\mathrm{diag}(\mathbf{v}^i)\mathbf{b})$. Since $|\mathrm{supp}(\mathbf{v}^i)| \leq m - 1$, we have at least one zero row each in $\mathrm{diag}(\mathbf{v}^1)\mathbf{A}$ and $\mathrm{diag}(\mathbf{v}^2)\mathbf{A}$. This makes it possible to apply the induction hypothesis, yielding $F(\mathrm{diag}(\mathbf{v}^i)\mathbf{A}, \mathrm{diag}(\mathbf{v}^i)\mathbf{b}) \wedge V \vdash^{\text{SRC-II}}_{f(|\mathrm{supp}(\mathbf{v}^i)|)+|H_i|} H_i$.

Scaling the equations does not produce different clauses, so we have $F(\mathrm{diag}(\mathbf{v}^1)\mathbf{A}, \mathrm{diag}(\mathbf{v}^1)\mathbf{b}) \cup F(\mathrm{diag}(\mathbf{v}^2)\mathbf{A}, \mathrm{diag}(\mathbf{v}^2)\mathbf{b}) \subseteq F(\mathbf{A}, \mathbf{b})$. Then we can combine the derivations of H_1 and H_2 to obtain $F(\mathbf{A}, \mathbf{b}) \wedge V \vdash^{\text{SRC-II}}_{f(|\mathrm{supp}(\mathbf{v}^1)|)+f(|\mathrm{supp}(\mathbf{v}^2)|)+|H|} H$. It holds:

$f(|\mathrm{supp}(\mathbf{v}^1)|)+f(|\mathrm{supp}(\mathbf{v}^2)|) = f(m-m_1)+f(m-m_2) \leq f(m-m_1)+f(m-(m-m_1)/(p-1)) =: T(m_1)$. By applying standard calculus techniques to the function T, we find that $T(m_1) \leq f(m)$ for all possible values of m_1. ◀

▶ **Corollary 25.** *Let $\mathbf{A} \in \mathbb{F}_p^{m\times n}$ and $\mathbf{b} \in \mathbb{F}_p^m$, such that there is no $\mathbf{x} \in \mathbb{F}_p^n$ satisfying $\mathbf{Ax} = \mathbf{b}$. Then there exists a resolution refutation of $F(\mathbf{A}, \mathbf{b}) \wedge V$ using the local symmetry rule, with its length bounded by $\mathcal{O}(m^{\Theta(p)}p^{L+1})$.*

5 Linear-sized Refutations for Non-Isomorphism of Multipedes

We can use the result on linear equations to show that there are short resolution proofs for the non-isomorphism of Multipede graphs.

▶ **Theorem 26.** *Let $G = (V, W, E)$ be a connected bipartite graph such that $MP(G)$ is asymmetric, and $\omega \in W$. Then $F\left(MP(G)_{a_\omega}, MP(G)_{b_\omega}\right)$ has a linear-sized resolution refutation using the local symmetry rule.*

Proof. Let $G = (\{v_1, \dots, v_m\}, \{w_1, \dots, w_n\}, E)$ be a connected bipartite graph and $\omega := w_k$ for some k. Our goal is to apply the techniques of the previous section to the formula $F_0 := F\left(MP(G)_{a_\omega}, MP(G)_{b_\omega}\right)$.

We first inspect the simpler formula $F_1 := F(MP(G), MP(G))$. The solutions of this formula correspond to the automorphisms of $MP(G)$. By applying resolution to F_1, we can derive the formula $F(\mathbf{M}(G), \mathbf{0})$: Let $i \in [1, m]$. Then

$$\begin{aligned} F(\mathbf{M}(G)_{i,*}, 0) &= \bigwedge_{\mathbf{x}\in P(\mathbf{M}(G)_{i,*},0)} \; \bigvee_{j\in\mathrm{supp}(\mathbf{M}(G)_{i,*})} \overline{\xi_{j,\mathbf{x}_j}} \\ &= \bigwedge_{\mathbf{x}\in P(\mathbf{M}(G)_{i,*},0)} \; \bigvee_{\{v_i,w_j\}\in E} \overline{\xi_{j,\mathbf{x}_j}} \\ &= \bigwedge_{\substack{B\subseteq N_G(v_i)\\ |B| \text{ odd}}} \left(\bigvee_{w_j\in B} \overline{\xi_{j,1}} \vee \bigvee_{w_j\in N_G(v_i)\setminus B} \overline{\xi_{j,0}} \right) \end{aligned}$$

Define $N := N_G(v_i)$. We define $\mathcal{P}_{even}(N)$ to be the subsets of N with even cardinality. For all $B \subseteq N$ with odd $|B|$ there exists a surjective function $\gamma : \mathcal{P}_{even}(N) \to N$ such that for all $S \in \mathcal{P}_{even}(N) : \gamma(S) \in S \setminus B \cup B \setminus S$. We can make the following resolution derivation from F_1:

$$\begin{array}{rr} \text{Type 1:} & \displaystyle\bigvee_{S\in\mathcal{P}_{even}(N)} z^v_{\emptyset,S} \\ \forall S \in \mathcal{P}_{even}(N) \text{ with } w := \gamma(S) \in B \setminus S : \text{Type 3:} & \overline{z^v_{\emptyset,S}} \vee \overline{y_{a_w,b_w}} \\ \forall S \in \mathcal{P}_{even}(N) \text{ with } w := \gamma(S) \in S \setminus B : \text{Type 3:} & \overline{z^v_{\emptyset,S}} \vee \overline{y_{a_w,a_w}} \\ \Longrightarrow & \displaystyle\bigvee_{w\in B} \overline{y_{a_w,b_w}} \vee \bigvee_{w\in N\setminus B} \overline{y_{a_w,a_w}}, \end{array}$$

taking $|\mathcal{P}_{even}(N)| \leq 2^{|N|}$ steps. Repeating this process for every B and i takes $\sum_{v\in V} |\mathcal{P}_{odd}(N_G(v))| \cdot 2^{|N_G(v)|} = \mathcal{O}(|F_1|)$ resolution steps. Define a variable renaming r on $F(\mathbf{M}(G), \mathbf{0})$ as follows:

$$r(\xi_{j,\kappa}) := \begin{cases} y_{a_{w_j},a_{w_j}} & \text{if } \kappa = 0 \\ y_{a_{w_j},b_{w_j}} & \text{if } \kappa = 1 \end{cases}$$

Then we have the derivation $F_1 \vdash_{\mathcal{O}(|F_1|)} r(F(\mathbf{M}(G), \mathbf{0}))$. As a consequence, $|F(\mathbf{M}(G), \mathbf{0})| = \mathcal{O}(|F_1|)$. The clauses of $r(V)$ are simply the Type 1 clauses of F_1.

To apply Theorem 24, we need to translate the symmetries $\Delta_{\mathbf{d}} \in \mathrm{Sym}(F(\mathbf{M}(G), \mathbf{0}))$ into symmetries of F_1. Let $\mathbf{d} \in \mathbb{F}_2^n$ such that $\mathbf{M}(G)\mathbf{d} = \mathbf{0}$. Define $D := \{w_i \in W \mid \mathbf{d}_i = 1\}$. Then the following map $\psi_{\mathbf{d}}$ is a symmetry of F_1: for $w \in W$ set $\psi_{\mathbf{d}}(y_{a_w,a_w})$ to be y_{a_w,b_w} if $w \in D$ and y_{a_w,a_w} otherwise. Similarly $\psi_{\mathbf{d}}(y_{a_w,b_w})$ is y_{a_w,a_w} if $w \in D$ and y_{a_w,b_w}. We also set $\psi_{\mathbf{d}}(y_{b_w,a_w})$ to be y_{b_w,b_w} if $w \in D$ and y_{b_w,a_w} and we set $\psi_{\mathbf{d}}(y_{b_w,b_w})$ to be y_{b_w,a_w} if $w \in D$ and y_{b_w,b_w}. Finally for $v \in V$ we define $\psi_{\mathbf{d}}(z^v_{S,T}) := z^v_{S,T\triangle D}$ and have the property $\psi_{\mathbf{d}}(r(c)) = r(\Delta_{\mathbf{d}}(c))$ for all clauses $c \in F(\mathbf{M}(G), \mathbf{0})$.

Now, if the graph $MP(G)$ is asymmetric, the only solution of $\mathbf{M}(G)\mathbf{y} = \mathbf{0}$ is $\mathbf{y} = \mathbf{0}$. Then we can deduce $\mathbf{y}_k = 0$ from the equation system by combining rows. Applying Theorem 24, we get $F(\mathbf{M}(G), \mathbf{0}) \wedge V \vdash^{\text{SRC-II}}_{\mathcal{O}(m2^{L+1})} \xi_{k,0}$. Renaming variables yields $r(F(\mathbf{M}(G), \mathbf{0})) \wedge r(V) \vdash^{\text{SRC-II}}_{\mathcal{O}(m2^{L+1})} y_{a_\omega,a_\omega}$. As we have seen, $r(F(\mathbf{M}(G), \mathbf{0}))$ and $r(V)$ can be derived from F_1 and the symmetries are preserved; hence $F_1 \vdash^{\text{SRC-II}}_{\mathcal{O}(m2^{L+1})} y_{a_\omega,a_\omega}$.

Note that F_0 is obtained from F_1 simply by replacing y_{a_ω,a_ω} and y_{b_ω,b_ω} with 0. Hence, $F_0 \vdash^{\text{SRC-II}}_{\mathcal{O}(m2^{L+1})} \perp$. ◀

References

1 Noriko H. Arai and Alasdair Urquhart. Local symmetries in propositional logic. In Roy Dyckhoff, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2000, St Andrews, Scotland, UK, July 3-7, 2000, Proceedings*, volume 1847 of *Lecture Notes in Computer Science*, pages 40–51. Springer, 2000. doi:10.1007/10722086_3.

2 Belaid Benhamou and Lakhdar Sais. Tractability through symmetries in propositional calculus. *J. Autom. Reasoning*, 12(1):89–102, 1994. doi:10.1007/BF00881844.

3 Joshua Blinkhorn and Olaf Beyersdorff. Proof complexity of QBF symmetry recomputation. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing – SAT 2019 – 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 36–52. Springer, 2019. doi:10.1007/978-3-030-24258-9_3.

4 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.

5 Vasek Chvátal and Endre Szemerédi. Many hard examples for resolution. *J. ACM*, 35(4):759–768, 1988. doi:10.1145/48014.48016.

6 Thierry Boy de la Tour and Stéphane Demri. On the complexity of extending ground resolution with symmetry rules. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995*, pages 289–297. Morgan Kaufmann, 1995. URL: http://ijcai.org/Proceedings/95-1/Papers/038.pdf.

7 Heidi E. Dixon, Matthew L. Ginsberg, David K. Hofer, Eugene M. Luks, and Andrew J. Parkes. Implementing a generalized version of resolution. In Deborah L. McGuinness and George Ferguson, editors, *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, pages 55–60. AAAI Press / The MIT Press, 2004. URL: http://www.aaai.org/Library/AAAI/2004/aaai04-009.php.

8 Uwe Egly. A first order resolution calculus with symmetries. In Andrei Voronkov, editor, *Logic Programming and Automated Reasoning,4th International Conference, LPAR'93, St. Petersburg, Russia, July 13-20, 1993, Proceedings*, volume 698 of *Lecture Notes in Computer Science*, pages 110–121. Springer, 1993. doi:10.1007/3-540-56944-8_46.

9 Michael Frank and Michael Codish. Logic programming with graph automorphism: Integrating nauty with prolog (tool description). *TPLP*, 16(5-6):688–702, 2016. doi:10.1017/S1471068416000223.

10 Yuri Gurevich and Saharon Shelah. On finite rigid structures. *J. Symb. Log.*, 61(2):549–562, 1996. doi:10.2307/2275675.

11 Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985. doi:10.1016/0304-3975(85)90144-6.

12 Balakrishnan Krishnamurthy. Short proofs for tricky formulas. *Acta Informatica*, 22(3):253–275, August 1985. doi:10.1007/BF00265682.

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

14 Daniel Neuen and Pascal Schweitzer. An exponential lower bound for individualization-refinement algorithms for graph isomorphism. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 138–150. ACM, 2018. doi:10.1145/3188745.3188900.

15 Pascal Schweitzer and Constantin Seebach. Resolution with symmetry rule applied to linear equations. *CoRR*, abs/2101.05142, 2021. (Full version of the paper). arXiv:2101.05142.

16 Stefan Szeider. The complexity of resolution with generalized symmetry rules. *Theory Comput. Syst.*, 38(2):171–188, 2005. doi:10.1007/s00224-004-1192-0.

17 Jacobo Torán. On the resolution complexity of graph non-isomorphism. In Matti Järvisalo and Allen Van Gelder, editors, *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, volume 7962 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 2013. doi:10.1007/978-3-642-39071-5_6.

18 G. S. Tseitin. *On the Complexity of Derivation in Propositional Calculus*, pages 466–483. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983. doi:10.1007/978-3-642-81955-1_28.

19 Alasdair Urquhart. Hard examples for resolution. *J. ACM*, 34(1):209–219, January 1987. doi:10.1145/7531.8928.

20 Alasdair Urquhart. The symmetry rule in propositional logic. *Discrete Applied Mathematics*, 96-97:177–193, 1999. doi:10.1016/S0166-218X(99)00039-6.

Quantum Approximate Counting with Nonadaptive Grover Iterations

Ramgopal Venkateswaran[1]
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Ryan O'Donnell
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

Approximate Counting refers to the problem where we are given query access to a function $f : [N] \to \{0,1\}$, and we wish to estimate $K = \#\{x : f(x) = 1\}$ to within a factor of $1 + \epsilon$ (with high probability), while minimizing the number of queries. In the quantum setting, Approximate Counting can be done with $O\left(\min\left(\sqrt{N/\epsilon}, \sqrt{N/K}\,/\,\epsilon\right)\right)$ queries. It has recently been shown that this can be achieved by a simple algorithm that only uses "Grover iterations"; however the algorithm performs these iterations adaptively. Motivated by concerns of computational simplicity, we consider algorithms that use Grover iterations with limited adaptivity. We show that algorithms using only nonadaptive Grover iterations can achieve $O\left(\sqrt{N/\epsilon}\right)$ query complexity, which is tight.

2012 ACM Subject Classification Theory of computation → Quantum complexity theory

Keywords and phrases quantum approximate counting, Grover search

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.59

Related Version *Previous Version*: `https://arxiv.org/pdf/2010.04370.pdf`

Funding *Ryan O'Donnell*: Supported by NSF grant CCF-1717606. This material is based upon work supported by the National Science Foundation under the grant number listed above. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation (NSF).

Acknowledgements The authors would like to thank Scott Aaronson and Patrick Rall for helpful communications.

1 Introduction

1.1 Grover Search recap

A famous, textbook algorithm in quantum computing is *Grover Search* [6], which solves the following task: Given is a quantum oracle for a function $f : [N] \to \{0,1\}$, where queries for $f(x)$ may be made in quantum superposition. It is promised that $K = \#\{x : f(x) = 1\}$ is exactly 1. The task is to find x^* such that $f(x^*) = 1$. Grover Search solves this problem (with high probability) using $O(\sqrt{N})$ queries.

The algorithm is particularly simple: First, a state $|s\rangle$ equal to the uniform superposition over all $|x\rangle$ is prepared; this state makes an angle of $\arccos\sqrt{1/N}$ with $|x^*\rangle$. We write $|x^*\rangle^\perp$ for the state perpendicular to $|x^*\rangle$ making an angle of $\theta^* = \arcsin\sqrt{1/N}$ with $|s\rangle$. Then the algorithm repeatedly performs *Grover iterations*, each of which consists of one query followed by the simple "Grover diffusion" operation. The effect of a Grover iteration is to rotate $|s\rangle$ by an angle of $2\theta^*$; thus after r rotations the angle of $|s\rangle$ from $|x^*\rangle^\perp$ is

[1] The ordering of the authors was randomized. The authors contributed equally.

$(2r+1)\theta^*$. Setting $r \approx \frac{\pi}{4}\sqrt{N}$, the algorithm makes $O(\sqrt{N})$ queries and ends up with a state at angle approximately $\frac{\pi}{2}$ from $|x^*\rangle^\perp$; measuring then results in $|x^*\rangle$ with probability $\sin^2((2r+1)\theta^*) \approx \sin^2 \frac{\pi}{2} = 1$.

In this form the algorithm relies on the assumption $K = 1$. For $K \geq 1$, the only change is that the parameter θ^* becomes $\arcsin \sqrt{K/N}$. Thus if the correct value of K is known to the algorithm (and we assume for simplicity that $K \leq N/2$), it can choose $r \approx \frac{\pi}{4}\sqrt{N/K}$ and solve the search problem using $O(\sqrt{N/K})$ iterations/queries. This algorithm also works if the algorithm knows an estimate K' of K that is correct up to small multiplicative error; say, $K' \overset{1.1}{\approx} K$. Here we are using the following notation:

▶ **Notation 1.** For $a, b, \eta > 0$ we write $a \overset{1+\eta}{\approx} b$ if $\frac{1}{1+\eta} \leq a/b \leq 1+\eta$.

If K is *unknown* to the algorithm, one possibility is try all estimates $K' = 1$, 1.1, 1.1^2, 1.1^3, $\dots$, $N/2$. The total number of Grover iterations (hence queries) will be $O(\sum_i \sqrt{N/1.1^i}) = O(\sqrt{N})$.[2] A strategy with improved query complexity for large K was given by Boyer, Brassard, Høyer, and Tapp [2]; in brief, for $i = 0, 1, 2, \dots$ it tries a random number of rotations in the range $[1, 1.1^i]$, stopping if ever an $x \in f^{-1}(1)$ is found. This algorithm solves the search problem using $O(\sqrt{N/K})$ iterations, despite not knowing K in advance.

1.2 Approximate Counting

A natural related problem, called *Approximate Counting* and introduced in [2], is to *estimate* K. More precisely, given as input a parameter $\epsilon \geq 0$, the task is to output a number $\widehat{K}$ such that (with high probability) $\widehat{K} \overset{1+\epsilon}{\approx} K$. To keep the exposition simple, in this paper we will make the following standard assumptions:

- $1/N \leq \epsilon \leq 1$ (setting $\epsilon = 1/N$ just yields the problem of exact counting – any smaller value of ϵ does not change the problem);
- $K \leq N/2$ (otherwise there is generally a dependence on $N - K$, since one can switch the roles of 0 and 1 in f's output);
- $K \neq 0$ (generally, all algorithms can easily be extended to work in the case of $K = 0$, with query complexity being the worst-case query complexity over all $K > 0$).

The quantum Approximate Counting problem was solved with optimal query complexity by Brassard, Høyer, Mosca, and Tapp [3]. Combining quantum Fourier transform ideas from Shor's Algorithm with the ideas behind Grover Search, their algorithm solves Approximate Counting using $O(\sqrt{N/K}\,/\epsilon)$ queries.

Let us make some remarks about this query complexity. First, note that the bound takes K into account, even though K is (initially) unknown to the algorithm. Second, although K could be as small as 1, the worst-case query complexity over all K need *not* be $\Omega(\sqrt{N}\,/\epsilon)$. (Indeed, this would lead to an illogical query complexity of $\Omega(N^{3/2})$ if one set $\epsilon = 1/N$ to do exact counting.) Instead, note that an algorithm can first run the algorithm from [3] with $\epsilon = 1$, expending $O(\sqrt{N/K}) \leq O(\sqrt{N})$ queries and learning a preliminary estimate $K' \overset{2}{\approx} K$. Now since K is an integer, there is no point in trying to approximate it to a factor better than $1 + 1/K$, hence better than $1 + 1/(2K')$. Thus the algorithm can now raise the initial input ϵ to $1/(2K')$ if necessary, and *then* run [3] to obtain its final estimate. This yields a final query complexity of

$$O(\sqrt{N/K}\,/\max\{\epsilon, 1/K\}) = \min\{O(\sqrt{NK}), O(\sqrt{N/K}\,/\epsilon)\} \leq O(\sqrt{N/\epsilon})$$

[2] One must take a small amount of care to bound the overall failure probability without incurring a log factor.

(where in the last inequality we took the geometric mean). This K-independent bound of $O(\sqrt{N/\epsilon})$ is logical: the smallest ϵ one should ever take is $\epsilon = 1/N$, and this leads to a query complexity of $O(N)$ for the general case of exact counting. By similar reasoning one can obtain the more precise fact that exact counting can done with $O(\sqrt{NK})$ queries. Finally, we remark that query complexity obtained by [3] was shown to be optimal by Nayak and Wu [8].

Let us also briefly mention the quantum *Amplitude Estimation* problem, which is essentially the same as the Approximate Counting problem except that the "initial angle" θ^* need not be of the form arcsin $\sqrt{K/N}$ for some integer K, but can be any value. The solution to the Amplitude Estimation problem in [3] is a widely used tool in quantum algorithm design, and leads to quadratic speedups over classical algorithms for a variety of statistical problems.

1.3 Simpler and nonadaptive?

Although the Approximate Counting algorithm from [3] has optimal query complexity, there has recently been a lot of interest in simplifying it [10, 9, 1]. In particular the latter two of these just-cited works strove to replace it with an algorithm that *only* uses Grover iterations, both for analytic simplicity and practical simplicity (the controlled amplifications of [3] being particularly problematic for NISQ devices). The work of Aaronson and Rall [1] provably succeeds at this challenge, providing an algorithm that solves the Approximate Counting problem using $O(\sqrt{N/K}\,/\,\epsilon)$ Grover iterations (hence queries). Briefly, the Aaronson–Rall algorithm has a first phase (somewhat similar to the algorithm in [2]) that performs a geometrically increasing sequence of Grover iterations until K can be estimated up to a constant factor of 1.1 (see Theorem 9 herein). This requires $O(\sqrt{N/K})$ iterations. In the second phase, their algorithm performs a kind of binary search to improve the approximation factor to $1 + \epsilon$; each step of the binary search requires additional Grover iterations, totalling $O(\sqrt{N/K}\,/\,\epsilon)$ in the end.

From the point of view of practicality and simplicity, there is a downside to the Aaronson–Rall algorithm, which is that its Grover iterations are *adaptive* (especially in the second phase of the algorithm). In other words, the steps of the algorithm involve many repetitions of the following: performing some Grover iterations, measuring, and doing some classical computation to decide how many Grover iterations to do in the next step. It has been argued that this repeated switching between quantum and classical computation could be undesirable in practice. Indeed, the final open question in [1] concerned the optimal query complexity of Approximate Counting using *nonadaptive* Grover iterations. This version of the problem was also stressed and studied in [9], but without any provable guarantees being provided.

Other recent developments in the area of approximate counting include [5, 7], which propose variants of the algorithm from [1] but with improved constant factors. As well, the work [4] proves a lower bound for the query complexity of approximate counting in the parallel case.

1.4 Our results

We investigate the problem of Approximate Counting using only nonadaptive Grover iterations. Note that for this version of the problem, there is no hope of obtaining the query complexity $O(\sqrt{N/K}\,/\,\epsilon)$ that improves as a function of K. To see this, suppose even that ϵ is fixed to 1. If the algorithm is to achieve query complexity $O(\sqrt{N/K})$, then it must be able to achieve $O(1)$ query complexity when $K = \Theta(N)$. Since it is nonadaptive, this means it must *always* make only $O(1)$ queries. But this is impossible, as even for *adaptive* algorithms it is known that $\Omega(\sqrt{N})$ queries are required in the case of $K = O(1)$, $\epsilon = 1$.

In other words, with nonadaptive algorithms we can only hope to achieve the optimal query complexity that is independent of K, namely $O(\sqrt{N/\epsilon})$. In this work we indeed show this is achievable. Our main theorem is:

▶ **Theorem 2.** *There is an algorithm for quantum Approximate Counting that uses only* nonadaptive *Grover iterations, and that has a query complexity of $O(\sqrt{N/\epsilon})$ (and minimal additional computational overhead).*

We also briefly sketch an extension of our algorithm achieving improved query complexity in the setting where we are allowed multiple rounds of nonadaptive Grover iterations (as opposed to just one).

2 Preliminaries

We will assume throughout that $K \leq 2^{-20}N$.[3] This without loss of generality since we may artificially replace N by $2^{20}N$ and extend f to $f : [2^{20}N] \to \{0,1\}$, with $f(x) = 0$ for $x > N$. We fix

$$\theta^* = \arcsin\sqrt{K/N},$$

sometimes called the "Grover angle". Recall that a query algorithm based on Grover iterations has the following property: At the cost of $q \in \mathbb{N}$ "queries", it can "flip a coin with bias $\sin^2((2q+1)\theta^*)$". By repeating this t times, it can obtain t independent flips of this coin. It is statistically sufficient to retain only the average of the coin flip outcomes, which is a random variable distributed as $\frac{1}{t}\mathrm{Bin}(t, \sin^2((2q+1)\theta^*))$, where "Bin" denotes a binomial distribution. These observations lead to the following:

▶ **Notation 3.** For real $r \geq 1$ we write $\llbracket r \rrbracket$ for the largest odd integer not exceeding r, and we write $p(r) = \sin^2(\llbracket r \rrbracket \theta^*)$.

▶ **Definition 4.** *A* Grover schedule *consists of two sequences: $R = (r_1, \dots, r_m)$ (each real $r_i \geq 1$) and $T = (t_1, \dots, t_m)$ (each $t_i \in \mathbb{N}^+$).* Performing *this Grover schedule refers to obtaining independent random variables $\widehat{\boldsymbol{p}}_1, \dots, \widehat{\boldsymbol{p}}_m$, where $\widehat{\boldsymbol{p}}_i$ is distributed as $\frac{1}{t_i}\mathit{Bin}(t_i, p(r_i))$.*

A nonadaptive Grover iteration algorithm for Approximate Counting is simply an algorithm that performs one fixed Grover schedule, and produces its final estimate $\widehat{K}$ by classically post-processing the results. One can more generally study algorithms with "s rounds of nonadaptivity"; this simply means that s Grover schedules are used, but they may be chosen adaptively.

▶ **Fact 5.** Performing the Grover schedule R, T uses at most $\frac{1}{2}\sum_i r_i t_i$ queries.

2.1 On how well we need to approximate θ^*

We will use the following elementary numerical fact:

▶ **Lemma 6.** *Suppose for real $0 \leq k, k' \leq N$ and $\eta \leq 1$ that $\arcsin\sqrt{k'/N} \overset{1+\eta}{\approx} \arcsin\sqrt{k/N}$. Then $k' \overset{1+3\eta}{\approx} k$.*

[3] Our work would be fine with, say, $K \leq N/8$, but we put 2^{-20} so as to be able to cite [1] as a black box.

This lemma helps us show that approximating θ^* well is equivalent to approximating K well:

▶ **Proposition 7.** *Suppose $\theta \overset{1+\epsilon/6}{\approx} \theta^*$. If $\kappa' \in \mathbb{R}$ satisfies $\theta = \arcsin\sqrt{\kappa'/N}$, and K' is the nearest integer to κ', then $K' \overset{1+\epsilon}{\approx} K$.*

Proof. Since $\theta \overset{1+\epsilon/6}{\approx} \theta^*$, Lemma 6 tells us that $\kappa' \overset{1+\epsilon/2}{\approx} K$, and hence

$$|\kappa' - K| \leq (\epsilon/2)K. \tag{1}$$

We also have $|\kappa' - K'| \leq 1/2$, and hence $|K' - K| \leq (\epsilon/2)K + 1/2$. But we can assume $(\epsilon/2)K \geq 1/2$, as otherwise Inequality (1) implies $|\kappa' - K| < 1/2$ and hence $K' = K$. Thus $|K' - K| \leq (\epsilon/2)K + (\epsilon/2)K = \epsilon K$; i.e., $K' \overset{1+\epsilon}{\approx} K$. ◀

▶ **Lemma 8.** *Given some $\theta' \overset{1.11}{\approx} \theta^*$, estimating θ^* to a factor of $1+1/(2N\sin^2\theta')$ is sufficient to estimate θ^* to a factor of $1+\epsilon/6$.*

Proof. From Lemma 6, $1 + 1/(2N\sin^2\theta') \leq 1 + 1.33/(2K) < 1 + 1/K$. The closest possible values to K are $K-1$ and $K+1$; therefore, estimating K within a factor of $1+1/K$ is the same as estimating K exactly. This is at least as good as estimating K to within a factor of $1+\epsilon/6$. ◀

3 The nonadaptive algorithm

Our algorithm can conceptually be thought of as having two stages: the first stage estimates θ^* to a constant factor, and the second stage improves this estimate to the desired factor of $1+\epsilon$. This two-stage approach is similar in flavor to the algorithms in [1, 2]. However we note that, consistent with our nonadaptivity condition, the two stages in our algorithm can be run in parallel.

For the first stage of our algorithm, we require the following result of Aaronson and Rall [1], which estimates θ^* up to a factor of 1.1, using $O(\sqrt{N})$ nonadaptive queries. (In fact, as Aaronson and Rall show, the obvious adaptive version of the algorithm incurs only $O(\sqrt{N/K})$ queries.)

▶ **Theorem 9.** *Let $R = (1, (12/11), (12/11)^2, \ldots, (12/11)^m)$, where $m = \Theta(\log N)$ is minimal with $(12/11)^m \geq \sqrt{N}$. Let T consist of m copies of $\lceil 10^5 \ln(120/\delta)\rceil$. Perform the Grover schedule R, T. (By Fact 5 this incurs $O(\sqrt{N})$ queries.) Then except with a failure probability of at most $\delta/2$, we can obtain $\widetilde{\theta} \overset{1.1}{\approx} \theta^*$ by doing the following: take the minimal value of t such that $\widehat{\boldsymbol{p}}_t \geq 1/3$, and set $\widetilde{\theta} = (5/8)(11/12)^t$.*

The second stage of our algorithm uses the following critical lemma, which we will prove in Section 4.

▶ **Lemma 10.** *Given the parameters θ', ϵ', and δ', there is an algorithm using only nonadaptive Grover iterations that performs $O(\log(1/\delta')/(\theta'\epsilon'))$ queries, and outputs a result θ_{est} with the following guarantee: if $\theta' \overset{1.11}{\approx} \theta^*$, then $\theta_{\text{est}} \overset{1+\epsilon'/6}{\approx} \theta^*$ except with probability at most $\delta/2$.*

In this section, we will show how to use Theorem 9 and Lemma 10 to prove Theorem 2. We will now state our algorithm:

Algorithm 1 Outline of the full algorithm.

1. Run the Aaronson–Rall algorithm from Theorem 9, allowing us to later compute $\widetilde{\theta}$.
2. For $\theta = \arcsin(\sqrt{1/N}), 1.001 \arcsin \sqrt{1/N}, (1.001)^2 \arcsin \sqrt{1/N}, \ldots, 1.1 \arcsin(2^{-20})$:
   Perform the algorithm in Lemma 10 with the parameters $\theta' = \theta$, $\epsilon' = \max(\epsilon, \frac{1}{2N \sin^2 \theta'})$, and $\delta' = \delta/2$.
3. Classical Post-processing: Among all iterations in the for-loop, take the iteration with the value of θ that was closest to $\widetilde{\theta}$, and output the result of that iteration.

Each iteration of the for loop in Step 2 can be done in parallel (there are no computational dependencies between the iterations), and Step 1 can also be done in parallel with Step 2. Therefore, the algorithm uses only nonadaptive Grover iterations. Note also that we can write the quantum parts of steps 1 and 2 as one fixed Grover schedule, with the classical parts and step 3 forming the post-processing step; however, it will be more convenient in this section to think about these as individual steps in a logical sequence.

▶ **Proposition 11.** *Algorithm 1 returns a value θ_{est} such that $\theta_{\text{est}} \overset{1+\epsilon}{\approx} \theta^*$, except with probability at most δ.*

Proof. By Theorem 9, Step 1 returns an estimate $\widetilde{\theta}$ such that $\widetilde{\theta} \overset{1.1}{\approx} \theta^*$, with a failure probability of at most $\delta/2$. The value of θ (in Step 2) that is closest to $\widetilde{\theta}$ is at most a factor of 1.001 away from $\widetilde{\theta}$ (note that $\widetilde{\theta}$ is at most $1.1 \arcsin(2^{-20})$ by our assumption that $\theta^* \leq 2^{-20}$). If Step 1 succeeded, this is at most a factor of $1.1 \times 1.001 < 1.11$ away from θ^*. By Lemma 10, the algorithm outputs an estimate θ_{est} such that $\theta_{\text{est}} \overset{1+\epsilon'/6}{\approx} \theta^*$, with a failure probability of at most $\delta/2$. Lemma 8 then implies that $\theta_{\text{est}} \overset{1+\epsilon/6}{\approx} \theta^*$. Using Proposition 7 (setting the parameter $\theta = \theta_{\text{est}}$), we get an estimate K_{est} such that $K_{\text{est}} \overset{1+\epsilon}{\approx} K$. By the union bound, the overall failure probability of the algorithm is at most δ. ◀

▶ **Proposition 12.** *Algorithm 1 makes $O(\sqrt{N/\epsilon} \log(1/\delta))$ queries.*

Proof. First, consider all iterations where $2N \sin^2(\theta) \leq 1/\epsilon$. In these cases, the query complexity given by Lemma 10 would be $O(\log(1/\delta)(N \sin^2(\theta))/\theta) = O(N\theta \log(1/\delta))$.

The query complexity associated with these iterations is a geometric series with a constant common ratio of 1.01 where the largest term is $O(\sqrt{N/\epsilon} \log(1/\delta))$. Therefore the overall query complexity due to these iterations is $O(\sqrt{N/\epsilon} \log(1/\delta))$.

Now consider all iterations where $2N \sin^2(\theta) > 1/\epsilon$. In these cases, the query complexity is $O(\log(1/\delta)/(\theta\epsilon))$. This forms a geometrically decreasing series (with a constant common ratio), where the first term is again $O(\sqrt{N/\epsilon} \log(1/\delta))$. The overall query complexity contributed by these schedules is thus also $O(\sqrt{N/\epsilon} \log(1/\delta))$.

Therefore, the query complexity of Algorithm 1 is $O(\sqrt{N/\epsilon} \log(1/\delta))$, as claimed. ◀

Having proven Proposition 11 and Proposition 12, we have established our main result Theorem 2 modulo the proof of Lemma 10, which will appear in the next section. Before giving this, we briefly sketch how our algorithm can also be extended to the setting of being allowed multiple rounds of nonadaptive Grover iterations. If we have two such rounds of nonadaptivity, we can first run step 1 of our algorithm to get a constant-factor approximation, and then based on its result run the algorithm in Lemma 10; this achieves a query complexity of $O(\sqrt{N} + \min(\sqrt{N/\epsilon}, \sqrt{N/K}/\epsilon))$. This nearly matches the query complexity of the fully adaptive case, but for the $\sqrt{N}$ term due to the first step. Given more rounds of nonadaptivity, we can reduce the cost of this first step by staging it over multiple initial rounds. One can show that with $O(\log N)$ rounds of nonadaptivity, this will yield the optimal query complexity corresponding to the fully adaptive case.

4 Proving Lemma 10

Our goal in this section will be to prove Lemma 10. Assume that we are given some θ', ϵ', and δ'. We will show a nonadaptive Grover iteration algorithm making $O(\log(1/\delta')/(\theta'\epsilon'))$ queries with the property that if $\theta' \overset{1.11}{\approx} \theta^*$, then its output will be a factor-$(1+\epsilon')$ approximation of θ^* (except with failure probability at most δ'). For the remainder of the section, we will assume that we are in the interesting case where $\theta' \overset{1.11}{\approx} \theta^*$ (in the other cases, the algorithm does not need to output a correct answer).

4.1 Proof idea

The algorithm for Lemma 10 is structured exactly as described in Definition 4 and Fact 5; there is an initial nonadaptive quantum part with a fixed Grover schedule (that we will later define), and a classical post-processing step at the end that uses the results of the quantum part to estimate θ^*.

Before stating the key ideas in the quantum part of our algorithm, we mention the "Rotation Lemma" of Aaronson and Rall [1, Lem. 2]. The main idea in that lemma can be roughly stated as follows: given that θ^* lies in some range $[\theta_{\min}, \theta_{\min} + \Delta\theta]$, we can pick an odd integer value of r (where $r = O(1/(\theta \cdot \Delta\theta))$), such that $r\theta_{\min}$ is close to $2\pi k$ and $r(\theta_{\min} + \Delta\theta)$ is close to $2\pi k + \pi/2$. If θ is close to $\theta_{\min}$, $p(r)$ will be nearly 0 (and if it is close to $\theta_{\min} + \Delta\theta$, it will be nearly 1). Aaronson and Rall use this lemma to continually shrink the possible range that θ^* could lie in by a geometric factor at each iteration, until the range is $1 \pm \epsilon$.

We will adopt a similar idea to find an efficient Grover schedule that can distinguish any two candidate angles with high probability; we do this by relaxing the condition of one angle being close to $2\pi k$ and the other being at distance $\pi/2$ from it. Instead, we choose the sequence R in our Grover schedule such that for any pair of values θ_1, and θ_2, there is some $r \in R$ such that $r\theta_1$ and $r\theta_2$ differ by approximately $\pi/8$, and are also "in the same quadrant" (meaning the same interval $[0, \frac{\pi}{2})$, $[\frac{\pi}{2}, \pi)$, $[\pi, \frac{3\pi}{2})$, $[\frac{3\pi}{2}, 2\pi)$ modulo 2π). This relaxation allows us to save on the total number of queries made by reusing the same value of r to distinguish many pairs of candidate angles. Due to this, the nonadaptivity requirement does not make the query complexity grow polynomially larger (whereas, for example, naively simulating the search tree from [1] in a nonadaptive fashion would incur an extra $1/\epsilon'$ factor).

The classical post-processing involves running a "tournament" between all candidate estimates of θ^*, which outputs the winning value as the estimate. This post-processing step can be implemented efficiently in $O(\log(1/\epsilon')/\epsilon')$ classical time.

4.2 Some arithmetic lemmas

We now define some useful sequences and prove a couple of arithmetic lemmas about them.

▶ **Definition 13.** *Define the sequence u by $u_0 = 1$, $u_1 = 1.01$, …, $u_L = 1.01^L$ where $L = O(\log(1/(\theta'\epsilon')))$ is minimal with $u_L \geq 1.2\pi/(\theta'\epsilon')$.*

▶ **Lemma 14.** *Suppose we are given θ_0 and θ_1 such that $\theta_0 < \theta_1$, $\theta_0 \overset{1.11}{\approx} \theta'$, $\theta_1 \overset{1.11}{\approx} \theta'$, and $\theta_0 \overset{1+\epsilon/6.1}{\not\approx} \theta_1$. Write $\eta = \theta_1 - \theta_0$. Then there exists some $0 \leq i \leq L$ such that $u_i\eta \overset{1.01}{\approx} \frac{\pi}{8}$.*

Proof. We know that $u_0\eta = \eta \leq \theta_1 \leq 1.1 \times 0.0001 < \frac{\pi}{8}$. We also have $u_L\eta \geq 1.2\pi\eta/(\theta'\epsilon') \geq 1.2\pi\eta/(1.1\theta_0\epsilon') \geq 1.2\pi/(1.1 \cdot 6.1) > \frac{\pi}{8}$ where we used $\theta_0 \overset{1+\epsilon/6.1}{\not\approx} \theta_1$ in the second-to-last inequality. The lemma now follows from the geometric growth of the u_i's with ratio 1.01. In particular, $i = \left\lfloor \log_{1.01}(\frac{\pi}{8\eta}) \right\rfloor$ works. ◀

For the u_i given by Lemma 14, we have $u_i\theta_1 - u_i\theta_0 \approx \frac{\pi}{8}$. This seems promising, in that the "coin probabilities" associated to these angles, namely $\sin^2(u_i\theta_1)$ and $\sin^2(u_i\theta_0)$, seem as though they should be far apart. Unfortunately, something annoying could occur; it could be that these angles are, say, $100\pi \pm \frac{\pi}{16}$, in which case the coin probabilities would be identical. As mentioned in Section 4.1, what we would *really* like is to have the two angles be far apart but also in the same quadrant. To achieve this, we will define a new sequence.

▶ **Definition 15.** *For each $0 \leq i \leq L$, define $a_{i,0} = 0$, $a_{i,1} = \frac{\pi}{4.8\theta'}$, $a_{i,2} = 1.01 \cdot \frac{\pi}{4.8\theta'}$, $a_{i,3} = 1.01^2 \cdot \frac{\pi}{4.8\theta'}$, $\ldots$, $a_{i,C+1} = 1.01^C \cdot \frac{\pi}{4.8\theta'}$, where $C = \lceil 2\log_{1.01}(1.2) \rceil$ is a constant. Also define $s_{i,j} = u_i + a_{i,j}$.*

▶ **Lemma 16.** *In the setting of Lemma 14, there exists some $0 \leq j \leq C+1$ such that*

$$s_{i,j}\eta \overset{1.5}{\approx} \frac{\pi}{8}$$

and such that $s_{i,j}\theta_0$ and $s_{i,j}\theta_1$ are in the same quadrant.

Proof. We first apply Lemma 14 and obtain

$$u_i\eta \overset{1.01}{\approx} \frac{\pi}{8}. \tag{2}$$

Now if $u_i\theta_0$ and $u_i\theta_1$ are already in the same quadrant then we can take $j = 0$ (implying $s_{i,j} = u_i$) and we are done. Otherwise, the plan will be to find $j > 0$ with $a_j\theta_0 \approx \frac{\pi}{4}$, thus shifting them to $s_{i,j}\theta_0$ and $s_{i,j}\theta_1$ that still differ by roughly $\frac{\pi}{8}$ but which now must be in the same quadrant.

To find the required j, observe that on one hand, $a_1\theta_0 = (\pi/(4.8\theta')) \cdot \theta_0 \leq 1.11\pi/4.8 \leq \frac{\pi}{4}$. On the other hand, $a_{i,C+1}\theta_0 \geq 1.2^2 \cdot (\pi/(4.8\theta')) \cdot \theta_0 \geq (1.2/1.1) \cdot \pi/4 \geq \frac{\pi}{4}$. By the geometric growth of the $a_{i,j}$'s with ratio 1.01, we conclude that there exists some $1 \leq j \leq \ell_i$ achieving

$$a_{i,j}\theta_0 \overset{1.05}{\approx} \frac{\pi}{4}. \tag{3}$$

We may now make several deductions. First,

$$\theta_0 \overset{1.1}{\approx} \theta_t, \theta_1 \overset{1.1}{\approx} \theta_t \implies a_{i,j}\theta_0 \overset{1.22}{\approx} a_{i,j}\theta_1 \implies a_{i,j}\eta \leq .22 \cdot a_{i,j}\theta_0 \leq .22 \cdot 1.05\tfrac{\pi}{4} \leq .24\tfrac{\pi}{4}.$$

Combining this with Inequality (2) we conclude

$$s_{i,j}\eta = u_i\eta + a_{i,j}\eta \in [\tfrac{1}{1.01}\tfrac{\pi}{8}, \tfrac{\pi}{8}(1.01) + .24\tfrac{\pi}{4}] \overset{1.5}{\approx} \frac{\pi}{8}. \tag{4}$$

Thus we started with $u_i\theta_0$ and $u_i\theta_1$ differing by $\frac{\pi}{8}$ (up to factor 1.01) but in different quadrants; by passing to $s_{i,j}\theta_0$ and $s_{i,j}\theta_1$, we have offset $u_i\theta_0$ by $\frac{\pi}{4}$ (up to factor 1.05, Inequality (3)) and the two angles still differ by around $\frac{\pi}{8}$ (up to factor 1.5, Inequality (4)). Thus $s_{i,j}\theta_0$ and $s_{i,j}\theta_1$ are in the same quadrant and the proof is complete. ◀

4.3 The algorithm

We can now describe our "second stage" algorithm, which simply runs the Grover schedule G defined as follows.

▶ **Definition 17.** *The Grover schedule G comprises the sequence $R = (s_{i,j})_{i=0\dots L,\ j=0\dots C+1}$ and $T = (\lceil A \log_2(1/(\delta'\theta'\epsilon' u_i))\rceil)_{i=0\dots L,\ j=0\dots C+1}$. Here A is a universal constant to be chosen later.*

Note that the $T_{i,j}$ values we use are exactly the number of coin flips used in the second stage of the algorithm in [1]. Like in their algorithm, this choice of values allows us to avoid stray $\log(1/\epsilon)$ or $\log\log(1/\epsilon)$ factors in the overall query complexity.

▶ **Proposition 18.** *Performing the Grover schedule G takes at most $O(\log(1/\delta')/(\theta'\epsilon'))$ queries.*

Proof. Using Fact 5, the query complexity of performing G is $\sum_{i=0}^{L}\log(1/(\delta'\theta'\epsilon'\cdot 1.01^i))1.01^i$ (up to constant factors). This is $\log(1/\delta)\sum_{i=0}^{L}1.01^i + \sum_{i=0}^{L}\log(1/(\theta'\epsilon'\cdot 1.01^i))1.01^i$. Noting that $1.01^L = O(1/(\theta'\epsilon'))$, the first term is clearly $O(\log(1/\delta)/(\theta'\epsilon'))$ and the second term, up to a constant factors, is $\sum_{i=0}^{L}(L-i)1.01^i = O(1.01^L) = O(1/(\theta'\epsilon'))$. Therefore, the overall query complexity is $O(\log(1/\delta)/(\theta'\epsilon'))$ as desired. ◀

▶ **Remark.** The above calculation mirrors the one done for stage 2 of the adaptive algorithm in [1]; this is expected because both algorithms use the same number of "coin flips" per coin ($T'_{i,j}$ values), as mentioned above.

It now remains for us to show how to approximate θ^* (with high probability) using the data collected from this Grover schedule.

4.4 Completing the algorithm

We first prove a lemma showing that we can distinguish between any pair of angles (that are not already sufficiently close to each other) by using the ideas developed in Lemma 14 and Lemma 19.

▶ **Lemma 19.** *There is an $O(1)$-time classical deterministic algorithm that, given*

- *$\theta_0 \overset{1.1}{\approx} \theta'$, $\theta_1 \overset{1.1}{\approx} \theta'$, such that $\theta_0 \overset{1+\epsilon/6.1}{\not\approx} \theta_1$*
- *the data collected by the Grover schedule G,*

outputs either "reject θ_0" or "reject θ_1". Except with failure probability at most $c\theta'\delta'\epsilon'/|\theta_1-\theta_0|$ (where $c>0$ is a constant to be chosen later), the following is true:

> *For $b=0,1$, if $\theta^* \overset{1+.001\epsilon}{\approx} \theta_b$, then the algorithm does not output "reject θ_b".*

Proof. The algorithm computes the (i,j) pair promised by Lemma 16, such that $s_{i,j}\theta_0$ and $s_{i,j}\theta_1$ are in the same quadrant and such that $|s_{i,j}\theta_1 - s_{i,j}\theta_0| \overset{1.5}{\approx} \frac{\pi}{8}$. Letting $q_b = \sin^2(s_{i,j}\theta_b)$ for $b=0,1$, it follows from the assumptions in the preceding sentence that $|q_0-q_1| \geq .04$. The algorithm may now select a threshold $q' \in [.01,.99]$ such that (without loss of generality) $q_0 \leq q' - .01$ and $q_1 \geq q' + .01$.

The algorithm will use just the coin flips from the $s_{i,j}$ part of the schedule; these coin flips have bias $p(s_{i,j}) = \sin^2(\llbracket s_{i,j}\rrbracket\theta^*)$. More precisely, the algorithm will output "reject θ_0" if $\widehat{\boldsymbol{p}}_{i,j} > q'$ and "reject θ_1" if $\widehat{\boldsymbol{p}}_{i,j} \leq q'$. We need to show that if $\theta^* \overset{1+.001\epsilon}{\approx} \theta_0$ then the algorithm outputs "reject θ_0" with probability at most $c\theta'\delta'\epsilon'/|\theta_1-\theta_0|$. (The case when $\theta^* \overset{1+.001\epsilon}{\approx} \theta_1$ is analogous.)

Now if $\theta^* \overset{1+.001\epsilon}{\approx} \theta_0$, then $\llbracket s_{i,j} \rrbracket \theta^* \overset{1+.001\epsilon}{\approx} \llbracket s_{i,j} \rrbracket \theta_0$. It follows that

$$\left| \llbracket s_{i,j} \rrbracket \theta^* - \llbracket s_{i,j} \rrbracket \theta_0 \right| \leq .001\epsilon \cdot \llbracket s_{i,j} \rrbracket \theta_0 \leq .001 s_{i,j} \cdot \epsilon\theta_0.$$

But we know that

$$\tfrac{\pi}{8} \overset{1.5}{\approx} |s_{i,j}\theta_1 - s_{i,j}\theta_0| = s_{i,j}|\theta_1 - \theta_0| \geq s_{i,j} \cdot (\epsilon/6.1)\theta_0,$$

the last inequality because $\theta_0 \overset{1+\epsilon/6.1}{\not\approx} \theta_1$. Combining the above two deductions yields

$$\left| \llbracket s_{i,j} \rrbracket \theta^* - \llbracket s_{i,j} \rrbracket \theta_0 \right| \leq .001 \cdot 1.5 \cdot 6.1 \cdot \tfrac{\pi}{8} \leq .004.$$

Moreover, $\llbracket s_{i,j} \rrbracket \theta_0$ and $s_{i,j}\theta_0$ differ by at most $2\theta_0 \leq .0002 < .001$. Thus we finally conclude

$$\left| \llbracket s_{i,j} \rrbracket \theta^* - s_{i,j}\theta_0 \right| \leq .005 \quad \Longrightarrow \quad \left| p(s_{i,j}) - q_0 \right| \leq .005 \quad \Longrightarrow \quad p(s_{i,j}) < q' - .005$$

(the first implication using that $\sin^2$ is 1-Lipschitz). Then, using a Chernoff bound, we have that $\widehat{\boldsymbol{p}}_{i,j} > q'$ with probability at most $c\delta'\theta'\epsilon' u_i$, where c is a constant that depends on A (as A increases, c decreases). From Lemma 14, we know that $u_i \overset{1.01}{\approx} \pi/(8|\theta_1 - \theta_0|)$. Then, assuming the constant A is chosen large enough as a function of c, we indeed have that $\widehat{\boldsymbol{p}}_{i,j} > q'$ with probability at most $c\theta'\delta'\epsilon'/|\theta_1 - \theta_0|$. ◀

4.4.1 Description of the post-processing algorithm

We have developed the necessary tools to describe and justify the classical post-processing algorithm.

Fix values $\theta_i = \theta'(1 + .001\epsilon')^i$ for $-V \leq i \leq V$, where $V = O(1/\epsilon')$ is a minimal integer such that $(1 + .001\epsilon')^V \geq 1.11$. We will refer to the θ_i's as "nodes". We may also assume that the number of nodes is a power of 2 for convenience (while there are $2|V| + 1$ nodes, we can always pad these actual nodes with some dummy nodes to reach the nearest power of 2).

The main idea is to repeatedly use Lemma 19 to run a "tournament" amongst all nodes. The tournament is structured as a series of "rounds". In a given round, suppose we start off with n nodes. Sort the nodes in order of the angles they correspond to. Now, pair up node 1 with node $n/2 + 1$, node 2 with node $n/2 + 2$, and so on, until node $n/2$ is paired up with node n. For each pair of nodes, use Lemma 19 to choose a winner to go to the next round. Note that it is possible that two nodes that are matched in the tournament do not satisfy the pre-condition of Lemma 19 because they are within a factor of $(1 + \epsilon/6.1)$ of each other – we call these "void" match-ups. We will call the part of the tournament we have described so far the first phase – when we see a void match-up, we stop this first phase and enter the second phase. (Note that if we never see any void match-ups and there is only one node left, we do not need the second phase and can directly output the remaining node as our estimate.)

When the first phase ends, take all remaining nodes and enter the second phase. In this phase, match up every pair of remaining nodes, and for every pair that does not form a void match-up, eliminate one of the nodes using Lemma 19. At the end of this, output any one of the remaining un-eliminated nodes (if there are none, then the program can output an arbitrary node - this is a failure condition and we will show that, with high probability, such failure conditions will not happen).

In our algorithm, we have arranged for $\theta_{-V} \leq \theta^* \leq \theta_V$, and hence there exists a node θ_{i^*} such that $\theta_{i^*} \overset{1+.001\epsilon}{\approx} \theta^*$. We will proceed to bound the overall failure probability of the algorithm by the probability that this node loses any match-up it is a part of.

▶ **Lemma 20.** *If θ_{i^*} never loses any match-up it is a part of, then the tournament outputs an estimate θ_{est} such that $\theta_{\text{est}} \overset{1+\epsilon/6}{\approx} \theta^*$.*

Proof. Suppose that θ_{i^*} never loses any match-up it is a part of. If the tournament ends in the first phase itself, then the algorithm will output θ_{i^*}, which is correct. If the tournament ends in the second phase, then the only possible other nodes that we could output are the ones that θ_{i^*} did not play, which can be at most a factor of $1+\epsilon/6.1$ away from it. Therefore, these nodes are also at most a factor of $(1+0.001\epsilon)(1+\epsilon/6.1) \leq 1+\epsilon/6$ away from θ^*. ◀

▶ **Lemma 21.** *The probability that θ_{i^*} loses any match-up in the first phase of the tournament can be upper-bounded by $c\delta'$ times a constant factor (where c is the constant from Lemma 19).*

Proof. Consider an arbitrary round in the first phase of the tournament, where we have 2^j nodes remaining for some j. By how we chose the match-ups, we know that every two angles that are matched up in that round must be at least a factor of $(1+0.001\epsilon')^{2^{i-1}}$ apart. This implies that their absolute difference is at least $(\theta'/1.1)\cdot((1+0.001\epsilon')^{2^{i-1}}-1)$. Then, by Lemma 19, the failure probability of any match-up in that round is at most $1.1c\delta'\epsilon'/((1+0.001\epsilon')^{2^{i-1}}-1)$.

Suppose the tournament begins with n nodes. We can use the union bound to upper-bound the probability that θ_{i^*} loses in any of the at most $\log_2 n$ rounds by

$$\sum_{j=1}^{\log_2 n} \frac{1.1c\delta'\epsilon'}{(1+0.001\epsilon')^{2^{i-1}}-1} \leq \sum_{j=1}^{\log_2 n} \frac{1.1c\delta'\epsilon'}{0.001\epsilon 2^{i-1}} \leq 2200c\delta'$$ ◀

▶ **Lemma 22.** *The probability that θ_{i^*} loses any match-up in the second phase of the tournament can be upper-bounded by $c\delta'$ times a constant factor (where c is the constant from Lemma 19).*

Proof. If the algorithm enters the second phase, this means that there is at least one void match-up. Let the number of nodes at this point be n. Then all $n/2$ nodes between the pair of nodes involved in the void match-up must be within a factor of $(1+\epsilon'/6.1)$ of the first node in the void pair. Therefore, there are at most $2\log_{1+0.001\epsilon'}(1+\epsilon'/6.1) \leq 12200$ nodes in total.

Every node that θ_{i^*} plays in a match-up is at least a factor of $1+\epsilon'/6$ away from it, which means that the absolute difference in value between the nodes is at least $(\theta'/1.1)\cdot(\epsilon'/6)$. By Lemma 19, this implies that the failure probability is at most $6.6c\delta'$. Since there are at most 12200 such match-ups, the overall failure probability is at most $81000c\delta'$ by the union bound. ◀

We can now prove Lemma 10, which we restate below.

▶ **Lemma 10.** Given the parameters θ', ϵ', and δ', there is a nonadaptive algorithm that performs $O(\log(1/\delta')/(\theta'\epsilon'))$ queries, and outputs a result θ_{est} with the following guarantee: if $\theta' \overset{1.11}{\approx} \theta^*$, then $\theta_{\text{est}} \overset{1+\epsilon'/6}{\approx} \theta^*$ except with probability at most $\delta/2$.

Proof. The algorithm achieving this lemma involves running the Grover schedule G and then post-processing using the tournament algorithm which we described. By Lemma 20, the failure probability of the algorithm is bounded by the probability that θ_{i^*} loses. This is bounded by the probability that it loses in the first phase or the second phase; by Lemma 21 and Lemma 22, this is at most a constant times $c\delta'$. By choosing c to be sufficiently small, we can successfully make this at most $\delta/2$. ◀

References

1 Scott Aaronson and Patrick Rall. Quantum approximate counting, simplified. In *Symposium on Simplicity in Algorithms*, pages 24–32. SIAM, 2020.

2 Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik: Progress of Physics*, 46(4-5):493–505, 1998.

3 Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.

4 Paul Burchard. Lower bounds for parallel quantum counting, 2019. arXiv:1910.04555.

5 Dmitry Grinko, Julien Gacon, Christa Zoufal, and Stefan Woerner. Iterative quantum amplitude estimation, 2020. arXiv:1912.05559.

6 Lov Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 212–219, 1996.

7 Kouhei Nakaji. Faster amplitude estimation, 2020. arXiv:2003.02417.

8 Ashwin Nayak and Felix Wu. The quantum query complexity of approximating the median and related statistics. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 384–393, 1999.

9 Yohichi Suzuki, Shumpei Uno, Rudy Raymond, Tomoki Tanaka, Tamiya Onodera, and Naoki Yamamoto. Amplitude estimation without phase estimation. *Quantum Information Processing*, 19(2):75, 2020.

10 Chu-Ryang Wei. Simpler quantum counting. *Quantum Information and Computation*, 19(11&12):0967–0983, 2019.